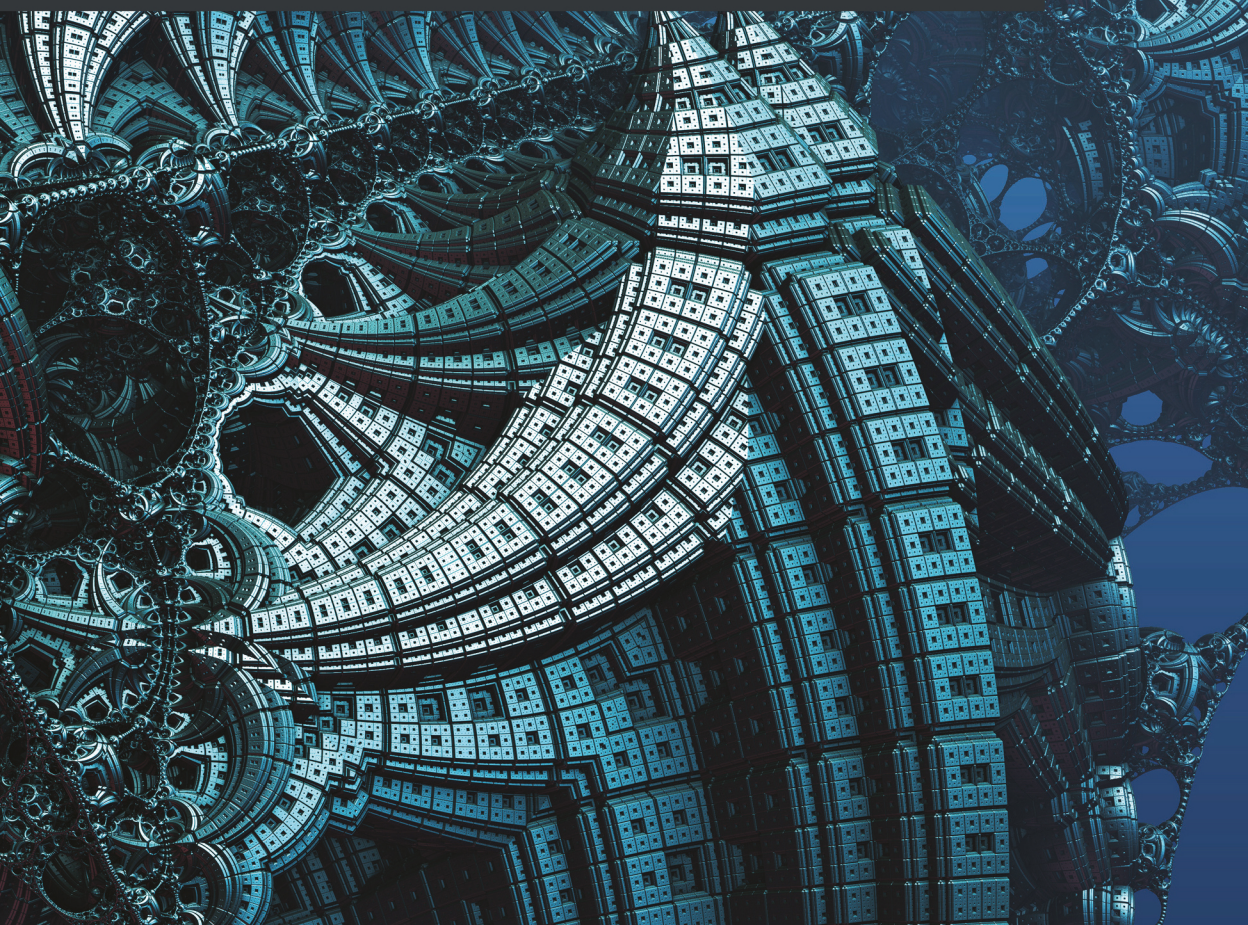


# Устройство и программирование автономных роботов

Проекты на Python и Raspberry Pi



Дэнни Стейпл

**АМК**  
ИЗДАТЕЛЬСТВО

Дэнни Стейпл

# **Устройство и программирование автономных роботов**



# Learn Robotics Programming

Build and control AI-enabled autonomous robots using the Raspberry Pi and Python

**Danny Staple**



BIRMINGHAM—MUMBAI

# Устройство и программирование автономных роботов

Проекты на Python и Raspberry Pi

Дэнни Стейпл



Москва, 2022



УДК 004.896  
ББК 32.816  
С79

**Дэнни Стейпл**

**С79** Устройство и программирование автономных роботов / пер. с англ. Е. В. Шевчук; науч. ред. В. С. Яценков. – М.: ДМК Пресс, 2022. – 520 с.: ил.

**ISBN 978-5-97060-989-7**

Эта книга посвящена созданию интеллектуального робота и разработке кода для его поведенческих сценариев. Для построения робота используются широко доступные компоненты – датчики, двигатели, камеры, микрофоны, динамики, светодиоды и микрокомпьютер Raspberry Pi. Раскрывается ряд специализированных тем, таких как компьютерное зрение и голосовое управление. Также читатель узнает о специализированных сообществах, посвященных робототехнике, и перспективах ее развития.

Книга подойдет как новичкам в области программирования, так и опытным программистам, желающим применить свои навыки в аппаратном проекте. Для изучения материала необходимо знание языка Python и умение работать с циклами, условиями и функциями.

УДК 004.896  
ББК 32.816

First published in the English language under the title 'Learn Robotics Programming- Second Edition – (9781839218804)

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN (анг.) 978-1-83921-880-4  
ISBN (рус.) 978-5-97060-989-7

Copyright ©Packt Publishing 2021. First  
© Оформление, издание, перевод, ДМК Пресс, 2022

# Оглавление

<b>Участники .....</b>	<b>17</b>
Об авторе.....	17
О рецензентах.....	17
<b>Предисловие.....</b>	<b>19</b>
Для кого эта книга .....	19
Какие темы описаны в книге .....	19
Как извлечь из книги наибольшую пользу .....	21
Загрузка файлов с примерами кода .....	21
Code in Action .....	22
Изображения в цвете.....	22
Используемые обозначения .....	22
Оставайтесь на связи.....	23
Отзывы .....	23
<b>ЧАСТЬ 1. ВВЕДЕНИЕ – ОСНОВЫ РОБОТОТЕХНИКИ... 25</b>	
<b>Глава 1. Введение в робототехнику.....</b>	<b>27</b>
Что такое робот? .....	27
Продвинутые и впечатляющие роботы .....	29
Марсоходы .....	31
Роботы в доме .....	32
Стиральная машина .....	32
Другие домашние роботы.....	33
Роботы в промышленности .....	34
Роботы-манипуляторы.....	34
Роботы на складах .....	36
Роботы для участия в соревнованиях, учебные и любительские роботы.....	36
Выводы .....	39
Задание.....	40
Дополнительные материалы .....	40
<b>Глава 2. Структурные элементы робота – код и электроника... 42</b>	
Технические условия .....	42
Внутреннее устройство робота.....	42
Типы компонентов робота.....	46
Типы двигателей .....	46
Другие типы приводов .....	48



Индикаторы состояния – дисплеи, световые и звуковые индикаторы.....	49
Типы сенсоров .....	49
Контроллеры и устройства ввода/вывода.....	51
Контакты ввода/вывода .....	52
Контроллеры.....	53
Модели контроллера Raspberry Pi .....	56
Набросок компонентов и структуры кода .....	57
Разработка аппаратного устройства робота.....	59
Выводы .....	62
Упражнения .....	62
Дополнительные материалы .....	62
<b>Глава 3. Изучение Raspberry Pi .....</b>	<b>63</b>
Технические требования.....	63
Функциональные возможности Raspberry Pi.....	63
Скорость и производительность .....	64
Возможности подключения.....	64
Преимущества Raspberry Pi 3A+ .....	64
Выбор способов подключения.....	65
Контакты для подключения питания.....	66
Шины передачи данных.....	67
Входы/выходы общего назначения.....	67
Платы расширения HAT для Raspberry Pi .....	67
Операционная система Raspberry Pi .....	68
Запись образа ОС Raspberry Pi на SD-карту.....	68
Выводы .....	70
Задание.....	71
Дополнительные материалы: .....	71
<b>Глава 4. Автономное управление роботом с помощью Raspberry Pi.....</b>	<b>72</b>
Технические требования.....	72
Сущность автономных систем и их преимущества .....	73
Настройка Wi-Fi и SSH на Raspberry Pi.....	74
Поиск Raspberry Pi в сети .....	76
Установка Bonjour для Windows.....	76
Тест системы .....	77
Устранение неполадок .....	77
Подключение к Raspberry Pi посредством PuTTY и SSH.....	78
Настройка конфигурации ОС Raspberry Pi .....	80
Изменение имени Raspberry Pi.....	80
Защита Raspberry Pi.....	82
Перезагрузка и повторное подключение .....	82
Обновление программного обеспечения на Raspberry Pi .....	85
Завершение сеанса Raspberry Pi .....	86
Выводы .....	86

Задание.....	87
Дополнительные материалы .....	87

## **Глава 5. Создание резервных копий**

### **с помощью Git и SD-карты ..... 88**

Технические требования.....	88
Причины возникновения проблем при написании и изменении кода .....	89
Повреждение SD-карты и потеря данных .....	89
Изменение кода или конфигурации .....	89
Стратегия 1 – хранение кода на ПК и его последующая передача .....	90
Стратегия 2 – создание копий с помощью Git.....	92
Стратегия 3 – резервное копирование данных SD-карты .....	94
Windows.....	95
Mac.....	97
Linux .....	100
Выводы .....	101
Задание.....	101
Дополнительные материалы .....	102

## **ЧАСТЬ 2. СОЗДАНИЕ АВТОНОМНОГО РОБОТА – ПОДКЛЮЧЕНИЕ ДАТЧИКОВ И ДВИГАТЕЛЕЙ К RASPBERRY PI ..... 103**

### **Глава 6. Сборка основания – колеса, питание**

#### **и электропроводка ..... 105**

Технические требования.....	105
Выбор набора шасси.....	106
Размер .....	106
Количество колес .....	107
Колеса и двигатели .....	108
Простота .....	109
Стоимость .....	110
Заключение .....	110
Выбор платы контроллера двигателя.....	110
Степень интеграции.....	111
Использование контактов.....	111
Размер .....	112
Пайка .....	113
Силовой источник питания .....	113
Способы подключения .....	113
Заключение .....	114
Питание робота.....	115
Проверка компоновки.....	117
Сборка основания.....	120



Установка диска энкодера.....	122
Установка двигателя с помощью опор.....	122
Установка поворотного колеса .....	124
Установка ведущих колес .....	125
Прокладка проводов.....	126
Установка Raspberry Pi .....	127
Установка источников питания.....	128
Готовое основание для робота.....	130
Подключение двигателей к Raspberry Pi.....	131
Подключение проводов к HAT-плате .....	132
Автономное питание .....	134
Выводы .....	136
Задание.....	137
Дополнительные материалы .....	137

## **Глава 7. Движение и повороты – код на Python**

### **для управления двигателями .....138**

Технические требования.....	138
Разработка кода для проверки двигателей .....	139
Подготовка библиотек.....	139
Тест – обнаружение HAT-платы двигателя .....	139
Тест – демонстрация работы двигателей.....	141
Устранение неполадок .....	142
Как работает код .....	142
Рулевое управление робота .....	144
Типы рулевого управления .....	144
Рулевое управление нашего робота .....	147
Создание объекта Robot – код для взаимодействия с роботом.....	148
Для чего нужен объект?.....	149
Что мы поместим в объект?.....	150
Разработка сценария для следования по заданной траектории.....	154
Выводы .....	157
Задание.....	157
Дополнительные материалы .....	157

## **Глава 8. Код на Python для работы с датчиками расстояния.....158**

Технические требования.....	158
Выбор между оптическими и ультразвуковыми датчиками .....	159
Оптические датчики.....	160
Ультразвуковые датчики.....	160
Логические уровни и трансляторы логических уровней.....	161
Зачем использовать два датчика? .....	164
Подключение ультразвуковых датчиков и обработка получаемых ими данных .....	165
Установка датчиков на шасси .....	165
Подключение выключателя питания.....	167

Подключение датчиков расстояния .....	170
Установка библиотек Python для работы с датчиками .....	172
Считывание данных ультразвуковых датчиков расстояния .....	172
Устранение неполадок .....	175
Обход стен – разработка сценария обхода препятствий .....	176
Добавление датчиков в класс Robot .....	176
Разработка сценария обхода препятствий .....	177
Первая попытка разработки стратегии обхода препятствий.....	178
Разработка более сложной стратегии обхода препятствий.....	182
Выводы .....	184
Задание.....	184
Дополнительные материалы .....	185
<b>Глава 9. Код на Python для работы со светодиодами .....</b>	<b>186</b>
Технические требования .....	186
Что такое линейка RGB-светодиодов? .....	187
Сравнение технологий светодиодной индикации .....	187
RGB-значения .....	189
Подключение светодиодной линейки к Raspberry Pi.....	190
Установка светодиодной линейки на работа .....	190
Разработка кода для управления дисплеем робота .....	191
Создание интерфейса для управления светодиодами .....	191
Добавляем светодиоды в объект Robot .....	193
Тест одного светодиода .....	194
Тест всех светодиодов .....	196
Реализация радужного свечения на светодиодной линейке.....	197
Цветовые модели.....	197
Радужное свечение светодиодов .....	199
Использование светодиодов для получения отладочных данных в сценариях обхода препятствий .....	201
Реализация обычного свечения .....	202
Реализация радужного свечения.....	204
Выводы .....	205
Задание.....	205
Дополнительные материалы .....	206
<b>Глава 10. Код на Python для управления сервоприводами ....</b>	<b>207</b>
Технические требования .....	207
Что такое сервоприводы?.....	208
Сервопривод: взгляд изнутри .....	209
Отправка управляющих сигналов сервоприводу.....	210
Позиционирование сервоприводов с помощью Raspberry Pi .....	212
Создание тестового кода .....	213
Устранение неполадок .....	216
Управление ДПТ и сервоприводами .....	217
Калибровка сервоприводов .....	218
Создание механизма поворота и наклона .....	219



Сборка механизма .....	220
Установка механизма поворота и наклона на робота.....	224
Разработка кода для механизма поворота и наклона.....	225
Создание объекта Servos .....	225
Добавляем сервоприводы в класс Robot .....	228
Разработка скрипта для движения «головы» робота по окружности .....	229
Запуск сценария .....	231
Устранение неполадок .....	231
Реализация сканирующего сонара .....	232
Установка датчика .....	232
Установка библиотек.....	235
Поведенческий скрипт.....	235
Выводы .....	238
Задание.....	239
Дополнительные материалы .....	239
<b>Глава 11. Код на Python для работы с энкодерами .....</b>	<b>240</b>
Технические требования.....	240
Измерение пройденного расстояния с помощью энкодеров.....	241
Где применяются энкодеры.....	241
Типы энкодеров .....	241
Кодирование абсолютного или относительного положения.....	243
Кодирование направления и частоты вращения .....	244
Энкодеры, которые мы будем использовать .....	245
Установка энкодеров на робота.....	246
Подготовка энкодеров.....	247
Снятие Raspberry Pi .....	248
Установка энкодеров на шасси .....	249
Подключение энкодеров к Raspberry Pi .....	249
Измерение пройденного расстояния с помощью кода на Python .....	251
Ведение журнала .....	251
Простой подсчет .....	252
Добавляем энкодеры в объект Robot.....	255
Преобразование тактов в миллиметры .....	257
Движение по прямой.....	259
Коррекция отклонения с помощью ПИД-регуляторов .....	259
Создание объекта ПИД-регулятора.....	261
Создание кода для движения по прямой.....	263
Устранение неполадок .....	265
Перемещение на определенное расстояние.....	266
Рефакторинг преобразования единиц измерения для класса EncoderCounter.....	266
Настройка констант.....	267
Разработка поведенческого скрипта для движения на определенное расстояние .....	268
Выполнение точного поворота.....	270
Создание функции drive_arc .....	274

Выводы .....	275
Задание .....	276
Дополнительные материалы .....	276

## **Глава 12. Код на Python для работы с IMU..... 278**

Технические требования .....	279
Подробнее об инерциальных измерительных модулях (IMU) .....	279
Предлагаемые модели IMU .....	280
Пайка – подсоединение контактов к IMU .....	281
Создание паянного соединения .....	281
Установка IMU на робота .....	283
Физическое размещение .....	283
Подключение IMU к Raspberry Pi .....	286
Считывание данных датчика температуры .....	288
Установка программного обеспечения .....	288
Устранение неполадок .....	289
Считывание регистра температуры .....	289
Устранение неполадок .....	293
Упрощение командной строки VPython .....	294
Считывание данных гироскопа с помощью Python .....	294
Как работает гироскоп .....	294
Добавление гироскопа в интерфейс .....	297
Построение графика данных гироскопа .....	297
Считывание данных акселерометра с помощью Python .....	299
Как работает акселерометр .....	299
Добавление акселерометра в интерфейс .....	300
Отображение данных акселерометра в виде вектора .....	300
Считывание данных магнитометра .....	302
Как работает магнитометр .....	302
Добавление интерфейса магнитометра .....	304
Отображение данных магнитометра .....	304
Выводы .....	306
Задание .....	306
Дополнительные материалы .....	306

## **ЧАСТЬ 3. СЛУХ И ЗРЕНИЕ – «ИНТЕЛЛЕКТУАЛЬНЫЕ» ДАТЧИКИ ДЛЯ РОБОТА..... 309**

### **Глава 13. Система технического зрения робота – камера Pi и OpenCV ..... 311**

Технические требования .....	311
Настройка камеры Raspberry Pi .....	312
Установка камеры на механизм поворота и наклона .....	313
Подключение камеры .....	316
Настройка программного обеспечения для задач компьютерного зрения .....	318

Настройка программного обеспечения камеры Pi .....	318
Получение изображения с Raspberry Pi .....	319
Установка OpenCV и вспомогательных библиотек .....	319
Создание приложения для потоковой передачи данных камеры Raspberry Pi.....	320
Разработка потокового сервера OpenCV.....	321
Создание объекта CameraStream .....	322
Создание главного приложения сервера изображений.....	323
Создание шаблона .....	325
Запуск сервера изображений.....	326
Устранение неполадок .....	326
Запуск фоновых задач во время потоковой передачи.....	327
Создание ядра веб-приложения .....	328
Разработка управляемого поведенческого скрипта .....	331
Создание шаблона элемента управления .....	332
Запуск управляемого сервера изображений .....	333
Отслеживание цветных объектов с помощью кода на Python .....	333
Преобразование изображения в информацию .....	334
Усовершенствование ПИД-регулятора .....	336
Создание компонентов поведенческого скрипта .....	338
Запуск поведенческого скрипта .....	343
Настройка параметров ПИД-регулятора .....	344
Устранение неполадок .....	345
Отслеживание лиц с помощью кода на Python .....	346
Поиск объекта на изображении.....	346
Сканирование базовых признаков.....	347
План поведенческого сценария отслеживания лиц.....	349
Создание кода для сценария отслеживания лиц.....	350
Запуск поведенческого скрипта отслеживания лиц .....	353
Устранение неполадок .....	354
Выводы .....	354
Задание.....	355
Дополнительные материалы .....	355

## Глава 14. Код на Python для отслеживания

### линий с помощью камеры..... 357

Технические требования.....	358
Введение в отслеживание линий.....	358
Что такое отслеживание линий?.....	358
Использование в промышленности.....	358
Типы алгоритмов отслеживания линий .....	359
Создание тестового маршрута.....	360
Материалы для создания тестового маршрута .....	361
Создание линии .....	361
Конвейерная обработка данных компьютерного зрения для следования по линиям .....	362

Алгоритмы отслеживания линий с помощью камеры .....	362
Конвейер обработки изображения .....	363
Проверка алгоритма компьютерного зрения на основе образцов изображений .....	366
Зачем нужны тестовые изображения? .....	366
Получение тестовых изображений .....	366
Создание кода на Python для обнаружения границ линии .....	368
Определение положения линии по границам .....	371
Тест снимков без четкой линии .....	372
Следование по линиям с помощью ПИД-алгоритма .....	374
Создание блок-схемы поведенческого сценария .....	375
Добавляем время в ПИД-регулятор .....	376
Разработка начального кода для поведенческого скрипта .....	376
Настройка параметров ПИД-регулятора .....	383
Устранение неполадок .....	383
Нахождение линии при ее потере .....	383
Выводы .....	384
Задание .....	385
Дополнительные материалы .....	385

## **Глава 15. Голосовая связь с роботом посредством Microsoft..... 386**

Технические требования .....	387
Введение в Microsoft – основные понятия .....	387
Преобразование речи в текст .....	387
Слово для пробуждения .....	387
Реплика .....	388
Намерение .....	388
Диалог .....	388
Словарь .....	388
Навык .....	388
Ограничения голосового управления .....	389
Аудиовход и выход для Raspberry Pi .....	389
Физическая установка .....	390
Установка программного обеспечения голосового помощника на Raspberry Pi .....	391
Установка программного обеспечения ReSpeaker .....	392
Настройка взаимодействия звуковой карты и Microsoft .....	394
Начало использования Microsoft .....	395
Устранение неполадок .....	397
Разработка API с помощью Flask .....	397
Как работает управления роботом с помощью Microsoft .....	398
Удаленный запуск поведенческого скрипта .....	399
Создание сервера API управления с помощью Flask .....	401
Устранение неполадок .....	403
Разработка кода для запуска Microsoft на Raspberry Pi .....	403
Разработка намерения .....	404

Устранение неполадок .....	409
Создание еще одного намерения .....	410
Выводы .....	411
Задание.....	412
Дополнительные материалы .....	413
<b>Глава 16. Погружаемся глубже в работу IMU .....</b>	<b>414</b>
Технические требования.....	414
Разработка кода для виртуального робота .....	415
Создание модели робота в VPython .....	415
Определение параметров вращения с помощью гироскопа.....	419
Калибровка гироскопа .....	421
Изменение положения виртуального робота на основе данных гироскопа .....	422
Измерение углов тангажа и крена с помощью акселерометра .....	426
Определение углов тангажа и крена на основе векторных данных акселерометра .....	426
Сглаживание данных акселерометра.....	429
Совместная обработка данных акселерометра и гироскопа .....	430
Определение направления с помощью магнитометра .....	433
Калибровка магнитометра .....	433
Приблизительное определение направления робота с помощью магнитометра на практике .....	439
Объединение показаний датчиков для ориентирования робота .....	441
Управление роботом на основе показаний IMU.....	447
Выводы .....	449
Задание.....	449
Дополнительные материалы .....	449
<b>Глава 17. Разработка кода на Python для управления роботом с помощью смартфона.....</b>	<b>451</b>
Технические требования.....	452
Когда голосовое управление не работает (или Зачем нам нужно управлять роботом) .....	452
Режимы меню – выбор поведенческих сценариев .....	452
Управление режимами робота .....	454
Устранение неполадок .....	455
Веб-сервер.....	456
Шаблон .....	456
Запуск приложения .....	458
Устранение неполадок .....	460
Выбор контроллера – как лучше управлять роботом и почему .....	461
Обзор будущего приложения.....	461
Подготовка Raspberry Pi к удаленному управлению – основы системы управления.....	464
Расширение кода ядра изображения .....	465
Разработка сценария ручного управления.....	466



Шаблон (веб-страница) .....	468
Таблица стилей .....	471
Разработка кода для ползунков .....	474
Запуск системы .....	477
Устранение неполадок .....	478
Реализация полного управления роботом через смартфон .....	479
Совместимость режимов меню с поведенческими сценариями, использующими Flask .....	479
Загрузка видеосервисов .....	480
Стилизация меню .....	482
Реализация запуска меню вместе с Pi .....	484
Добавляем светодиоды на сервер меню .....	484
Автоматический запуск с помощью инструмента systemd .....	485
Выводы .....	487
Задание .....	488
Дополнительные материалы .....	488
<b>Часть 4. Дальнейшее изучение робототехники .....</b>	<b>491</b>
<b>Глава 18. Дальнейшее развитие навыков программирования робототехнических систем .....</b>	<b>493</b>
Робототехнические интернет-сообщества – форумы и социальные сети .....	493
Каналы на YouTube, с которыми стоит ознакомиться .....	495
Технические вопросы – куда обратиться за помощью .....	496
Мероприятия – соревнования, площадки для совместной работы и встречи .....	496
Площадки для совместной работы .....	497
Maker Faire, Raspberry Jam и Coder Dojo .....	497
Соревнования .....	498
Навыки, которые стоит освоить, – 3D-печать, пайка, печатные платы и станки с ЧПУ .....	499
Навыки проектирования .....	499
Навыки обработки компонентов и сборки .....	500
Навыки работы с электроникой .....	502
Подробнее о компьютерном зрении .....	504
Книги .....	504
Онлайн-курсы .....	504
Социальные сети .....	505
Переход к машинному обучению .....	505
Операционная система для роботов .....	506
Выводы .....	507
Дополнительные материалы .....	507
<b>Глава 19. Планирование следующего робототехнического проекта – подводим итоги .....</b>	<b>508</b>
Технические требования .....	508
Представляем нового робота – как он будет выглядеть .....	508

Создание блок схемы.....	510
Выбор компонентов .....	511
Планирование кода .....	512
Рассказываем миру о проекте .....	515
Выводы .....	516
<b>Предметный указатель .....</b>	<b>517</b>

# Участники

## ОБ АВТОРЕ

**Дэнни Стейпл (Danny Staple)** занимается созданием роботов и гаджетов в качестве хобби, снимает видеоролики, посвященные робототехнике, а также является участником таких мероприятий, как Pi Wars и Arduino Day. С 2000 года профессионально занимается разработкой программного обеспечения. В 2009 году начал заниматься программированием на Python, уделяя особое внимание процессам разработки и автоматизации. Большую часть своей профессиональной деятельности Дэнни посвятил работе со встроенными системами, включая встроенные системы Linux. Сейчас является наставником в CoderDojo Ham, где обучает детей программированию. Ранее руководил клубами LEGO Robotics.

Вместе с детьми Дэнни создал таких роботов, как TankBot, SkittleBot, Bangers N Bash (робот из ланчбокса), Big Ole Yellow (еще один гусеничный робот), ArmBot и SpiderBot.

*Я хотел бы поблагодарить Дэвида Андерсона (David Anderson) за ценные советы относительно моих идей и мотивацию. Также я выражаю благодарность Бену Нутталлу (Ben Nuttall) и Дейву Джонсу (Dave Jones) (@waveform80) за разработку GPIOZero и ответы на мои бесчисленные вопросы в Twitter. Со встречи с Дейвом Джонсом (Dave Jones), автором библиотеки PiCamera, в ресторане Кардиффа начался мой путь к изучению компьютерного зрения. И наконец, я благодарю своих детей, Хелену (Helena) и Джонатана (Jonathan), за их поддержку, терпение и помощь при работе с графиками.*

## О РЕЦЕНЗЕНТАХ

**Лео Уайт (Leo White)** – выпускник Кентского университета (University of Kent), профессиональный инженер-программист, интересующийся электроникой, 3D-печатью и робототехникой. Сначала он занимался программированием на Commodore 64, затем разработал несколько приложений для Acorn Archimedes, а сейчас на постоянной основе занимается разработкой кода для ТВ-приставок. На основе Raspberry Pi Лео создавал роботов из детских игрушек и роботов-манипуляторов, попутно описывая этот процесс и свой опыт в блоге. Также проводил презентации на Raspberry Jams и участвовал в соревнованиях Pi Wars.

**Рамкумар Гандинатан (Ramkumar Gandhinathan)** – исследователь и профессиональный робототехник. Своего первого робота создал в шестом классе. Благодаря личным и профессиональным связям он занимается робототехникой уже более 15 лет. Рамкумар создал более 80 различных роботов. Общий

профессиональный опыт в области робототехники составляет 7 лет (4 года полной занятости и 3 года неполной/стажировки). На протяжении 5 лет он работал с ROS (Robot Operating System – операционная система для роботов). В рамках профессиональной карьеры Рамкумар разработал свыше 15 решений ROS для промышленных роботов. Также он увлекается созданием и пилотированием дронов. В круг его исследовательских интересов и увлечений входит SLAM (Simultaneous Localization And Mapping – одновременная локализация и построение карты), планирование движений, совместное использование данных датчиков, коммуникация между роботами и системная интеграция.

# Предисловие

Эта книга посвящена созданию интеллектуального робота и разработке кода для его поведенческих сценариев. Вы узнаете о навыках, которые требуются для создания робота из отдельных компонентов, а также о тонкостях их выбора. Для построения робота из этой книги вы будете использовать такие компоненты, как датчики, двигатели, камеры, микрофоны, динамики, светодиоды и Raspberry Pi.

Далее в книге вы узнаете, как разработать код для перечисленных компонентов. Вы будете использовать язык программирования Python, а также немного HTML/CSS и JavaScript.

Все компоненты робота, описанные в этой книге, широко доступны. Приведенные примеры кода призваны продемонстрировать, как работает та или иная технология. В дальнейшем вы сможете комбинировать разные части программного и аппаратного обеспечения, чтобы создавать сложных роботов с более интересными поведенческими сценариями.

В книге пересекаются темы программирования и робототехники, а также раскрывается ряд специализированных тем, таких как компьютерное зрение и голосовое управление.

## Для кого эта книга

Книга подойдет как новичкам в области программирования, так и опытным программистам, желающим применить свои навыки в аппаратном проекте. Для этого не обязательно быть экспертом, достаточно быть способным ввести несколько строк кода и уметь работать с циклами, условиями и функциями. В книге затрагивается тема объектно-ориентированного программирования, но вам не обязательно разбираться в нем перед прочтением.

Для создания робота из этой книги вам не понадобится специально оборудованная мастерская – достаточно уметь паять и крепить компоненты болтами. Позже мы поговорим об этом подробнее.

Вам вовсе не обязательно иметь опыт работы с электроникой или созданием каких-либо устройств. В книге представлены базовые понятия, и я надеюсь, что это вызовет у вас здоровый интерес к дальнейшему изучению. Самое главное – это ваше желание создать робота и научить его делать интересные вещи.

## Какие темы описаны в книге

В главе 1 рассказывается о том, как устроены роботы, как они применяются на производстве и в быту, а также описываются некоторые примеры роботов, созданных новичками.

В главе 2 подробно рассказывается о компонентах робота и о том, как их выбирать. Также здесь представлены блок-схемы будущей системы и кода.

В главе 3 говорится о микрокомпьютере Raspberry Pi, его подключении и операционной системе – Raspbian Linux. В этой главе вы запишете образ операционной системы на SD-карту и научитесь использовать ее в работе.

В главе 4 рассказывается, как реализовать беспроводную связь робота с Raspberry Pi.

В главе 5 вы узнаете, какие неполадки могут возникнуть в коде и как предотвратить его потерю, сохранив копии в репозиториях.

В главе 6 мы переходим к сборке основания робота. Здесь рассказывается о тонкостях выбора компонентов и проверке их размещения.

В главе 7 показано, как разработать код для движения робота. Также здесь закладываются основы для кода из следующих глав.

В главе 8 мы добавим роботу датчики и разработаем для них код, благодаря которому робот сможет самостоятельно обходить стены и препятствия.

В главе 9 рассказывается, как добавить роботу цветные светодиоды. Здесь вы узнаете, как использовать дополнительные выводы для отладки или просто для развлечения.

В главе 10 рассказывается о сервоприводах и их использовании для позиционирования сенсорной головки, а также «конечностей» робота.

В главе 11 показано, как создать код для получения точной информации о перемещении колес с помощью одометра/тахометра. Здесь ваш робот научится ездить по прямой, выполнять точные повороты и определять, насколько далеко он переместился. В этой главе мы впервые поговорим о ПИД-регуляторах.

В главе 12 рассказывается об **инерциальном измерительном модуле (IMU)**, который включает датчики для измерения температуры, ускорения, скорости вращения и магнитных полей. В этой главе мы впервые поговорим о пайке и VPython.

В главе 13 вы узнаете, как получать данные с камеры, и научите робота передвигаться, основываясь на компьютерном зрении. Также здесь показано, как осуществляется потоковая передача обработанного видео в браузер.

В главе 14 рассказывается, как научить робота следовать по линиям с помощью камеры Raspberry Pi.

В главе 15 описывается разработка системы управления роботом с помощью голосового помощника. Вы научитесь использовать голосовые команды и получать от робота ответы.

В главе 16 рассказывается об объединении датчиков, описанных в главе 12. Вы узнаете, как использовать их для определения положения робота, а также разработаете поведенческий скрипт, который будет действовать как компас.

В главе 17 мы переходим к созданию веб-приложения меню и панели управления, похожей на геймпад. С помощью этой системы вы сможете управлять роботом со смартфона, опираясь на видео с камеры.

В главе 18 вы узнаете больше о мире робототехники. Здесь рассказывается о различных робототехнических сообществах, потенциальных областях развития и соревнованиях роботов.

В заключительной главе 19 обобщается представленная в книге информация, а также предлагаются варианты по созданию следующего робототехнического проекта.

## КАК ИЗВЛЕЧЬ ИЗ КНИГИ НАИБОЛЬШУЮ ПОЛЬЗУ

Прежде чем начать работать с книгой, я рекомендую вам потренироваться в работе с текстовыми языками программирования. Познакомьтесь с основными переменными, условными операторами, циклами и функциями.

Для работы вам потребуется компьютер с операционной системой macOS, Linux или Windows, доступ к интернету и Wi-Fi.

Что касается навыков ручного труда, я думаю, что вы умеете пользоваться отверткой и без проблем справитесь с кропотливой работой. Также я надеюсь, что вас не пугает пайка.

Представленные в книге примеры кода были протестированы на Python 3 с помощью Raspbian Buster и Picroft Buster Keaton. Их установка описывается в тексте соответствующих разделов. Также в тексте рассказывается, как брать нужные аппаратные компоненты.

Программное/аппаратное обеспечение, используемое в книге	Требования к операционной системе
Python 3	Raspbian Buster
Picroft/Mycroft	Picroft Buster Keaton
OpenCV	Raspbian Buster/Python 3
VPython	Raspbian Buster/Python 3
Flask	Python 3

*Прежде чем приобрести какие-либо аппаратные компоненты, ознакомьтесь с главами, в которых рассказывается, как правильно их выбрать.*

Если вы используете электронную версию книги, я рекомендую вводить код самостоятельно или брать его из репозитория на GitHub (ссылка предоставлена в следующем разделе). Это позволит избежать ошибок, связанных с неправильным копированием/вставкой фрагментов кода.

## ЗАГРУЗКА ФАЙЛОВ С ПРИМЕРАМИ КОДА

Все файлы с кодом из этой книги доступны через ваш аккаунт на веб-сайте [www.packt.com](http://www.packt.com). Если вы приобрели книгу в другом месте, зарегистрируйтесь, перейдите по адресу <https://www.packtpub.com/support> и оставьте запрос на отправку файлов по электронной почте.

Загрузить файлы с кодом вы можете следующим образом.

1. Войдите (или зарегистрируйтесь) в аккаунт на веб-сайте [www.packt.com](http://www.packt.com).
2. Перейдите во вкладку **Support** (Поддержка).
3. Нажмите **Code Downloads** (Загрузить код).
4. В строке поиска введите название книги и следуйте дальнейшим инструкциям на экране.

После загрузки файлов нужно обязательно распаковать или извлечь папки с помощью новейшей версии одной из следующих программ:



- WinRAR/7-Zip для Windows;
- Zipeg/iZip/UnRarX для Mac;
- 7-Zip/PeaZip для Linux.

Комплект исходного кода для этой книги размещен на GitHub по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition>. В случае обновления кода репозиторий также обновится.

По адресу <https://github.com/PacktPublishing/> доступны другие комплекты исходного кода из нашего богатого каталога книг и видео. Ознакомьтесь и с ними!

## CODE IN ACTION

Видеоролики Code in Action для этой книги доступны по адресу <http://bit.ly/3bu5wHp>.

## ИЗОБРАЖЕНИЯ В ЦВЕТЕ

По адресу [https://static.packt-cdn.com/downloads/9781839218804\\_ColorImages.pdf](https://static.packt-cdn.com/downloads/9781839218804_ColorImages.pdf) вы можете скачать файл, содержащий все изображения (в том числе скриншоты и схемы) из книги, в формате PDF.

## ИСПОЛЬЗУЕМЫЕ ОБОЗНАЧЕНИЯ

В книге используется ряд текстовых обозначений.

**Код в тексте:** так в тексте выделяются фрагменты кода, названия таблиц баз данных, папок, файлов, расширения файлов, составные имена файлов, пути, фиктивные URL-адреса, фрагменты пользовательского ввода и ссылки на Twitter. Например: «... задает параметр `color` для светодиода с номером `led_number`».

Блоки кода выглядят следующим образом:

```
cyan_rgb = [int(c * 255) for c in cyan]
```

Элементы или строки кода, на которые нужно обратить особое внимание, выделяются жирным шрифтом:

```
right_distance = self.robot.right_distance_sensor.distance
# Отображение
self.display_state(left_distance, right_distance)
```

Ввод или вывод командной строки выглядит так:

```
>>> r.leds.show()
```

**Жирный шрифт:** так выделяются новые термины, важные части текста или текст, который вы должны видеть на экране. Например, текст из меню или диалоговых окон: «Выберите **4** для **Other USB Microphone** (Другой USB-микрофон) и проверьте звук».

### Советы и важные примечания

Они выделяются следующим образом.

## ОСТАВАЙТЕСЬ НА СВЯЗИ

Мы всегда рады обратной связи.

**Общие вопросы:** любые вопросы насчет этой книги вы можете задать в электронном письме по адресу [customercare@packtpub.com](mailto:customercare@packtpub.com), указав в теме письма название книги.

**Ошибки и опечатки:** мы приложили все усилия, чтобы содержание было точным, однако ошибки все же случаются. Если вы нашли ошибку в этой книге, сообщите нам об этом. Для этого перейдите по адресу [www.packtpub.com/support/errata](http://www.packtpub.com/support/errata), выберите нужную книгу, щелкните по ссылке **Errata Submission Form** (Форма отправки сведений об ошибках) и введите ваше сообщение.

**Нарушение авторского права:** если вы столкнетесь с незаконными копиями наших книг, размещенными в интернете в любой форме, сообщите нам адрес или название веб-сайта. Для этого отправьте письмо на электронный адрес [copyright@packt.com](mailto:copyright@packt.com) со ссылкой на материал.

**Для авторов:** по вопросам сотрудничества авторы могут обратиться по адресу <https://authors.packtpub.com>.

## Отзывы

Почему бы не оставить отзыв о прочитанной книге на сайте, где вы ее приобрели? Благодаря вам потенциальные читатели перед приобретением смогут ознакомиться с вашим непредвзятым мнением, а издательство Packt и авторы увидят отзыв о своем продукте. Спасибо!

Больше информации на веб-сайте [www.packt.com](http://www.packt.com).



# Часть 1

## Введение – основы робототехники

В этой части вы на примерах рассмотрите, что представляет из себя робот, узнаете, как он устроен, и увидите, как подготовить Raspberry Pi для дальнейших экспериментов.

Часть включает в себя следующие главы:

- главу 1. Введение в робототехнику;
- главу 2. Структурные элементы робота – код и электроника;
- главу 3. Изучение Raspberry Pi;
- главу 4. Автономное управление роботом с помощью Raspberry Pi;
- главу 5. Создание резервных копий с помощью Git и SD-карты.



# Глава 1

## Введение в робототехнику

В этой книге мы расскажем, как построить робота и создать программы, позволяющие ему имитировать разум и способность принимать самостоятельные решения. Мы напишем код для сенсоров, благодаря которому робот сможет наблюдать за окружающей средой. Также посредством кода мы наделим робота такими способностями, как зрение, речь и распознавание речи.

Вы увидите, как путем комбинации простых методов сборки и небольшого количества кода можно создать робота, по многим признакам напоминающего настоящее домашнее животное. Кроме того, мы расскажем об отладке роботов подобного типа, что пригодится вам в случае, если что-то пойдет не так (к слову, это неизбежно). Вы узнаете, как обучить робота давать сигналы об ошибках, а также выбирать поведение, соответствующее вашим установкам. Мы подключим к роботу джойстик с функцией голосового управления и, наконец, покажем как организовать дальнейшую сборку робота.

Прежде чем приступать к созданию робота, необходимо разобраться в том, что он из себя представляет. Здесь мы рассмотрим некоторые типы роботов, а также выделим особенности, отличающие их от других машин. Вы попробуете провести грань между роботами и другими машинами. Однако, когда вы узнаете, как обстоит ситуация на самом деле, все может стать чуть более запутанным. Далее мы рассмотрим ряд уже существующих роботов, созданных робототехниками-любителями.

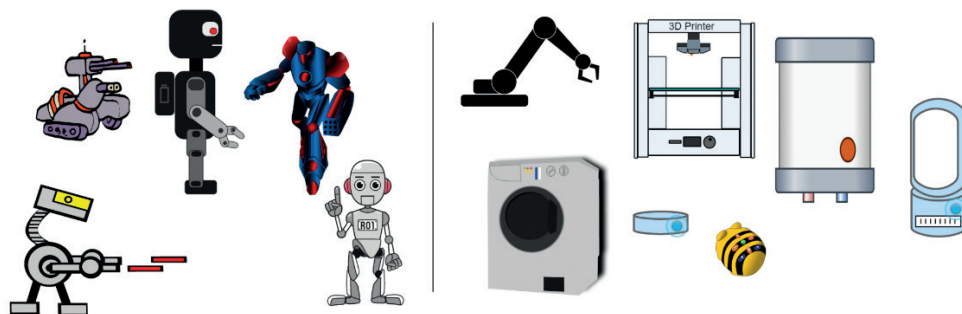
В этой главе будут разобраны следующие темы:

- что такое робот;
- продвинутые и впечатляющие роботы;
- роботы в доме;
- роботы в промышленности;
- роботы для участия в соревнованиях, учебные и любительские роботы.

### Что такое робот?

**Робот** – это машина, способная принимать автономные решения на основе данных сенсоров. Программный агент – это программа, обеспечивающая автоматическую обработку входных данных и выполнение выходных операций. Возможно, лучше всего можно описать робота как автономного программного агента с сенсорами, способного передвигаться в соответствии с выходными операциями. Часто роботов характеризуют как электромеханические платформы с программным обеспечением. Так или иначе, робот состоит из трех основных элементов: электроники, механических деталей и кода.

Слово «*робот*» вызывает в воображении образы творений из мира фантастики и научной фантастики, машин, обладающих невероятной силой и интеллектом. Зачастую эти творения очень похожи на людей, т. е. являются **андроидами**. Авторы наделяют их индивидуальностью, ввиду чего роботы ведут себя как несколько наивные люди:



**Рис. 1.1.** Роботы из мира научной фантастики и реальные роботы. Используемые изображения взяты из публичной библиотеки OpenClipArt

Название «*робот*» пришло из научно-фантастических произведений. Оно происходит от чешского слова, означающего «*раб*». Впервые название было использовано в пьесе Карела Чапека (Karel Capek) «*Россумские универсальные роботы*» (*Rossum's Universal Robots*), поставленной в 1921 году. Слово «*робототехника*» придумал автор научно-фантастических произведений Айзек Азимов (Isaac Asimov) во время исследования поведения «интеллектуальных» роботов.

Большинство роботов, используемых человеком, на самом деле не являются столь продвинутыми и притягивающими внимание. Большинство из них не стоит на двух ногах, да и вообще не имеет ног. Некоторые из них передвигаются с помощью колес, а некоторые не передвигаются и вовсе, но при этом имеют движущиеся части и сенсоры.

Такие роботы, как стиральные машины, роботы-пылесосы, полностью саморегулирующиеся бойлеры и вентиляторы с функцией забора проб воздуха, проникли в наши дома и стали частью повседневной жизни. Они не несут угрозы и являются просто машинами вокруг нас. Гораздо больший интерес вызывают 3D-принтеры, роботы-манипуляторы и обучающие игрушки.

По своей сути любой робот может быть представлен в виде трех составляющих: элемента, отвечающего за выходные операции (**выход**), например двигателя; элемента, осуществляющего сбор входных данных (**вход**), например сенсора; и **контроллера**, обеспечивающего обработку данных и исполнение кода. Итак, базовый робот выглядит следующим образом.

- Он оснащен входами и сенсорами, предназначенными для измерения свойств окружающей среды.
- Он имеет выходы, например двигатели, световые и звуковые индикаторы, клапаны и нагревательные элементы, способные влиять на окружающую среду.
- Он принимает решения относительно выходных операций, основываясь на полученных входных данных.



В следующем разделе мы рассмотрим более продвинутых роботов.

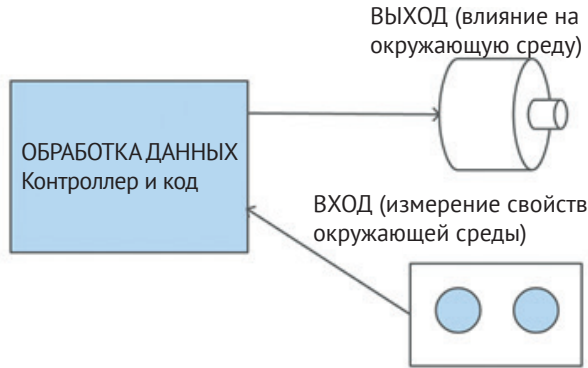


Рис. 1.2. Упрощенное представление робота

## ПРОДВИНУТЫЕ И ВПЕЧАТЛЯЮЩИЕ РОБОТЫ

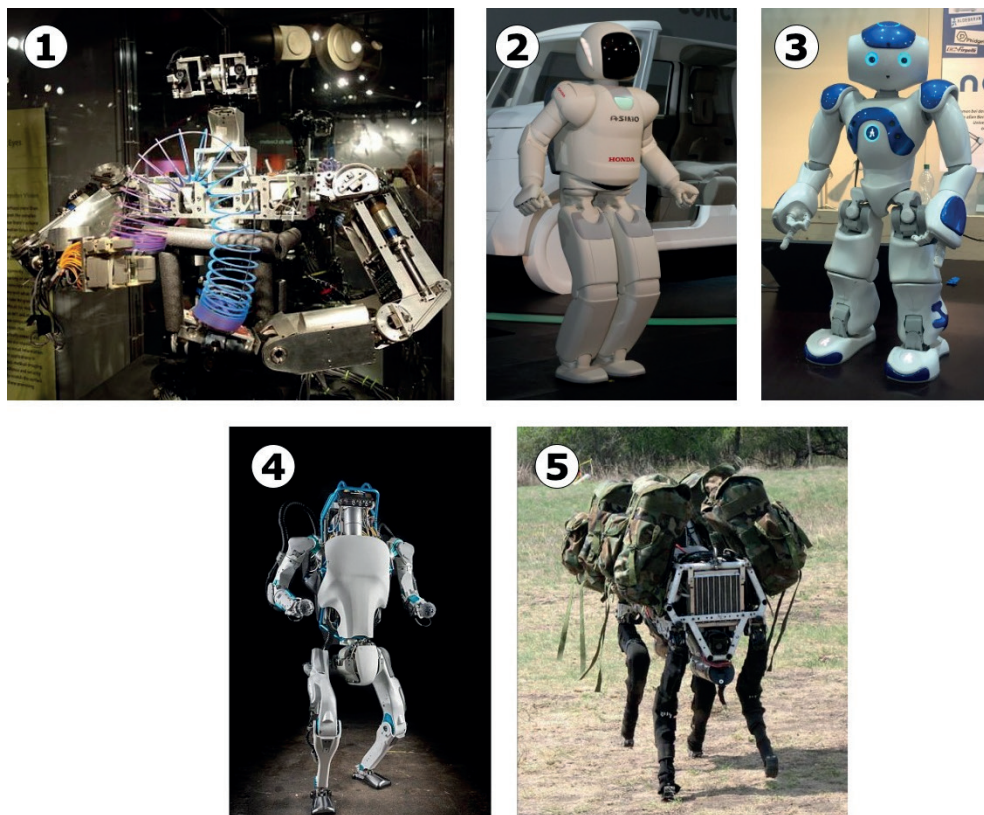
Теперь, когда вы имеете общее представление о роботах, я представлю несколько наиболее впечатляющих примеров роботов и расскажу об их способностях. Помимо марсоходов, зачастую создатели роботов отдают предпочтение роботам, похожим на людей и животных. Такой выбор обусловлен большей адаптивностью подобных форм, в отличие от роботов, разработанных для промышленного использования и предназначенных для многократного выполнения однообразных задач (*single repeated use*).

На рис. 1.3 представлены роботы, похожие на людей или животных.

Общим для представленных роботов является то, что они имитируют людей и животных.

1. Робот 1 – это Cog, созданный командой Массачусетского технологического института (Massachusetts Institute of Technology, MIT). Cog должен был походить на человека своими движениями и «органами восприятия» (сенсорами).
2. Робот 2 – это робот-андроид ASIMO, разработанный корпорацией Honda. Он умеет двигаться и разговаривать почти как человек. ASIMO оснащен двумя камерами, служащими для *обхода препятствий*, распознавания жестов и лиц. Также робот способен определять поверхность посредством лазерных датчиков расстояния. Благодаря инфракрасным сенсорам он может обнаруживать отметки на полу и следовать по ним. ASIMO понимает голосовые команды на английском и японском языках.
3. Робот 3 – Nao, созданный Softbank Robotics. Этот симпатичный робот высотой всего 58 см был разработан как обучающее и игровое программируемое устройство. Он оснащен сенсорами, посредством которых может следить за собственными движениями (например, распознавать падение) и ультразвуковыми датчиками, позволяющими *предотвращать столкновения* с другими объектами. Nao может распознавать и обрабатывать голосовые команды с помощью динамиков и микрофона. Он, как и ASIMO, оснащен несколькими камерами для схожих целей.

4. Робот 4 – Atlas от Boston Dynamics. Этот робот отличается скоростью перемещения на двух ногах и плавностью движений. Он оснащен матрицей лазерных сканеров – **лидаров** (LIDAR), благодаря которым может обнаруживать окружающие его объекты, что позволяет планировать дальнейшее движение и предотвращать столкновения.
5. Робот 5 – это четвероногий робот BigDog, также созданный Boston Dynamics. От других подобных роботов он отличается устойчивостью, способностью оставаться в вертикальном положении при толчках и не падать при движении по льду.



**Рис. 1.3.** Подборка роботов, похожих на людей и животных. [Источники графических материалов: Изображение 1: [https://commons.wikimedia.org/wiki/File:Cog,\\_1993-2004,\\_view\\_2\\_-\\_MIT\\_Museum\\_-\\_DSC03737.JPG](https://commons.wikimedia.org/wiki/File:Cog,_1993-2004,_view_2_-_MIT_Museum_-_DSC03737.JPG) (открытый доступ); Изображение 2: [https://commons.wikimedia.org/wiki/File:Honda\\_ASIMO\\_\(ver.\\_2011\)\\_2011\\_Tokyo\\_Motor\\_Show.jpg](https://commons.wikimedia.org/wiki/File:Honda_ASIMO_(ver._2011)_2011_Tokyo_Motor_Show.jpg), автор Morio, CC BY-SA 3.0: <https://creativecommons.org/licenses/by-sa/3.0/deed.en>; Изображение 3: это [https://commons.wikimedia.org/wiki/File:Nao\\_Robot\\_\(Robocup\\_2016\).jpg](https://commons.wikimedia.org/wiki/File:Nao_Robot_(Robocup_2016).jpg) (открытый доступ); Изображение 4: [wikimedia.org/wiki/File:Atlas\\_from\\_boston\\_dynamics.jpg](https://commons.wikimedia.org/wiki/File:Atlas_from_boston_dynamics.jpg), автор [https://www.kansascity.com/news/business/technology/917xpi/picture62197987/ALTERNATES/FREE\\_640/atlas%20from%20boston%20dynamics](https://www.kansascity.com/news/business/technology/917xpi/picture62197987/ALTERNATES/FREE_640/atlas%20from%20boston%20dynamics), CC BY-SA 4.0: <https://creativecommons.org/licenses/by-sa/4.0/deed.en>; Изображение 5: [https://commons.wikimedia.org/wiki/Commons:Licensing#Material\\_in\\_the\\_public\\_domain](https://commons.wikimedia.org/wiki/Commons:Licensing#Material_in_the_public_domain) (открытый доступ)]

Вы сможете реализовать подобные функции в своем роботе. Мы обсудим применение датчиков расстояния с целью обхода препятствий, рассмотрим ультразвуковые датчики расстояния, как у Nao, а также лазерные датчики расстояния, как у ASIMO. Далее мы исследуем применение камер для обработки видеоданных, линейных сенсоров для отслеживания отметок на полу, а также обработку голоса для реализации управления голосовыми командами. Также мы разработаем механизм поворота и наклона камеры, как у Cog.

## Марсоходы

Роботы-марсоходы предназначены для работы на другой планете, где в случае поломки не сможет вмешаться человек. Они обладают прочной конструкцией. Обновление программного обеспечения марсоходов осуществляется удаленно, поскольку нецелесообразно отправлять с этой целью на Марс человека с экраном и клавиатурой. По своей конструкции марсоходы предназначены для полностью автономной деятельности :



**Рис. 1.4.** Марсоход НАСА Curiosity в Глен-Этив, Марс (Источник: NASA/JPL-Caltech/MSSS; <https://mars.nasa.gov/resources/24670/curiosity-at-glen-etive/?site=msl>)

Марсоходы передвигаются с помощью колес, а не ног, поскольку стабилизировать колесного робота намного проще. Также в связи с этим снижаются риски. Каждое колесо марсохода имеет собственный двигатель. Колеса расположены таким образом, чтобы обеспечить максимальное сцепление и устойчивость при движении по каменистой поверхности Марса в условиях пониженной гравитации.

Марсоход Curiosity оснащен высокочувствительной камерой, которая в момент отправки была сложена. После приземления на Марс камера была развернута посредством сервоприводов. Камера наводится с помощью **механиз-**

**ма поворота и наклона.** Главная задача здесь – охватить как можно больше фрагментов ландшафта Марса и отправить отснятые материалы и фотографии в НАСА (National Aeronautics and Space Administration – NASA) для анализа.

Подобно марсоходам, робот, которого вы создадите в результате прочтения этой книги, будет передвигаться с помощью колес с приводом от двигателя. Наш робот предназначен для работы без устройств ввода, поскольку его конструкция не предусматривает прямое участие человека в его работе. По мере расширения возможностей нашего робота мы также будем использовать сервоприводы для управления механизмом поворота и наклона.

## Роботы в доме

Многие роботы уже проникли в наши дома. Мы не воспринимаем их как таковых, поскольку на первый взгляд они кажутся довольно обыденными и приземленными. Однако на самом деле они сложнее, чем кажется.

### Стиральная машина

Для начала поговорим о стиральной машине, которая сегодня есть почти в каждом доме. Каждый день мы используем этот бытовой прибор для стирки, отжима и сушки одежды. Но можно ли считать его роботом?

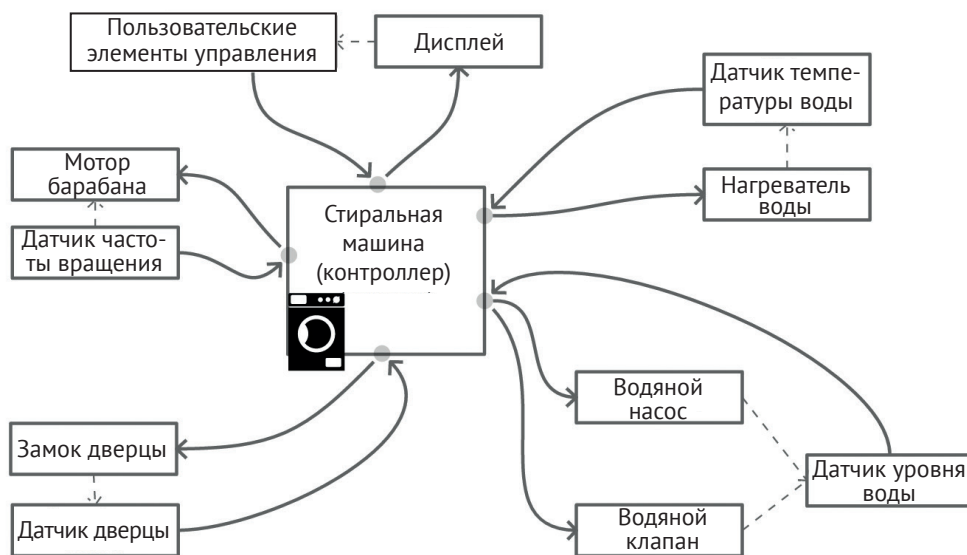


Рис. 1.5. Устройство стиральной машины

На рис. 1.5 схематично представлено устройство стиральной машины. Здесь мы видим центральный контроллер, который подключается к дисплею с элементами управления для выбора программы. Линии, выходящие из контроллера, являются выходами, а входящие представляют собой данные, поступающие с сенсоров (входы). Пунктирными линиями, идущими от выходов к входам, обозначен замкнутый цикл выходных операций, происходящих в ре-

альном мире и воздействующих на сенсоры. Данный цикл называется *петлей обратной связи*, или просто *обратной связью*, и является одной из важнейших концепций в робототехнике.

Пользователь выбирает настройки стиральной машины путем нажатия кнопок, а статус процесса отображается на дисплее. После нажатия кнопки запуска контроллер проверяет датчик дверцы, и в случае, если дверца не закрыта, процесс стирки не запустится. При запуске стирки дверца блокируется. Затем посредством нагревателей, клапанов и насосов, барабан стиральной машины заполняется водой. Путем обратной связи по сигналам датчика регулируется уровень и температура воды.

```

запустить водяной насос
включить нагреватель воды
while барабан not заполнен and вода not достаточно горячая:
    if барабан заполнен
        остановить водяной насос
    if вода достаточно горячая
        выключить нагреватель воды
    else
        включить нагреватель воды

```

Каждый процесс можно представить в виде набора строк, представленного выше. Подобный набор строк отвечает за одновременное наполнение барабана и поддержание температуры воды.

Обратите внимание, что строка после `else` необходима для тех случаев, когда температура воды падает ниже установленной. Далее стиральная машина начинает вращать барабан – сначала медленно, а затем быстро, при этом определяя оптимальную скорость для выбранной программы. После стирки вода из барабана сливается и стиральная машина отжимает одежду. Затем снимается блокировка дверцы, и процесс завершается.

Стиральную машину можно назвать роботом во всех отношениях. Это устройство имеет сенсоры, на основе данных которых оно способно выполнять выходные операции. Посредством обработки сигналов, полученных от сенсоров с обратной связью, контроллер следует установленной пользователем программе. Так, любой мастер по ремонту стиральных машин может быть даже ближе к робототехнике, чем я.

## Другие домашние роботы

Газовый бойлер, используемый вместо центрального отопления (gas central heating boiler), оснащен сенсорами, насосами и клапанами. Посредством механизмов обратной связи такой бойлер может поддерживать температуру в доме, регулировать расход воды на отопление, а также внимательно следить за тем, чтобы горелка не погасла. Бойлеры во многом похожи на роботов. Однако они являются стационарными, ввиду чего не могут быть адаптированы для иных целей. То же самое можно сказать и о других бытовых приборах, таких как умные вентиляторы и принтеры.

Умные вентиляторы определяют температуру, влажность и качество воздуха в помещении посредством сенсоров. На основании полученных данных они регулируют скорость вращения и нагревательные элементы.



Многие бытовые приборы, например такие как микроволновая печь, умеют работать только по таймеру. Такие устройства крайне просты и не способны принимать самостоятельные решения. Следовательно, их нельзя отнести к роботам.

На рис. 1.6 представлен, пожалуй, наиболее знакомый нам домашний робот – робот-пылесос.



**Рис. 1.6.** Робот-пылесос PicaBot (Источник: Handitec [Открытый доступ – [https:// commons.wikimedia.org/wiki/File:PicaBot.jpg](https://commons.wikimedia.org/wiki/File:PicaBot.jpg)])

Этот колесный мобильный робот похож на тот, которого мы предлагаем построить в этой книге, но чуть более симпатичный. Робот-пылесос оснащен сенсорами для обнаружения стен, ступеней и ограниченных зон. Также сенсоры помогают ему избегать столкновений. Как и наш будущий робот, робот-пылесос обладает такими характеристиками, как автономность, мобильность и программируемость.

По мере создания нашего робота мы научимся использовать сенсоры для обнаружения объектов. Также наш робот сможет реагировать на препятствия посредством формирования петель обратной связи, подобных тем, которые мы рассматривали в подразделе о стиральных машинах.

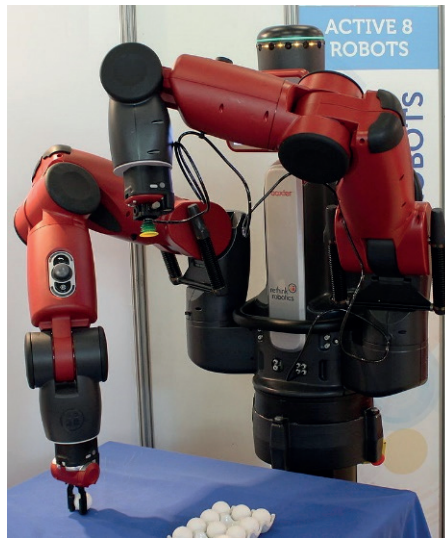
## РОБОТЫ В ПРОМЫШЛЕННОСТИ

Еще одно место, где часто встречаются роботы, – это промышленность. Первые роботы появились на заводах уже очень давно.

### Роботы-манипуляторы

**Роботы-манипуляторы** применяются для самых разных целей. Это могут быть небольшие роботы для переворачивания яиц или исполинские устрой-

ства, способные перемещать транспортировочные контейнеры. Как правило, роботы-манипуляторы оснащены шаговыми двигателями и сервоприводами. В этой книге мы рассмотрим серводвигатели в механизмах поворота и наклона. Большинство промышленных роботов-манипуляторов (например, сварочные роботы от компании ABB) следуют заранее заданной схеме движений и не принимают самостоятельные решения. Однако среди промышленных роботов встречаются и другие, представляющие собой интеллектуальные системы, основанные на сенсорах. Например, робот-манипулятор **Baxter** от Rethink Robotics, представленный на рис. 1.7.



**Рис. 1.7.** Робот-манипулятор Baxter от Rethink Robotics (Источник: Baxter на выставке Innorobo, © Ксавье Капе (Xavier Caré)/Wikimedia Commons [CC-BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0/>)])

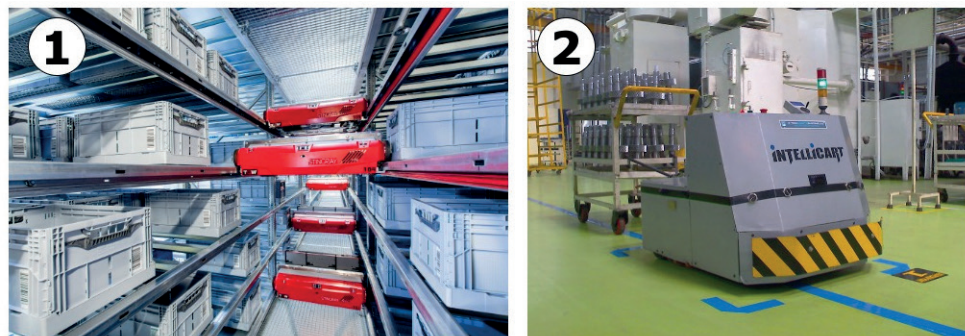
Работать рядом с многими роботами-манипуляторами небезопасно, поскольку это может привести к несчастным случаям. Поэтому подобные роботы зачастую размещаются за ограждениями, а рядом с ними наносится предупредительная маркировка. Однако Baxter несколько отличается от них: он умеет распознавать нахождение человека поблизости, а также может останавливаться во избежание несчастного случая. На рис. 1.7 вы можете рассмотреть сенсоры, отвечающие за упомянутые функции. Они расположены вокруг «головы» робота. Сенсоры, расположенные на «руках», а также нежесткие сочленения позволяют роботу обнаруживать столкновения и реагировать на них.

Baxter оснащен механизмом обучения и повторения, благодаря которому сотрудники производства могут адаптировать его к различным задачам. При тренировке или воспроизведении движений Baxter определяет и фиксирует нужные позиции. Наш робот будет использовать *кодовые датчики* (их часто называют просто *энкодерами*) для управления движением колес.



## Роботы на складах

Другой распространенный тип роботов в промышленности – это роботы, служащие для транспортировки предметов по цеху или складу.



**Рис. 1.8.** Роботизированные системы на складах: система Stingray от TGWmechanics [CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)] и система Intellicart от Mukeshhrs [открытый доступ]

Под номером 1 на рис. 1.8 представлены роботизированные системы подъемных кранов, предназначенные для транспортировки поддонов в складских комплексах. Они осуществляют транспортировку внутри стеллажных систем согласно получаемым инструкциям.

Под номером 2 на рис. 1.8 представлены небольшие роботы Intellicart, также предназначенные для перемещения предметов. Они полагаются на линейные сенсоры, ориентируясь на линии на полу, магнитно-чувствительные провода под полом или маркерные маяки (как ASIMO). Наш робот также сможет следовать по линиям. Как правило, все подобные роботы являются колесными, поскольку такой вариант наиболее прост в обслуживании, а также позволяет создавать стабилизированные платформы.

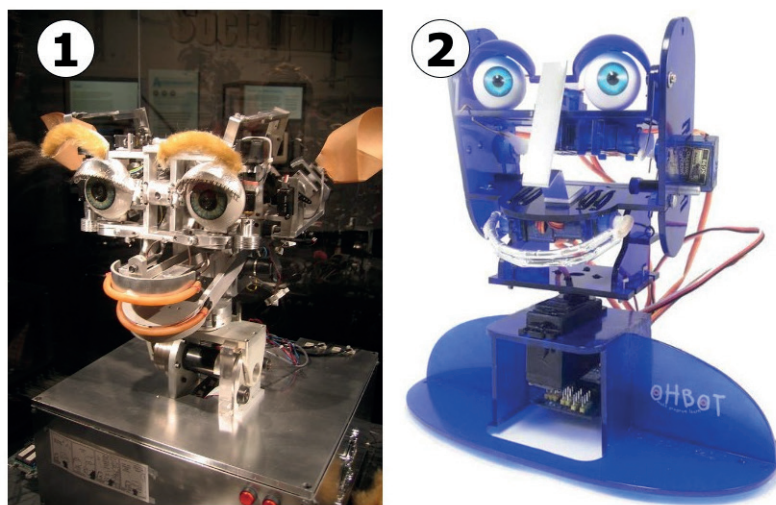
## РОБОТЫ ДЛЯ УЧАСТИЯ В СОРЕВНОВАНИЯХ, УЧЕБНЫЕ И ЛЮБИТЕЛЬСКИЕ РОБОТЫ

Самые занятные и инновационные роботы обычно создаются робототехниками-любителями.

Робототехника заняла прочную позицию в образовании. Многие робототехники, занятые в сфере образования, применяли свои разработки для обучения и экспериментов. Из таких учебных проектов впоследствии сформировалось множество коммерческих проектов. Университетские роботы, как правило, являются продуктом коллективного труда. Студентам и научным сотрудникам, занимающимся разработкой роботов, открывается доступ к высокотехнологичному оборудованию.

Kismet (под номером 1 на рис. 1.9) был создан командой Массачусетского технологического института (Massachusetts Institute of Technology – MIT) в конце 90-х годов. На его основе впоследствии было создано несколько любими-

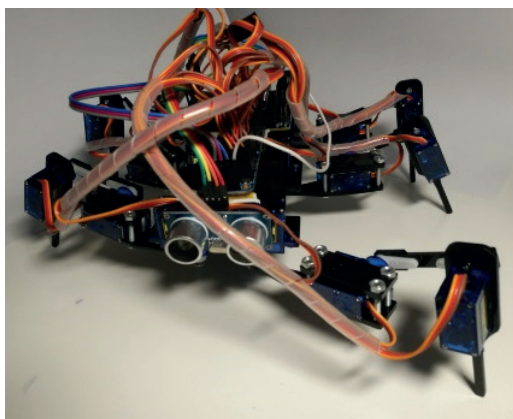
тельных роботов. Для своего времени Kismet был новаторской разработкой. Посредством приводов он мог управлять движениями и имитировать человеческую мимику. В частности, на основе Kismet создан OhBot (под номером 2 на рис. 1.9) – бюджетный любительский проект. Этот робот также может имитировать человеческую мимику и делает это под управлением Raspberry Pi на основе распознавания голоса и обработки данных камеры.



**Рис. 1.9.** Kismet [Джаред С. Бенедикт (Jared C Benedict) CC BY-SA 2.5 <https://creativecommons.org/licenses/by-sa/2.5>] и OhBot [AndroidFountain [CC BY-SA 4.0 (<https://creativecommons.org/licenses/by-sa/4.0>)]]

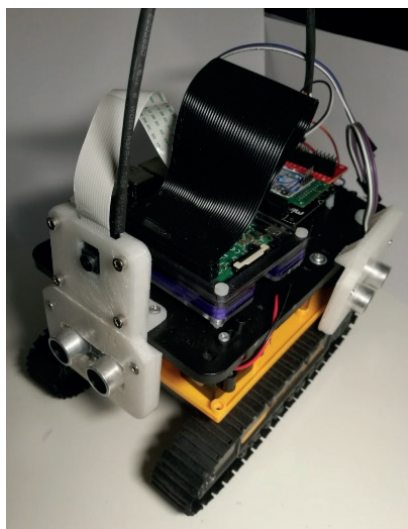
Любительская робототехника тесно связана с сообществом открытого программного и аппаратного обеспечения. На таких веб-сайтах, как GitHub (<https://github.com>), любители делятся своими проектами и кодами, что рождает новые идеи. Любительские роботы могут создаваться на основе готовых проектов в интернете, но с некоторыми модификациями и дополнениями. На сегодняшний день в интернете доступно множество наборов для сборки роботов, начиная от простых трехколесных платформ до дронов и гексаподов<sup>1</sup>. В некоторые наборы входят как механические, так и электронные компоненты. Подробнее мы рассмотрим их в главе 6, «Основы создания роботов – колеса, питание и электропроводка». Для исследования шагающих движений робота я создал SpiderBot (рис. 1.10) на основе набора для сборки гексапода.

<sup>1</sup> Гексапод – это похожий на паука шестиногий робот, который используется для самых разных целей, в первую очередь связанных с исследованиями в области робототехники. – *Прим. ред.*



**Рис. 1.10.** SpiderBot, созданный мной на основе готового проекта. В основе конструкции контроллер ESP8266 и 16-канальный сервоконтроллер Adafruit

Skittlebot был создан мной для участия в Pi Wars 2018. Основой для него послужил переделанный игрушечный экскаватор на дистанционном управлении. **Pi Wars** – это турнир, в котором принимают участие автономные роботы на базе Raspberry Pi, где оценивается их способность выполнять задачи как автономно, так и посредством ручного управления. В нем принимают участие самые разные роботы, и некоторые из них отличаются особым дизайном корпуса или изобретательными инженерными решениями. **Skittlebot** (рис. 1.11) оснащен тремя датчиками расстояния, благодаря которым он умеет обходить стены. Данный тип датчика мы рассмотрим более подробно в главе 8. Также Skittlebot умеет различать цветные объекты посредством камер. Подробнее об этом процессе мы поговорим в главе 13.



**Рис. 1.11.** Skittlebot – робот, в основе которого лежит игрушка. Создан мной для участия в Pi Wars 2018

Некоторые любительские роботы создаются с нуля посредством 3D-печати, лазерной резки, вакуумной формовки, обработки дерева, на станках ЧПУ и с помощью других технологий.

Робота, представленного на рис. 1.12, я создал с нуля для лондонской группы робототехников *The Aurorans* в 2009 году. Изначально он назывался EeeBot, поскольку предназначался для работы под управлением ноутбука Eee PC. Сообщество *The Aurorans* проводило собрания, где участники обсуждали робототехнику. Чуть позже EeeBot был оснащен Raspberry Pi, а также манипулятором (от uArm), благодаря чему он получил название **ArmBot**.

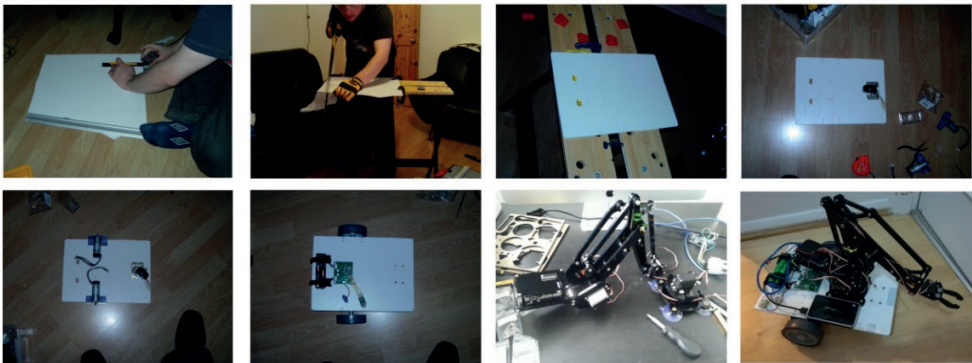


Рис. 1.12. Процесс создания ArmBot

Сегодня на рынке доступно множество готовых наборов для изготовления шасси, поэтому, чтобы создать функционирующего робота, новичку в робототехнике не придется вымерять и вырезать детали. Подобные проекты создаются для проведения экспериментов и призваны побуждать к творчеству других робототехников и детей. Ближе к концу книги мы поговорим о сообществах, которые создают роботов и популяризируют робототехнику. Также мы рассмотрим основные приемы конструирования для создания роботов с нуля.

«Битвы роботов» (Robot Wars) – это телевизионное шоу, посвященное соревнованиям между роботами, воплощающими впечатляющие конструкторские и инженерные решения. Однако все роботы-участники управляются вручную, как машинки на дистанционном управлении. Стиральные машины хотя и менее интересны, но гораздо умнее таких роботов.

## Выводы

В этой главе мы узнали, что означает слово «*робот*», а также рассмотрели факты и вымыслы, связанные с роботами. Мы определили, что такое настоящий робот и каким требованиям должна отвечать машина, чтобы считаться роботом.

Мы поговорили о роботах, которые встречаются дома и в промышленности. Также вы узнали, что некоторые из них поистине удивительны и предназначены для полетов на другие планеты. Мы рассмотрели любительских и образовательных роботов и узнали, что некоторые из них созданы просто для развлечения. Вы познакомились с устройством реальных приборов, которых

до этого, возможно, не считали роботами. Мы выяснили, что роботы уже давно проникли в наши дома.

Я надеюсь, что эта глава заставит вас задуматься о том, каким требованиям должен отвечать настоящий робот. Стиральная машина может быть полностью автоматической, запускаться в определенное время по программе, а некоторые современные машины способны уменьшать расход воды путем определения качества и чистоты уже использованной для стирки воды. Однако машины, которые принято называть роботами, на самом деле могут быть просто устройствами на дистанционном управлении. В их числе роботы телеприсутствия и участники «Битв роботов». Несомненно, каждая из таких машин является сложным инженерным продуктом, а для их создания требуются соответствующие высокие навыки.

Одни роботы явно воспринимаются нами как таковые, например ASIMO от Honda и Baxter. В случаях с другими роботами провести черту бывает немного сложнее. Если рассматривать робота как широкую концепцию электро-механической машины, в их ряд не войдут роботы с дистанционным управлением. Если рассматривать их как мобильные машины, то роботом можно будет называть игрушечную машинку на радиоуправлении, но не полностью автономную умную стационарную машину. Машине можно придать человеческие черты. Однако достаточно ли этого для того, чтобы считать ее роботом, если она не сможет выполнять программы и реагировать на окружающую среду?

Теперь, когда мы узнали, что такое робот, предлагаю перейти к следующей главе, в которой мы рассмотрим, как спроектировать робота, чтобы в дальнейшем построить его.

## Задание

Осмотрите вокруг. Скорее всего, в вашем доме найдется множество автоматических машин, выполняющих функции роботов. Возьмите любой бытовой прибор (кроме стиральной машины) и определите его входы и выходы. Затем визуализируйте их связь с контроллером в виде диаграммы. Подумайте, могут ли эти устройства передвигаться по дому, и если да, то каким образом. Затем выясните, какие петли обратной связи могут присутствовать в этой системе. А что насчет управления этими устройствами? Как оно реагирует на команды пользователя?

## Дополнительные материалы

С дополнительными материалами вы можете ознакомиться по следующим ссылкам.

- ASIMO от Honda: <http://asimo.honda.com/>.
- Baxter от Rethink Robotics: <https://www.rethinkrobotics.com/baxter/>.
- Kismet от MIT:  
<http://www.ai.mit.edu/projects/humanoid-robotics-group/kismet/kismet.html>.
- OhBot: <http://www.ohbot.co.uk/>.



- Марсианская научная лаборатория (The Mars Science Laboratory) от NASA: <https://mars.nasa.gov/msl/>.
- Для создания робота-манипулятора, подобного тому, что используется в ArmBot, ознакомьтесь с MeArm: <https://github.com/mimeindustries/MeArm>.
- Узнать больше о моем ArmBot вы можете по адресу: [https://www.youtube.com/watch?v=xY6Oc4\\_jdmU](https://www.youtube.com/watch?v=xY6Oc4_jdmU).

# Глава 2

## Структурные элементы робота – код и электроника

В этой главе мы разберем робота и рассмотрим его детали и системы. Мы поговорим о программном (код, команды и библиотеки) и аппаратном обеспечении, а также о том, как они работают вместе. Приступая к созданию робота, важно заранее продумать какие детали вам понадобятся и как они будут соотноситься друг с другом. Я рекомендую сделать набросок устройства вашего будущего робота – блок-схему, которая послужит руководством к тому, как связать код и детали робота. Данный аспект мы также рассмотрим в этой главе.

Эта глава призвана раскрыть следующие темы:

- внутреннее устройство робота;
- типы элементов робота;
- контроллеры и устройства ввода/вывода;
- планирование элементов и структуры кода;
- планирование аппаратного устройства робота.

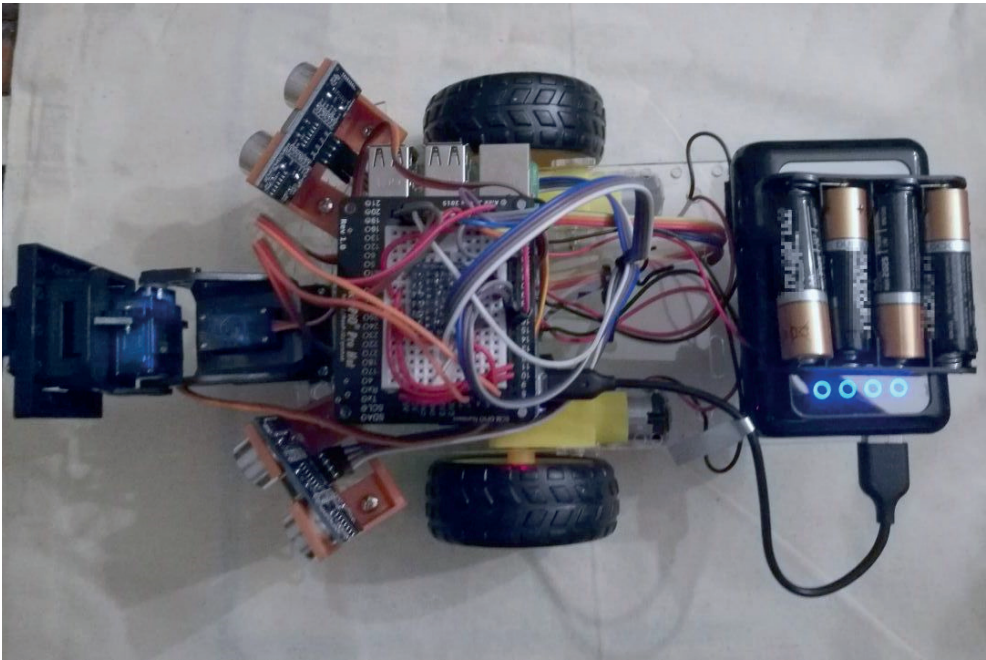
### ТЕХНИЧЕСКИЕ УСЛОВИЯ

Вам потребуются:

- ручка и бумага;
- графический редактор, например Draw.io (необязательно).

### ВНУТРЕННЕЕ УСТРОЙСТВО РОБОТА

Для начала я предлагаю рассмотреть робота как физическую систему. На рис. 2.1 изображен простой любительский робот.



**Рис. 2.1.** Любительский робот в собранном виде

На рис. 2.2 представлено его внутреннее устройство. Компоненты, представленные на рис. 2.2, делятся на 9 типов.

1. *Шасси или корпус* составляют основу конструкции робота. Сюда крепятся все остальные детали.
2. *Поворотное колесо* помогает роботу удерживать баланс.
3. *Два ведущих колеса*. У других роботов может быть больше колес/ног.
4. *Двигатели*, приводящие робота в движение.
5. *Контроллер двигателя* служит посредником между главным контроллером и двигателями.
6. *Главный контроллер*, в данном случае Raspberry Pi, выполняет команды, получает и обрабатывает информацию от сенсоров, на основе чего управляет устройствами вывода, такими как двигатели, посредством контроллера двигателя.
7. *Источник питания* – обычно один или несколько комплектов аккумуляторов.
8. *Сенсоры*, осуществляющие сбор информации об окружающей среде или состоянии физических систем робота.
9. *Устройства отладки* – устройства вывода, позволяющие роботу предоставлять пользователям информацию о коде. Также они могут служить в качестве элемента дизайна.



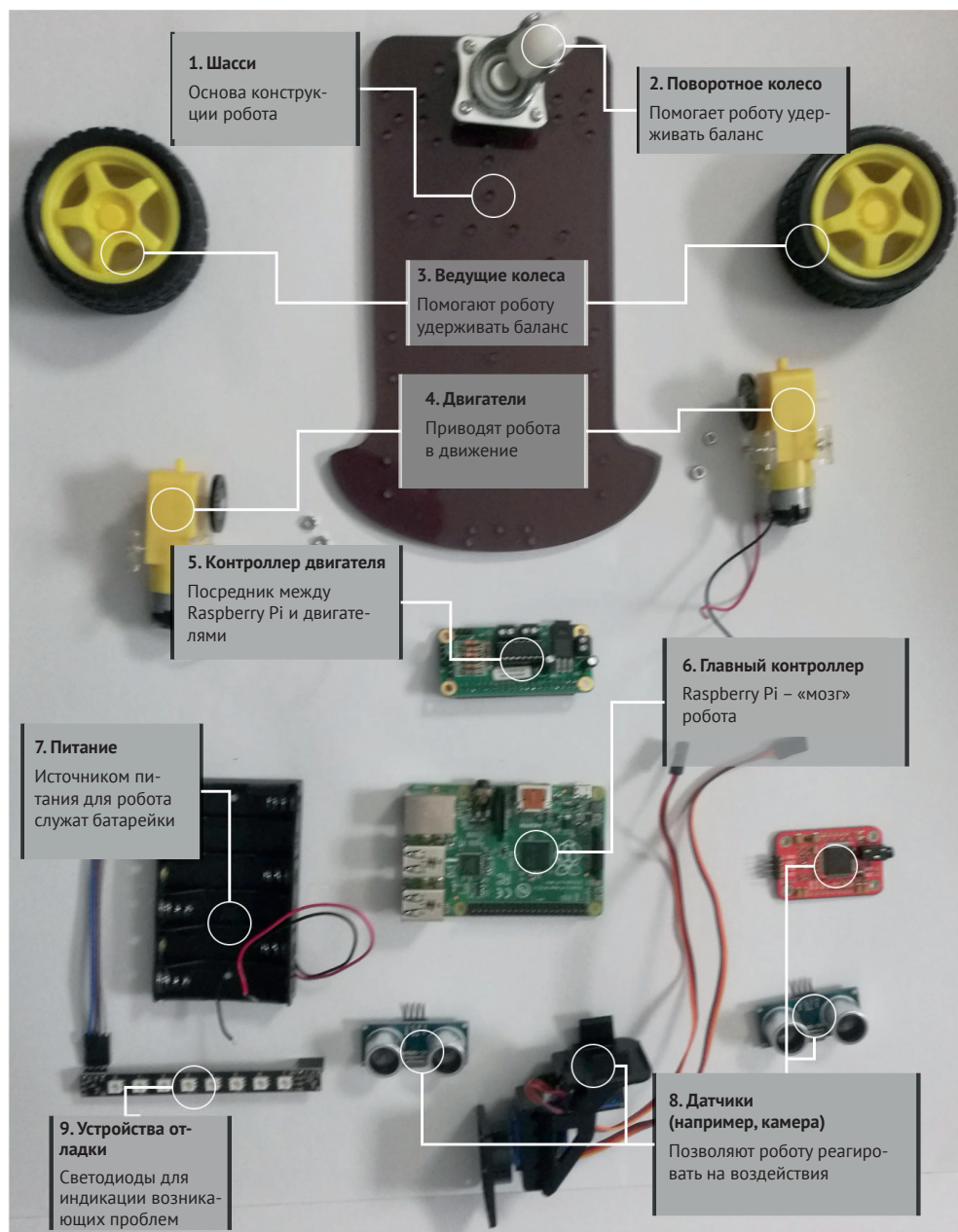


Рис. 2.2. Компоненты любительского робота

Подробнее обо всех компонентах мы поговорим далее в этой главе.

На рис. 2.3 представлена упрощенная блок-схема, где мы можем видеть связанные между собой части робота.

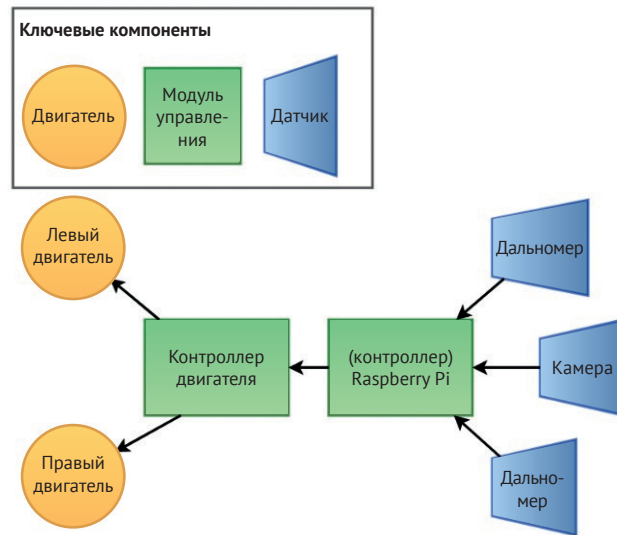


Рис. 2.3. Блок-схема робота

Данная блок-схема не содержит формальных обозначений. Здесь представлено объяснение, которое первым пришло мне в голову, однако оно отражает все ключевые сенсоры, устройства вывода и контроллеры. Блок-схема выглядит так же просто, как набросок на бумаге. Самое главное, что на ней наглядно представлены все аппаратные функциональные блоки, между которыми проходит высокоуровневый поток данных.

На основе этой схемы можно разработать более подробные планы, которые будут содержать информацию об электронике, требования к питанию и аппаратному обеспечению и сведения о том, сколько пространства понадобится для размещения той или иной детали. Первым шагом к созданию робота является набросок его блок-схемы.

### Важное примечание

Блок-схема не является масштабным чертежом готового робота. Здесь не представлены сведения об электронных компонентах, а также отсутствуют многие мелкие детали. Например, здесь не указывается, как подать команду ультразвуковому дальномеру, чтобы он сработал. Линии, соединяющие детали на блок-схеме, представляют поток данных лишь в общих чертах. Блок-схема создана для того, чтобы показать тип и количество двигателей и сенсоров, а также дополнительных контроллеров.

Итак, мы кратко рассмотрели основные компоненты робота (наш будущий робот будет иметь схожие компоненты). Мы провели обзор блок-схемы робота и разобрали ее предназначение. В следующем разделе рассмотрим каждый из компонентов в отдельности, начиная с двигателей.

## Типы компонентов робота

Прежде чем рассматривать типы двигателей и сенсоров, необходимо разобраться, что они из себя представляют.

*Двигатель* – это устройство вывода, которое вращается при подаче питания. Двигатели являются частью *привода*<sup>2</sup>. Привод – это устройство вывода, которое заставляет перемещаться движущиеся части робота за счет подачи электричества. Для управления движением применяются модулированные электрические сигналы. К приводам относятся, например, соленоиды, клапаны и пневматические цилиндры.

*Сенсор* – это устройство ввода, предназначенное для сбора данных об окружающей среде. Существует множество различных типов сенсоров, поэтому описать их все в одной книге не представляется возможным. Мы поговорим о наиболее распространенных и простых в использовании сенсорах. *Дисплеи* и *индикаторы* представляют собой выходные устройства отладки, которые предназначены для обратной связи, т. е. для передачи данных о роботе человеку – пользователю или программисту. В этом разделе рассмотрим некоторые из них.

Далее мы поговорим о двигателях и сенсорах более подробно.

## Типы двигателей

Сейчас мы разберем основные типы двигателей, использующихся в робототехнике. В частности, поговорим о том, как они работают и как их можно применять для разных типов движения.

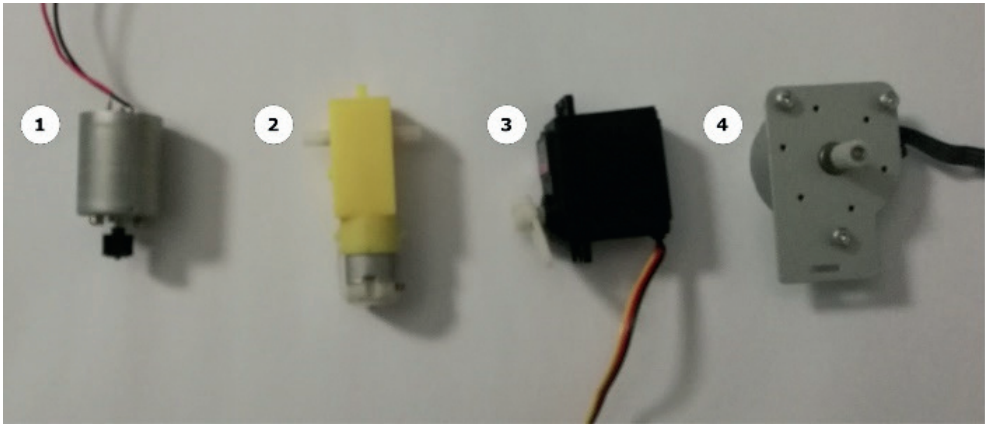
### Важное примечание

Крутящий момент – это сила вращения/скручивания. Например, это сила, которая требуется двигателю для вращения колеса. Чем больше крутящий момент, тем больше мощности (т. е. электрического тока) понадобится двигателю, в связи с чем он может замедлить вращение. Существует **предел крутящего момента**, при достижении которого вал двигателя перестает вращаться.

Рассмотрим, как работает каждый тип двигателя, более детально.

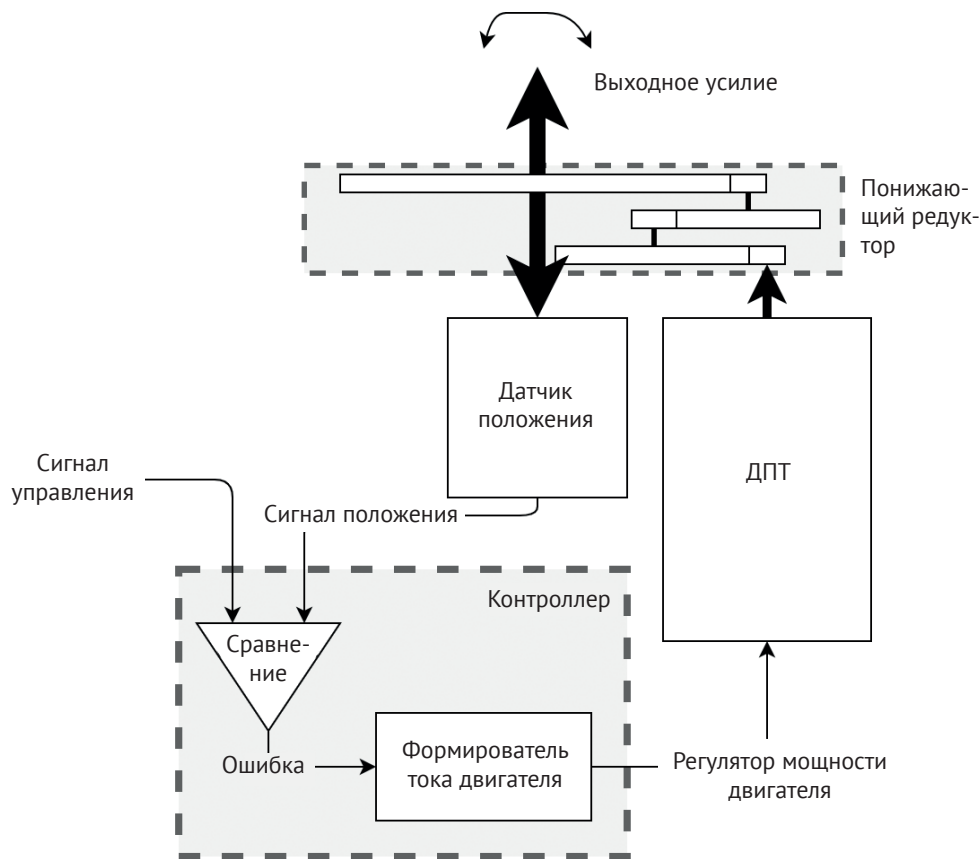
1. Электродвигатель постоянного тока (ДПТ): наиболее простой тип двигателя, применяющийся в робототехнике. Также он может составлять основу редукторных двигателей. Принцип работы ДПТ заключается в преобразовании протекающего через двигатель **постоянного тока** в механическую энергию. Это означает, что его можно привести в действие путем подачи напряжения определенной полярности. Частота вращения двигателя пропорциональна проходящему через него напряжению и крутящему моменту, необходимому для движения. ДПТ без редуктора, представленный на рис. 2.4, может вращаться слишком быстро, ввиду чего он может не развить достаточный крутящий момент и остановиться.

<sup>2</sup> Или исполнительного механизма. – Прим. ред.



**Рис. 2.4.** Различные типы двигателей – электродвигатель постоянного тока, редукторный двигатель постоянного тока, серводвигатель и шаговый двигатель

2. **Редукторный двигатель постоянного тока:** он представляет собой комбинацию ДПТ и редуктора. Редуктор обеспечивает снижение скорости вращения и увеличивает крутящий момент. Благодаря этому двигатель может выдержать большую нагрузку. Обратите внимание, что редукторный двигатель не имеет припаянных выводов! Такой тип двигателей отлично подходит для колес роботов. В главах 6 и 7 мы применим редукторный ДПТ в нашем роботе.
3. **Серводвигатель (или *сервомеханизм*):** такой тип двигателя сочетает в себе редукторный двигатель с датчиком и встроенный контроллер, как представлено на рис. 2.5. Желаемое положение двигателя указывается посредством передачи управляющего сигнала на контроллер. Затем с помощью петли обратной связи с датчиком контроллер пытается достичь этого положения. Серводвигатели используются в механизмах поворота и наклона, а также в роботах-манипуляторах и конечностях роботов. Более подробно мы поговорим о серводвигателях и их программировании в главе 10.
4. **Шаговый двигатель:** имеет несколько обмоток, последовательная активация которых вызывает угловые перемещения двигателя (шаги). Инженеры применяют двигатели такого типа там, где необходима высокая точность движения. Как правило, шаговые двигатели работают медленнее и выделяют больше тепла по сравнению с ДПТ или серводвигателями. Такие двигатели применяются в устройствах с точным управлением, например в 3D-принтерах и высокотехнологичных роботах-манипуляторах. Кроме того, шаговые двигатели гораздо тяжелее и дороже двигателей других типов.
5. **Бесколлекторный (бесщеточный) двигатель:** не представлен на рис. 2.4. Такие двигатели управляются специализированными контроллерами и могут развивать высокую частоту вращения и крутящий момент. Они работают довольно тихо и часто применяются в дронах. Однако, ввиду того что бесколлекторный двигатель не может заменить редукторный ДПТ, дополнительно может потребоваться редуктор.



**Рис. 2.5.** Обобщенная схема механизма серводвигателя

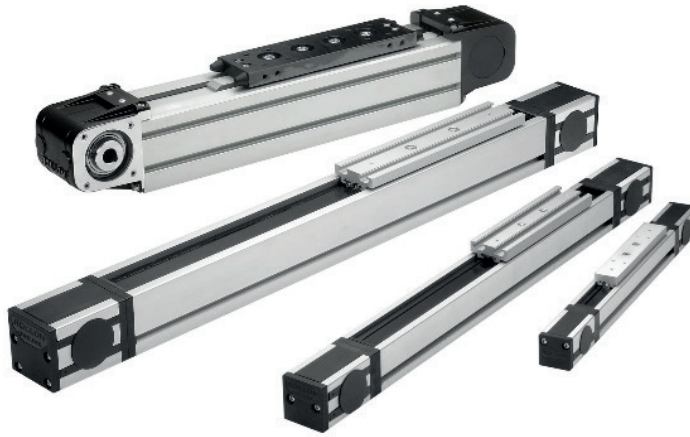
### Важное примечание

Для управления всеми типами двигателей, кроме серводвигателей с помощью Raspberry Pi, требуется специальный контроллер. Он позволяет управлять энергоемкими устройствами без риска повредить главный контроллер. Никогда не подключайте ДПТ, шаговые двигатели или соленоиды напрямую к Raspberry Pi!

Далее мы рассмотрим несколько других типов приводов.

## Другие типы приводов

На рис. 2.6 представлены *линейные приводы*. Они представляют собой устройства, преобразующие электрические сигналы в линейное движение. Такие приводы могут быть оснащены шаговыми двигателями, приводящими в движение винт в фиксированном корпусе, или набором катушек и магнитов.



**Рис. 2.6.** Линейные приводы: автор Rollon91 [Источник изображения: <https://commons.wikimedia.org/wiki/File:Uniline.jpg?uselang=fr> [CC BY-SA 3.0 (<https://creativecommons.org/licenses/by-sa/3.0/>)]

*Соленоид* – это простой линейный привод с электромагнитной обмоткой с металлическим сердечником, который втягивается или выталкивается при включении. Обычно приводы такого типа применяются в гидравлических или пневматических клапанах. Гидравлические и пневматические системы обладают большой мощностью и применяются, например, в экскаваторах.

## Индикаторы состояния – дисплеи, световые и звуковые индикаторы

Одним из наиболее полезных устройств вывода является *дисплей*. Один светодиод (небольшой электронный световой индикатор) может указывать на состояние отдельных частей робота. Массив светодиодов может формировать экран, на который выводится информация разного цвета. Графический дисплей может отображать текст и изображение подобно экрану на мобильном телефоне. В главе 9 мы научимся подключать светодиодную ленту с изменяемым цветом свечения к роботу в качестве дисплея.

Робот может «общаться» с людьми посредством динамиков и генераторов звуковых сигналов. Такие генераторы могут издавать самые разные звуки, начиная от простых шумов и заканчивая речью или воспроизведением музыки.

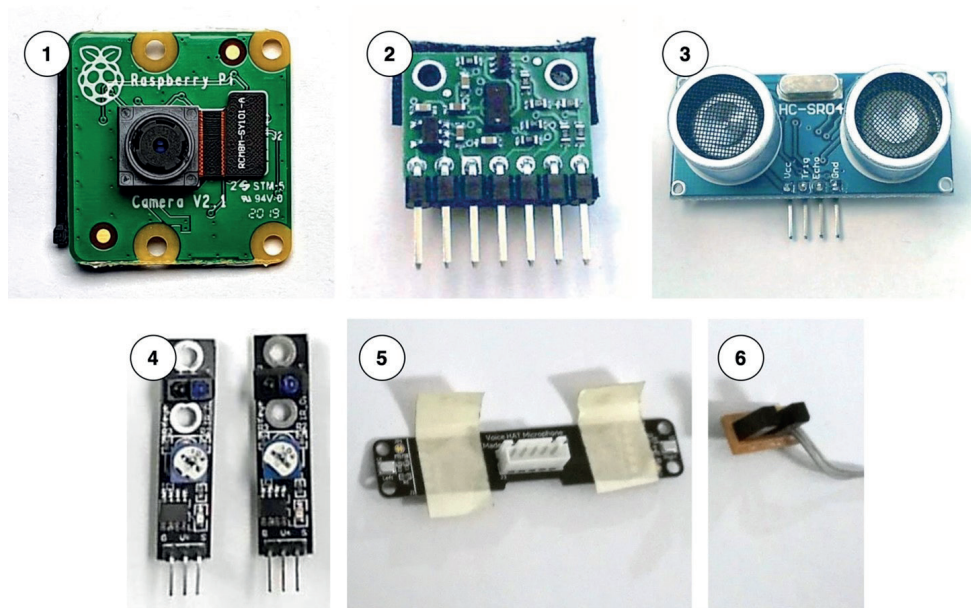
Многие роботы не оснащены дисплеями. Вместо этого к ним подключают телефоны или ноутбуки. Для управления роботом и контроля его состояния мы также будем использовать смартфон. Подробнее об этом поговорим в главе 17.

## Типы сенсоров

На рис. 2.7 представлены типы сенсоров, применяемые в робототехнике. Их мы будем использовать и при создании нашего робота. Давайте рассмотрим некоторые из типов сенсоров и способы их использования. Обратите внимание, что они могут отличаться от сенсоров тех же типов, которые мы видели ранее, – для



выполнения одних и тех же задач могут применяться разные сенсоры. Мы рассмотрим этот аспект более подробно при создании робота.



**Рис. 2.7.** Сенсоры, используемые в моих роботах: 1 – камера для Raspberry Pi, 2 – оптический дальномер, 3 – ультразвуковой дальномер, 4 – датчики линии, 5 – микрофоны и 6 – оптопара

Рассмотрим сенсоры, представленные на рис. 2.7.

1. **Модуль камеры для Raspberry Pi:** подключается к Raspberry Pi, предоставляя компьютеру возможность обработки изображений. Мы будем использовать данный подход при программировании процесса обработки видеоданных в главе 13. Модуль камеры способен захватывать как изображения, так и видеоряд. Он быстро генерирует большой объем данных, что является одной из основных проблем технического зрения. Модуль нуждается в хорошем освещении.
2. **Оптический дальномер:** на рис. 2.7 изображен лазерный дальномер VL53L0X с технологией Time of Flight (ToF). Принцип его работы заключается в следующем: испускаемый устройством лазерный луч отражается от объектов, на основе чего сенсор определяет расстояние до них. Стоит отметить, что такой тип сенсоров чувствителен к условиям освещения. Связь сенсора VL53L0X с Raspberry Pi осуществляется с помощью шины I2C (*Inter-Integrated Circuit*). Благодаря I2C к одним и тем же выводам контроллера можно подключить несколько устройств. Данная технология полезна при наличии большого количества сенсоров и устройств вывода, ввиду чего может не хватать места для их подключения. К слову, наличие поддержки I2C делает плату контроллера ощутимо дороже.
3. **Ультразвуковой дальномер:** HC-SR04 – еще один сенсор для измерения расстояния/дальности. Он определяет расстояние до объекта путем

испускания звуковых импульсов, отражающихся от поверхностей объектов. Точность измерения расстояния зависит от типа материала, из которого выполнен объект. Некоторые материалы, например ткани, поглощают звуковые импульсы, а не отражают их. Также звуковые импульсы могут проходить сквозь такие типы поверхностей, как сетки и решетки, ввиду чего объект станет невидим для сенсора.

Для определения времени распространения звуковой волны HC-SR04 требует реализации точного отсчета времени в контроллере. Позже мы научимся реализовывать это с помощью кода. HC-SR04 дешевле, чем VL-53L0X, и при этом имеет больший диапазон. Однако чувствительность этого сенсора снижается на близких расстояниях. В главе 8 мы научимся программировать ультразвуковые дальномеры.

4. **Датчики линии:** это набор из трех сенсоров, которые используют свет для обнаружения перехода от светлых областей к темным. Такие модули пригодны для работы в самых разных условиях. Существует несколько конфигураций подобных модулей. В зависимости от обнаружения светлой или темной области они включают или выключают выходной сигнал. Такой тип сенсоров является наиболее простым.
5. **Микрофон:** на рис. 2.7 представлена пара микрофонов. Они могут подключаться напрямую к контактам *PCM* (pulse code modulation – импульсно-кодовая модуляция) на *Pi*. В некоторых случаях микрофоны необходимо подключать к электронным устройствам, преобразующим сигнал в понятный для Raspberry Pi формат. В главе 15 мы научимся использовать микрофоны для обработки голоса.
6. **Оптический сенсор прерывания (оптопара):** служит для обнаружения прерывания инфракрасного луча, проходящего через промежуток между фотоизлучателем и фотоприемником. Этот сенсор применяется совместно со щелевым диском для измерения скорости вращения путем подсчета прорезей в диске. Сенсор, состоящий из щелевого диска и оптопары, называется *энкодером*. Подробнее мы рассмотрим энкодеры в главе 11.

На сегодняшний день существует множество различных сенсоров, служащих для распознавания положения конечностей робота, дыма, источников тепла и магнитных полей. Они применяются для создания более продвинутых роботов.

В этом разделе мы на примерах разобрали различные двигатели, дисплеи, индикаторы, сенсоры и их разновидности. Благодаря этим деталям реализуется взаимодействие робота с окружающим миром. Далее рассмотрим контроллеры – детали робота, отвечающие за исполнение кода и выступающие посредником между сенсорами и двигателями.

## КОНТРОЛЛЕРЫ И УСТРОЙСТВА ВВОДА/ВЫВОДА

Как представлено на рис. 2.3, в центре блок-схемы робота находится контроллер. У всех роботов есть главный контроллер – как правило, это компьютер. У некоторых роботов могут быть также вторичные контроллеры. А наиболее



необычные роботы могут иметь сразу много контроллеров. Здесь мы рассмотрим наиболее простой вариант и оснастим будущего робота лишь одним центральным контроллером. Контроллер является связующим звеном между всеми элементами и составляет основу их взаимодействия.

Прежде чем приступить к рассмотрению контроллеров, необходимо более детально изучить один важный компонент, который соединяет контроллер с другими элементами – контактами ввода/вывода.

## Контакты ввода/вывода

Контакты ввода/вывода предназначены для ввода и вывода данных из контроллера. С их помощью контроллер может быть подключен к сенсорам и двигателям.

Количество контактов ввода/вывода на контроллере ограничено. Если к роботу требуется подключить больше устройств ввода/вывода, необходимо использовать вторичные контроллеры. Также вы можете встретить такой термин, как **интерфейс ввода/вывода общего назначения** (*GPIO – general-purpose input/output*). Контакты ввода/вывода на контроллере имеют разные функциональные возможности.

Простейшие контакты ввода/вывода могут только выводить или считывать сигнал высокого и низкого уровня, как представлено на рис. 2.8. Их называют цифровыми контактами ввода/вывода. Их можно запрограммировать для выполнения сложных задач с помощью последовательности импульсов. Такой же принцип применяется в ультразвуковом дальномере HC-SR04. График на рис. 2.8 представляет собой изменение уровня напряжения с течением времени. Итак, когда мы движемся по оси *x*, напряжение находится на оси *y*. Высокий уровень соответствует логической единице (1, True, On), а низкий – логическому нулю (0, False, Off). Контроллер будет стремиться интерпретировать любое напряжение на входе как высокий или низкий логический уровень.



Рис. 2.8. Цифровой сигнал

Аналоговые контакты ввода/вывода способны считывать сигналы с меняющимся уровнем, как представлено на рис. 2.9, где также изображен график зависимости напряжения от времени. Если сенсор выдает изменяющееся сопротивление или непрерывно меняющееся напряжение, значит, здесь необходим аналоговый вход. Также существует такое понятие, как *предел разрешающей способности*; например, 8-битный аналоговый ввод сможет читать 256 возможных значений.

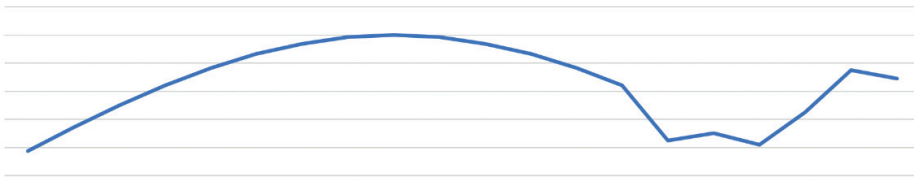


Рис. 2.9. Аналоговый сигнал

Контакты **широтно-импульсной модуляции (ШИМ)** (*PWM – pulse-width modulation*) выводят циклический цифровой сигнал, как представлено на рис. 2.10. Здесь тоже показано изменение напряжения с течением времени. Пунктирной линией обозначается зависимость непрерывного (аналогового) уровня сигнала от времени. Посредством кода мы можем задавать частоту и длительность импульсов на контактах ШИМ. Изменение продолжительности времени пребывания во включенном и выключенном состоянии в цикле позволяет менять выходной сигнал. Такой подход часто применяется для управления частотой вращения двигателя.

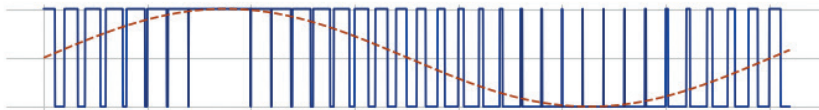


Рис. 2.10. Синяя линия – сигнал ШИМ, пунктирная линия – его усредненное значение

Подробнее о контактах ШИМ мы поговорим в главе 6.

Из контактов ввода/вывода формируются линии передачи данных, такие как последовательные магистральные шины, шины *I2S (Integrated Inter-chip Sound)*, *I2C* и *SPI (Serial Peripheral Interface – последовательный периферийный интерфейс)*. Также их называют шинами передачи данных. Такие шины используются для двусторонней передачи данных другим контроллерам и интеллектуальным сенсорам. В главе 9 мы будем использовать шину *SPI* для RGB светодиодов.

Для цифрового или аналогового ввода/вывода или для части шины передачи данных могут применяться выводы микроконтроллера. Многие контроллеры позволяют настраивать режим работы выводов при помощи программного обеспечения, которое вы запускаете на них. Однако у некоторых выводов выбор режимов значительно ограничен.

## Контроллеры

Обладая соответствующими навыками, вы можете создавать электронные компоненты робота и печатные платы, приобретая по отдельности микроконтроллеры и остальные детали. Однако в этой книге мы упростим задачу и будем использовать модули контроллеров. Они, как правило, представляют собой скомпонованные и простые в использовании системы.

На рис. 2.11 представлены некоторые из моих любимых контроллеров. Источником питания для каждого из них может выступать USB. Также с помо-

щью USB можно программировать все представленные контроллеры, кроме Raspberry Pi. Каждый из них оснащен коннекторами для доступа к контактам ввода/вывода. Далее мы рассмотрим модули каждого из контроллеров, а также их преимущества и недостатки.

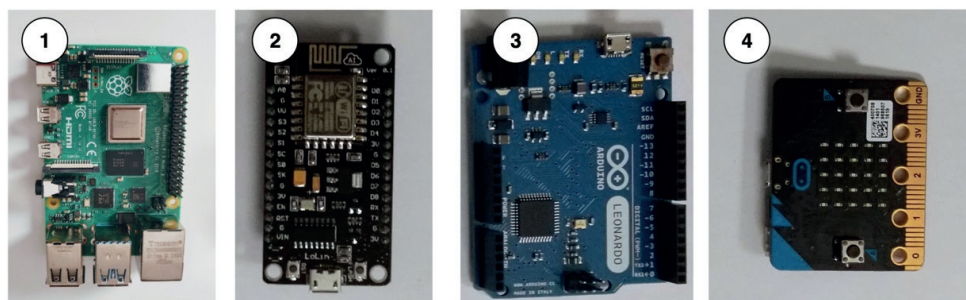


Рис. 2.11. Модуль контроллеров: Raspberry Pi, NodeMCU, Arduino и micro:bit

1. **Raspberry Pi**: этот контроллер достаточно мощный и подходит для обработки видеоданных. Как правило, он потребляет больше энергии и стоит дороже. Однако по функциональным возможностям такой контроллер может быть сравним с мобильным телефоном. Он обладает наиболее гибкой средой для программирования. Существует несколько моделей контроллера Raspberry Pi. Все они имеют множество контактов ввода/вывода, но ни один из них не имеет аналоговых входов.
2. **NodeMCU**: этот контроллер представляет собой платформу на основе модуля ESP8266, поддерживающего протоколы Wi-Fi. Для программирования этого контроллера можно использовать Arduino C++, MicroPython или Lua. Он имеет множество контактов ввода/вывода, однако только один из них способен принимать аналоговые сигналы. Также он поддерживает различные виды шин передачи данных. Данный контроллер несколько быстрее и имеет больший объем памяти, чем Arduino. Среди всех представленных контроллеров NodeMCU является самым дешевым.
3. **Arduino Leonardo**: контроллер на основе чипа Atmega 32u4. Модули контроллеров Arduino легли в основу большинства роботов, созданных мной в период 2010–2012. Данный контроллер интересен своей простотой в использовании. Его можно подключить к ПК с помощью USB и запрограммировать на взаимодействие с устройствами, подключенными к его контактам ввода/вывода. Как правило, для программирования Arduino используют язык C++. Встроенные контакты ввода/вывода (семь аналоговых контактов, несколько цифровых контактов, ШИМ-контакты вывода) легко программируются на обработку большинства шин передачи данных. Процессор Arduino очень прост, ввиду чего он не подходит для задач обработки видеоданных и обработки речи. Из всех представленных контроллеров Arduino обладает самым низким показателем энергопотребления.
4. **micro:bit**: этот контроллер, выпущенный в 2015 году, предназначен для применения в образовательных целях и идеально подходит для обуче-

ния детей. По умолчанию данный контроллер имеет 3 контакта ввода/вывода, однако, если при строительстве робота вам понадобится больше чем 3 контакта, можете подключить дополнительный адаптер. Несмотря на это, контроллер является довольно мощным, а также оснащен встроенной светодиодной матрицей. Он программируется посредством MicroPython, C, JavaScript и нескольких других языков.

Помимо контроллеров, представленных в этой главе, стоит упомянуть и микроконтроллер *PIC* (*peripheral interface controller* – контроллер интерфейса периферии). Он стал применяться робототехниками-любителями задолго до других контроллеров и по сей день имеет популярность у членов робототехнического сообщества.

В табл. 2.1 приводится сравнение описанных выше контроллеров, а также их преимущества и недостатки.

**Таблица 2.1.** Преимущества и недостатки контроллеров

Контроллер	Преимущества	Недостатки
Arduino	Низкое энергопотребление, настраиваемые контакты ввода/вывода	Относительно большой размер, низкие возможности процессора, не подходит для обработки видеоданных и речи. Ограниченные возможности среды программирования
Micro:bit	Светодиодная матрица, легкость в программировании	Необходимость применения дополнительного адаптера для подключения большего количества устройств, не подходит для обработки видеоданных и речи. Ограниченные возможности контактов ввода/вывода
ESP8266/ NodeMCU	Поддержка многих языков программирования, поддержка Wi-Fi-протоколов, относительно низкая цена, настраиваемые контакты ввода/вывода, небольшой размер (как правило)	Не подходит для обработки видеоданных и речи. Наличие лишь одного аналогового входа
Raspberry Pi	Высокая производительность. Удобный доступ к контактам ввода/вывода. Возможность запуска полнофункциональной системы Linux. Поддержка многих языков программирования. Возможность применения для обработки видеоданных и речи. Поддержка Wi-Fi-протоколов и Bluetooth	Необходимость использования дополнительных устройств для реализации аналогового входа. Как правило, более высокая стоимость и энергопотребление

В то время как другие контроллеры способны запускать лишь простые интерпретаторы или скомпилированный код, Raspberry Pi может запускать полнокомплектную операционную систему. Существующие на сегодняшний день модели данного контроллера имеют поддержку Wi-Fi-протоколов и Bluetooth. Благодаря этому мы можем изготовить полностью автономного робота, управляемого джойстиком.

## Модели контроллера Raspberry Pi

На рис. 2.12 представлены существующие модели контроллера Raspberry Pi. По мере выпуска новых Raspberry Pi робототехнику-любителю, возможно, придется адаптировать свой проект к последней версии. Все представленные модели поддерживают Wi-Fi и Bluetooth. Контакты ввода/вывода Raspberry Pi поддерживают многие типы шин данных и цифровой ввод/вывод. Для считывания аналоговых сигналов и реализации некоторых других функций ввода/вывода потребуются внешние контроллеры.

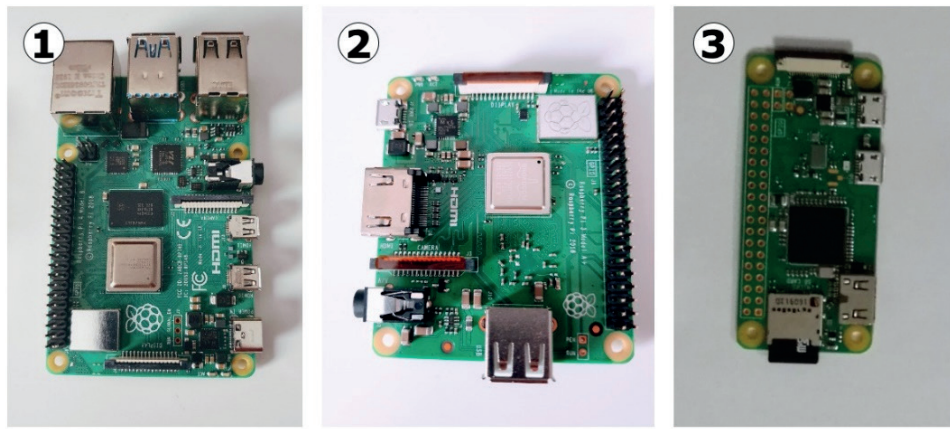


Рис. 2.12. Модели Raspberry Pi: 4B, 3A+ и Zero W

Давайте рассмотрим каждую из моделей более подробно.

1. **Raspberry Pi 4B:** самая мощная и быстрая модель, последняя разработка в линейке Raspberry Pi на момент написания этой книги. Обладает относительно большим размером и является наиболее дорогим из представленных моделей. Также имеет высокий показатель энергопотребления.
2. **Raspberry Pi 3A+:** модель, которую мы будем использовать при создании нашего робота. Представляет собой идеальное сочетание размера и производительности. Пригодна для обработки видеоматериалов и речи. Данная модель уступает по скорости 4B+, однако для нашего проекта этого будет достаточно.
3. **Raspberry Pi Zero W:** недорогая и более легкая альтернатива другим моделям, способная обрабатывать данные с камер и вывод звука на динамик. Версия Zero WH оснащена коннекторами для линий ввода/вывода. Данная модель справляется с задачами распознавания речи и видеоматериалов медленнее, чем Raspberry Pi 3 и 4. Однако благодаря небольшим размерам данная модель может применяться для создания устройств дистанционного управления.

В табл. 2.1 приведено сравнение (преимущества и недостатки) упомянутых выше моделей.

**Таблица 2.1.** Преимущества и недостатки моделей контроллера Raspberry Pi

Модель	Преимущества	Недостатки
Raspberry Pi 4B	Наиболее быстрая, самый большой объем памяти. Способно быстро обрабатывать видеоданные. Оснащена 4 USB-портами (в том числе 2 высокоскоростными портами USB 3) и разъемом HDMI	Подвержена перегреву. Быстрая разрядка аккумулятора. Наиболее дорогая. Больше и тяжелее остальных моделей
Raspberry Pi 3A+	Значительно быстрее, чем Zero W. Меньшее потребление энергии аккумулятора. Меньше, легче и дешевле, чем 4B. Оснащена одним USB-портом и одним разъемом HDMI	Больше и тяжелее, чем Zero W. Медленнее, чем 4B, а также обладает меньшим объемом памяти
Raspberry Pi Zero W	Маленькая, легкая и недорогая. Наиболее низкое энергопотребление	Самая медленная модель линейки. Выводы коннектора требуют пайки (кроме модели WH). Требуется адаптер для использования портов USB OTG и HDMI. Меньший CSI-порт для камеры

Несмотря на отличные показатели производительности Raspberry Pi 4B, модель 3A+ также отлично подойдет для всех решений, которые мы реализуем в нашем роботе.

## НАБРОСОК КОМПОНЕНТОВ И СТРУКТУРЫ КОДА

В предыдущих разделах вы вкратце ознакомились с некоторыми компонентами роботов, а также рассмотрели блок-схемы, представляющие соединения компонентов. Сейчас можете приступить к следующему этапу создания робота и продумать соединения компонентов и структуру написанного для них кода.

Структурировать код гораздо проще, если представить его в виде логических блоков. При создании более сложных проектов вы можете структурировать код способами, аналогичными разработке функциональных схем аппаратного обеспечения.

Итак, вернемся к блок-схеме робота, представленной на рис. 2.3, и попытаемся выяснить, какой код нам необходимо написать. На блок-схеме мы видим три сенсора и два выхода. Код может потребоваться для каждого компонента (датчика, выхода, платы контроллера). Также нам необходим код для программирования поведения связанных модулей.

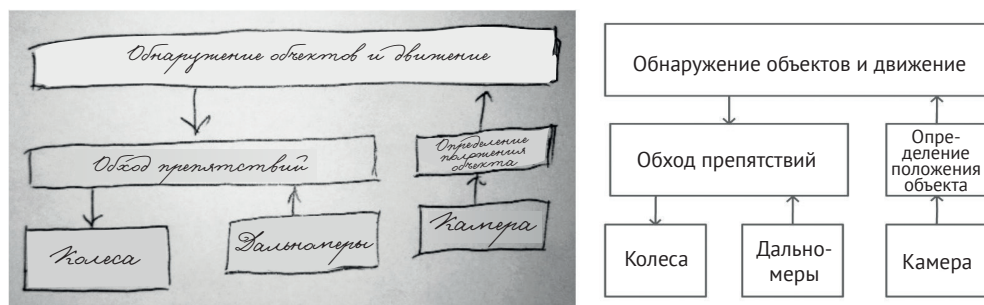
Контроллеры двигателей бывают разных видов. Для каждого из них характерны свои способы управления двигателем, а также особенности контроля уровня заряда батареи. Некоторые интеллектуальные контроллеры двигателей могут напрямую взаимодействовать с колесными энкодерами, чтобы контролировать соответствие положения колес заданным значениям. Я не рекомендую писать код для контроллера двигателя, поскольку это усложнит процесс замены компонентов робота и структурирование его кода. Вместо этого я предлагаю внедрить абстрактный промежуточный интерфейсный уровень – что-то между реальным кодом контроллера двигателя и стандартным интер-



фейсом. Такой подход в случае необходимости позволит облегчить замену элементов робота. Мы рассмотрим его на практике в главе 7.

Практически то же самое мы будем делать и с сенсорами. Для каждого из них напишем код, который позволит нам регулировать сбор и обработку данных. Каждому сенсору может соответствовать свой стартовый (настроечный) и рабочий код, который должен исполняться при запуске или завершении определенной программы поведения. Одним из сложных примеров является камера, поскольку данным, которые мы можем использовать для выполнения задачи, требуется предварительная обработка значений.

Так же как и структуру оборудования робота, программное обеспечение мы можем представить в виде простой схемы. Вы можете сделать это в любом графическом редакторе или просто нарисовать на бумаге. На рис. 2.13 я намеренно представил вариант, нарисованный «от руки», чтобы показать, что для его создания вам не потребуется ничего, кроме бумаги и ручки. Разумеется, в таком исполнении блок-схема может выглядеть слегка неаккуратно, зато вы легко перерисуете ее при необходимости. Если же идея посетит вас во время обеда, вы сможете нарисовать ее даже на обратной стороне чека. Рисую карандашом, не забудьте обвести текст ручкой или тонким маркером, чтобы ваш набросок не потускнел. Для большей наглядности на рис. 2.13 я также представил рисунок, созданный в графическом редакторе (но можно обойтись и без него).



**Рис. 2.13.** Блок-схема программного обеспечения, написанная ручкой на бумаге, и та же схема, созданная на компьютере

### Совет

Можете сканировать свои рукописные документы. Если у вас есть сканер или смартфон с такой функцией, я рекомендую отсканировать или сфотографировать ваши наброски для дальнейшего использования. Поместив их в Evernote или OneNote в виде изображений или PDF-файлов с соответствующими тегами, позже вы легко отыщете их.

Сделав набросок «от руки», далее можете перерисовать его в графическом редакторе (но на это потребуется чуть больше времени). К слову, постарайтесь поменьше отвлекаться на украшение схемы при помощи инструментов редактора.

Что касается самой блок-схемы, так или иначе она будет представлять собой упрощенное представление устройства робота. Поле «**Колеса**» означает блок

кода, взаимодействующий с контроллером двигателя колес. Такой код можно написать поверх кода контроллера двигателя, предустановленного компанией-производителем. Также можно использовать контакты ввода/вывода, подключенные к контроллеру.

Поле **«Дальномеры»** представляет собой блок кода для определения расстояния до предметов на основе данных полученных от сенсора, а также для активации этого сенсора при необходимости. Мы рассмотрим два разных типа сенсоров и сравним их. При внесении изменений в блок кода сенсора на этом уровне весь остальной код менять не придется.

Поле **«Камера»** представляет блок кода камеры. Этот код выполняет такие сложные функции, как настройка камеры, настройка разрешения, баланса белого и т. д. Поверх этого уровня находится уровень, который будет обрабатывать изображение с камеры. Он может определить положение цветного объекта и отправить соответствующие данные уровню, расположенному еще выше.

Рядом с уровнями двигателей и дальнометрических сенсоров находится *поведенческий уровень*, позволяющий роботу избегать столкновений. При возникновении риска столкновения принимаются поведенческие решения развернуться и немного отъехать.

На высоком уровне осуществляется принятие поведенческих решений на основе позиционных данных кода **«Определение положения объекта»**. На основе этих данных осуществляется выбор направления, а затем формируются команды управления двигателем. При наличии поведенческой задачи обхода препятствий может возникнуть сложная форма взаимодействия, при котором робот будет искать нужный объект, в то же время объезжая препятствия и предметы. В таком случае он не сможет подъехать достаточно близко к нужному объекту.

Каждый модуль относительно прост. Возможно, что более сложными являются нижние уровни, расположенные ближе к аппаратному обеспечению (особенно это касается камер).

Разбивка кода на блоки позволяет нам работать над ними по отдельности. Настроив и протестировав один, можно сосредоточиться на другом. Также подобные блоки кода могут применяться повторно, что может пригодиться, например, при программировании двигателя. При таком подходе вам не придется писать один и тот же код несколько раз.

Разделение программного обеспечения на блоки позволяет нам настраивать их и взаимодействие между ними разными способами. Можно выбрать, что использовать для этих блоков – функции, классы или сервисы. Я расскажу об этом подробнее, когда мы начнем писать код и рассмотрим различные подходы.

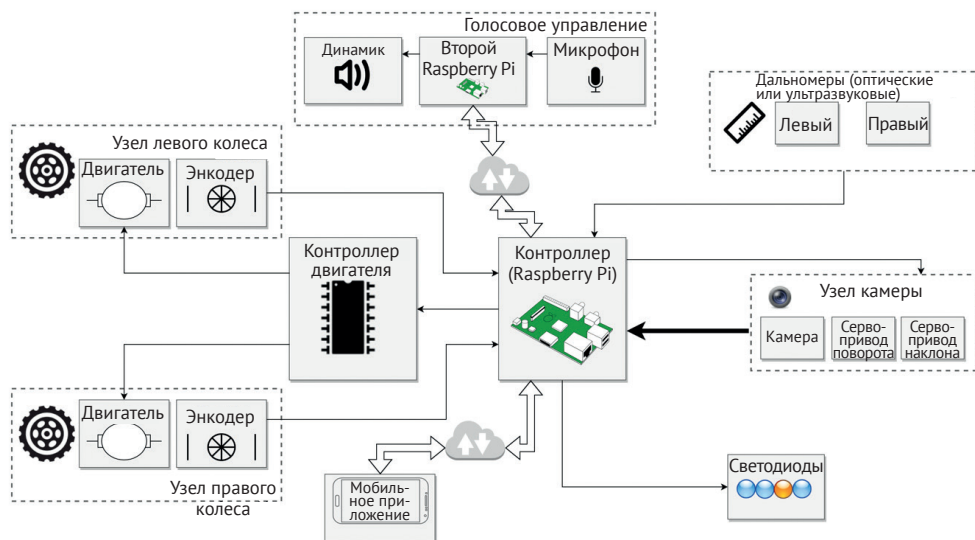
## РАЗРАБОТКА АППАРАТНОГО УСТРОЙСТВА РОБОТА

Далее применим полученные знания и спланируем расположение физических компонентов робота. С каждой новой главой мы будем добавлять новые компоненты, поэтому важно держать в голове их общую схему, чтобы понять на каком этапе мы находимся. Мечтать о том, какие функции будет выполнять робот, очень увлекательно. Наш робот будет обладать следующими характеристиками:



- у него будут колеса, и он сможет передвигаться по полу;
- он будет оснащен контроллером Raspberry Pi 3A+;
- он будет иметь контроллер двигателя для колес;
- он сможет показывать свое состояние посредством разноцветных светодиодных индикаторов;
- робот будет оснащен парой серводвигателей для реализации механизма поворота и наклона;
- благодаря ультразвуковым или лазерным датчикам робот сможет избегать столкновений со стенами и объезжать препятствия;
- на каждом колесе будет установлен энкодер, который будет передавать данные об изменении положения контроллеру;
- посредством камеры робот сможет распознавать цветные объекты и лица;
- посредством камеры он сможет следовать по линиям на полу;
- он будет оснащен микрофоном и динамиком для реализации голосового управления;
- в качестве пульта дистанционного управления для робота выступит джойстик;
- для всего вышеперечисленного роботу потребуется питание.

Ну и ну! У нашего робота достаточно большие функциональные возможности. Теперь необходимо сделать набросок аппаратных блоков. На рис. 2.14 представлена наша блок-схема. Я создал ее с помощью Draw.io – отличный вариант для начального этапа. При создании большинства своих роботов я начинаю именно с этого.



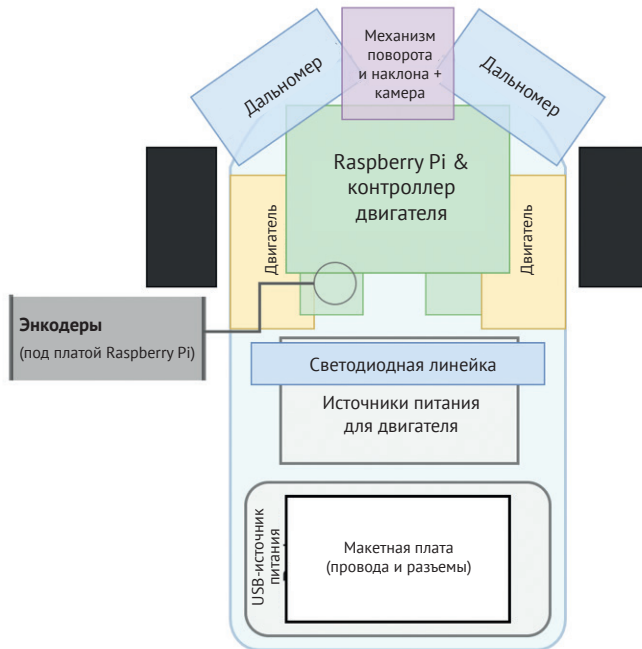
**Рис. 2.14.** Блок-схема нашего робота, созданная в веб-приложении draw.io

Ввиду наличия множества элементов блок-схема выглядит несколько устрашающе. Однако мы будем рассматривать по одной функциональной области

в каждой главе. Записи на блок-схеме не являются формальными обозначениями, это просто способ визуализировать все части, которые необходимо будет соединить. Помимо надписей, я рекомендую разметить места физического размещения сенсоров и других компонентов.

Набросок на рис. 2.15 не является исчерпывающе точным и масштабным. Он отражает представление о том, где я хотел бы разместить детали. Здесь стоит обратить внимание на следующие моменты:

- ничто не загромождает поле зрения сенсоров, а датчики немного повернуты в стороны. Я расскажу более подробно о том, почему это важно, в главах, посвященных сенсорам;
- энкодеры размещены над колесами, позицию которых они будут измерять;
- тяжелые элементы, особенно батареи, следует размещать внизу (ниже центра тяжести), чтобы робот не опрокинулся;
- батарейки нужно менять, поэтому они должны быть в легком доступе;
- старайтесь размещать компоненты, которые подключены напрямую, достаточно близко друг к другу;
- это лишь приблизительный набросок, а не план для окончательной сборки. Он не обязательно должен быть подробным.



**Рис. 2.15.** Пример эскиза размещения физических компонентов робота (нарисован при помощи Draw.io)

По мере прочтения этой книги вы изучите все детали, представленные на блок-схеме. Постепенно в голове начнет вырисовываться устройство реально-

го будущего робота. Любую подобную схему следует воспринимать как приблизительную. Она не является точным чертежом, и слепо следовать ей нельзя. Блок-схема представляет собой, скорее, схематическую карту, который следует руководствоваться на начальном этапе проекта.

## Выводы

В этой главе мы рассмотрели ряд компонентов, входящих в состав робота. Теперь вы можете приступить к созданию плана в виде блок-схемы и начинать визуализировать устройство своего робота. Здесь вы узнали, что свои идеи можно быстро набросать на бумаге, а затем привести их в более аккуратный вид с помощью графического редактора. Вы ознакомились с датчиками и контроллерами, а также с несколькими способами связи контроллеров с другими подключенными к ним устройствами – аналоговыми и цифровыми контактами, контактами ШИМ и шинами передачи данных. На основе всего вышеперечисленного мы создали приблизительный план нашего будущего робота.

В следующей главе рассмотрим и начнем настройку Raspbian – операционной системы для Raspberry Pi, которую мы будем использовать в нашем проекте.

## УПРАЖНЕНИЯ

1. Попробуйте создать блок-схему для другого робота, продумав реализацию входов, выходов и контроллеров.
2. Являются ли Raspberry Pi 4B и 3A+ самыми передовыми моделями в линейке? Какую из них вы бы выбрали? Расскажите о компромиссах.
3. Каковы недостатки лазерного дальномера в сравнении с ультразвуковым?
4. Попробуйте нарисовать примерную схему размещения компонентов для другого типа робота с другим контроллером.

## Дополнительные материалы

- «*Raspberry Pi Sensors*», Руши Гаджар (*Rushi Gajjar*), Packt Publishing: научитесь интегрировать сенсоры в свои проекты с Raspberry Pi и позвольте вашему мощному микрокомпьютеру взаимодействовать с окружающим миром.
- «*Make Sensors: A Hands-On Primer for Monitoring the Real World with Arduino and Raspberry Pi*», Теро Карвинен (*Tero Karvinen*), Киммо Карвинен (*Киммо Карвинен*), Вилле Валтокари (*Ville Valtokari*), Maker Media, Inc.: научитесь использовать сенсоры для взаимодействия контроллера Raspberry Pi или Arduino с окружающим миром.
- «*Make: Electronics: Learning Through Discovery*», Чарльз Платт (*Charles Platt*), Make Community, LLC: полезный ресурс, если вы хотите узнать больше об электронных компонентах робота и изучить другие элементы более подробно.

# Глава 3

## Изучение Raspberry Pi

В предыдущей главе мы уже встречали **Raspberry Pi** при рассмотрении робота в разобранном виде. Не удивительно, что наш робот также будет оснащен этим микрокомпьютером.

В качестве **контроллера** для нашего робота выступит **Raspberry Pi 3A+**. В этой главе мы рассмотрим подключение устройств к Raspberry Pi. Затем перейдем к изучению операционной системы (ОС) **Raspberry Pi OS**. И в заключение займемся вопросами подготовки ОС для дальнейшего использования в Raspberry Pi.

В этой главе раскрыты следующие темы:

- функциональные возможности Raspberry Pi;
- выбор способов подключения;
- операционная система (ОС) Raspberry Pi;
- запись образа ОС Raspberry Pi на SD-карту.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам понадобится:

- карта microSD емкостью 16 ГБ или более;
- Raspberry Pi 3A+;
- компьютер или ноутбук на Windows, Linux или macOS с доступом к интернету, способный считывать/записывать информацию на SD-карту;

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/3bBJQt9>.

### Функциональные возможности RASPBERRY PI

В главе 2 мы узнали, что выбор контроллера является чрезвычайно важным этапом создания робота. Он определяет дальнейший выбор входов и выходов, сенсоров, а также структуру кода и требования к питанию. Замена контроллера может потребовать написания нового кода, а изменение конструкции контроллера может повлиять на требования к питанию.

Raspberry Pi – это линейка микрокомпьютеров, предназначенных для использования в образовательных целях. Каждая модель Raspberry Pi является полноценным компьютером, а также имеет контакты ввода/вывода для подключения других устройств. Все это делает данный микрокомпьютер фаворитом пользователей (тех, кто увлекается созданием роботов и гаджетов). Также их при-

влекает небольшой (относительно стандартных вычислительных устройств) размер и стоимость. Ко всем моделям Raspberry Pi можно подключить камеру, монитор и клавиатуру. Также все они могут быть подключены к сети.

## Скорость и производительность

Raspberry Pi достаточно производителен и способен справляться с задачами обработки видеоматериалов, такими как распознавание лиц и отслеживание объектов. Новые модели в линейке делают это достаточно быстро. Также Raspberry Pi может распознавать голосовые команды. Для упомянутых задач лучше всего подходят модели 4B, 3B+ и 3A+. Модели Zero и Zero W справляются с ними намного медленнее.

Raspberry Pi – это **одноплатный компьютер** (SBC – Single-Board Computer), на который можно установить полноценную операционную систему, например некоторые версии **Linux**. Подробнее мы поговорим об этом в разделе «*Операционная система Raspberry Pi*». Стоит отметить, что благодаря этой возможности и поддержке многих библиотек и инструментов мы сможем написать код на **Python** для обработки видеоданных и голоса. Другие микроконтроллеры, такие как **Arduino**, **Esp8266** и **micro:bit**, не обладают таким функционалом.

Такие одноплатные компьютеры, как **BeagleBone**, **CHIP**, **OnionIoT** и **Gumstix Linux**, также способны запускать Linux. Однако они дороже, чем Raspberry Pi, и при этом предоставляют меньше возможностей. Лишь некоторые из них имеют возможность подключения камеры. Несмотря на то что BeagleBone обладает функциями аналогового ввода/вывода, Raspberry Pi 3A+ более универсален и имеет больше возможностей для расширения.

## Возможности подключения

Raspberry Pi 3A+ оснащен портами USB и HDMI. Они не сыграют важной роли в создании нашего робота, но могут пригодиться при отладке или решении некоторых проблем, таких как потеря связи с роботом. Для этой цели я рекомендую дополнительно приобрести монитор и клавиатуру.

Модели Raspberry Pi 4, 3 и Zero W поддерживают Wi-Fi и Bluetooth. К нашему роботу мы будем подключаться посредством Wi-Fi, поэтому я рекомендую использовать модель, которая поддерживает эту технологию. Wi-Fi можно использовать для программирования и управления роботом, а также для запуска кода.

Raspberry Pi имеет контакты ввода/вывода, позволяющие подключать его к датчикам. В отличие от моделей Raspberry Pi Zero и Zero W, в 3A+ разъемы **GPIO** готовы к использованию, так как все контакты (также известные как линейные разъемы, или «гребенки») уже припаяны. Первые платы Raspberry Pi имели разные разъемы ввода/вывода. Именно поэтому наилучшими вариантами являются модели 3-й и 4-й серий.

## Преимущества Raspberry Pi 3A+

Raspberry Pi 3A+ представляет собой полноценный компьютер. Его функциональные возможности отвечают всем нашим требованиям:

- наличие входов/выходов;
- наличие разъема для подключения камеры;

- возможность обработки видеоматериалов и речи;
- поддержка Wi-Fi и Bluetooth;
- поддержка кода на Python;
- наличие припаянных контактов для подключения к роботизированным устройствам;
- небольшой размер и стоимость.

В дополнение к этому модель 3A+ имеет четырехъядерный ARM-процессор с тактовой частотой 1,4 ГГц. Такие характеристики прекрасно подходят для нашего проекта.

## ВЫБОР СПОСОБОВ ПОДКЛЮЧЕНИЯ

При создании робота мы будем использовать определенные способы подключения, доступные в Raspberry Pi. Давайте посмотрим, что это за подключения и как их применять. При непосредственном подключении сенсоров и деталей к Raspberry Pi мы рассмотрим необходимые контакты подробнее, а сейчас проведем их краткий обзор.

На рис. 3.1 выделены необходимые нам разъемы.

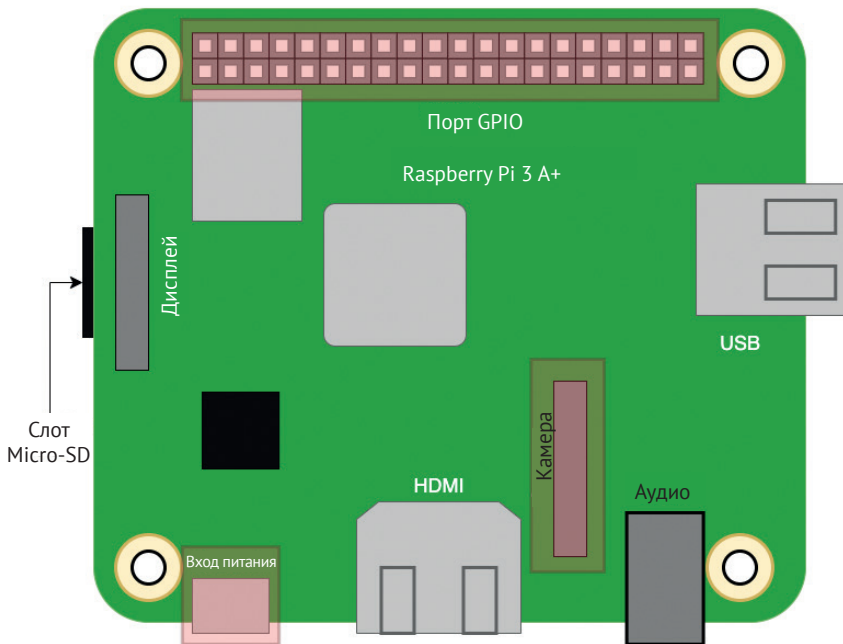


Рис. 3.1. Разъемы на плате Raspberry Pi

Разъем питания расположен в нижнем левом углу схемы и обозначен как «Вход питания». Подача питания осуществляется через разъем Micro-USB, аналогичный тому, что есть на многих телефонах. Наряду с подачей питания такие разъемы могут использоваться при реализации естественного интерфейса. К разъему Micro-USB можно подключить USB-аккумуляторы, отвечаю-

щие определенным требованиям мощности. Для Raspberry Pi требуется источник питания на 2,5А (обычно достаточно не менее 2А).

Разъем, расположенный в нижней части справа (ближе к центру схемы), – это **порт камеры** (*последовательный порт камеры – Camera Serial Interface, CSI*). К нему мы будем подключать **камеру** для распознавания объектов и обработки полученных данных.

Для установки ОС и обеспечения возможности исполнения кода нам потребуется слот для карт microSD. Как упоминалось ранее, связь с роботом будет осуществляться посредством Wi-Fi, поэтому разъемы Ethernet и HDMI нам не пригодятся<sup>3</sup>. Большой разъем, расположенный в верхней части схемы на рис. 3.1, – это порт GPIO.

На рис. 3.2 порт GPIO представлен в более подробном виде. Здесь указаны названия и назначение некоторых контактов. К этому порту мы будем подключать большинство сенсоров и двигателей. Внешние устройства подключаются с помощью шин **SPI**, **I2C**, **Serial** и **I2S** или цифровых контактов ввода/вывода.

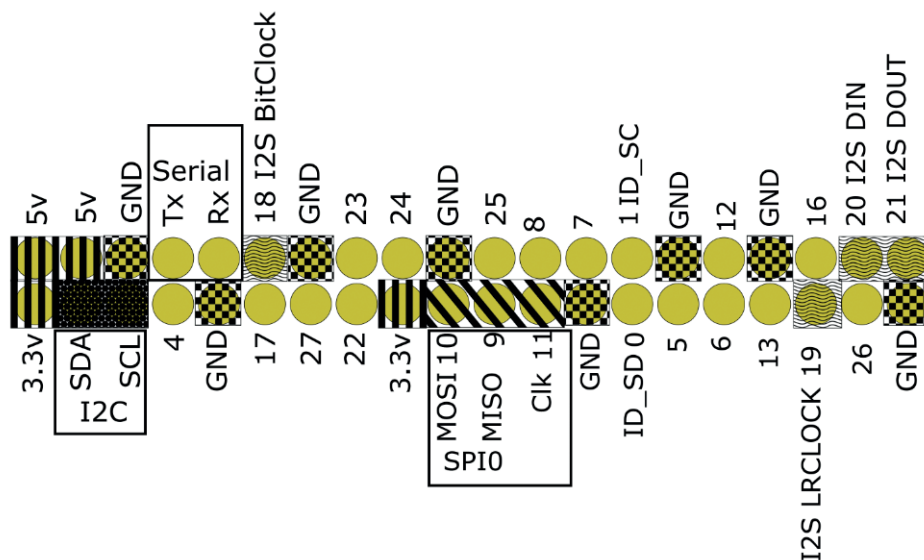


Рис. 3.2. Порт GPIO в Raspberry Pi (B+, 2, 3, 3B+, Zero и Zero W)

## Контакты для подключения питания

Для подключения питания используются контакты на 5 и 3,3В, а также контакты с маркировкой **GND**. GND (сокращение от GROUND – земля) – это вывод, соответствующий отрицательной клемме аккумулятора или источника питания. К контакту на 5В можно подключать аккумуляторы для питания микрокомпьютера. Контакты как на 3,3В, так и на 5В можно использовать для подачи питания на маломощные электронные компоненты или сенсоры.

<sup>3</sup> Этот разъем пригодится вам, если вы установите на Raspberry Pi полнофункциональную версию Linux с графическим интерфейсом. Если у вас нет монитора со входом HDMI, можете подключить Raspberry Pi к обычному телевизору. – Прим. ред.



## Шины передачи данных

Шины SPI, I2C и Serial используются для передачи управляющих сигналов и данных сенсоров между контроллером и интеллектуальными устройствами. I2S применяется для двусторонней передачи цифровых звуковых сигналов (**PCM-сигналов**). В зависимости от настроек эти выводы можно использовать или как порты шины передачи данных, или в качестве цифровых входов/выходов общего назначения.

Контакты, отмеченные на схеме как **SDA** и **SCL**, представляют собой линии шины данных I2C. Такие контакты применяются для подключения сенсоров и плат управления двигателями. Через данный порт отправляются управляющие сигналы.

Контакты **9, 10 и 11** относятся к порту **SPI**, который используется для управления RGB светодиодами.

В Raspberry Pi предусмотрен аудиопорт, но он не совсем подходит для подключения к нему динамика, поэтому вместо него мы будем использовать контакты I2S на порте GPIO. Контакты шины I2S представлены на схеме под номерами **18, 19, 20 и 21**. Они могут применяться для ввода аудиосигнала, ввиду чего мы будем использовать их для обработки голосовых сигналов.

## Входы/выходы общего назначения

Остальные контакты на схеме, обозначенные номерами, являются контактами ввода/вывода общего назначения. Они предназначены для цифрового ввода и вывода данных с серводвигателей, энкодеров и ультразвуковых сенсоров.

### Важное примечание

Номера контактов на схеме идут не по порядку. Это обусловлено тем, что в документации Raspberry Pi контакты нумеруются согласно кодам BCM, соответствующим номерам контактов на чипах от компании Broadcom (см. рис. 3.2).

## Платы расширения HAT для Raspberry Pi

*Платы расширения HAT<sup>4</sup>* для Raspberry Pi – это печатные платы, подключаемые к разъему GPIO и предназначенные для подключения к Raspberry Pi различных двигателей или сенсоров.

Одни платы имеют контакты GPIO, к которым можно подключать другие платы/соединения, а другие, напротив, требуют платы расширения, с помощью которых они могут подключиться к GPIO.

Платы HAT используют контакты GPIO по-разному. Например, аудио-HAT использует контакты I2S для подключения звука, а HAT контроллера двигателя использует контакты для управления двигателем. Помните, что при использовании нескольких HAT или определенных шин могут возникать проблемы. Подробнее об этом мы поговорим в главе 6 во время выбора контроллера двигателя.

<sup>4</sup> Hardware Attached on Top, оборудование поверх основной платы. – Прим. ред.



## ОПЕРАЦИОННАЯ СИСТЕМА RASPBERRY PI

Главным элементом системного программного обеспечения для управления нашим Raspberry Pi и исполнения кода послужит ОС Raspberry Pi – официальная операционная система от компании Raspberry Pi Foundation, включающая в себя множество предустановленных программ. Эта ОС может использоваться как полноценная операционная система, а также ее можно настроить для выполнения некоторых задач из командной строки. Помимо этого, Raspberry Pi OS может выступать в качестве сетевой ОС.

Raspberry Pi OS основана на дистрибутиве Linux **Debian**. Debian – это комплекс программ с открытым исходным кодом, обладающий широким функционалом. Подобные дистрибутивы Linux являются основной для многих серверных, мобильных и других ОС. Программное обеспечение упомянутой ОС оптимизировано для аппаратной платформы Raspberry Pi, а именно для специфических ядер и драйверов. Также система предоставляет возможность настройки специализированных функций.

Для нашего проекта лучше всего подойдет версия ОС **Lite**, поскольку робот будет несколько проще, чем настольный компьютер. Данная система представляет собой минимальную версию ОС Raspberry Pi. Она имеет низкие требования и занимает меньше места на карте microSD. Отказ от использования оконного режима (графического интерфейса) позволяет освободить память и задействовать меньше вычислительной мощности, ввиду чего Raspberry Pi может лучше справляться с такими задачами, как обработка видеоматериалов. Мы добавим в ОС Raspberry Pi Lite некоторые программы и инструменты, которые пригодятся при программировании будущего робота.

По большей части мы будем взаимодействовать с роботом с помощью кода и командной строки. Linux и ОС Raspberry Pi предусматривают использование командной строки по сети, благодаря чему они идеально подходят для реализации автономного управления робота.

Для нас важно, что Linux предоставляет поддержку языка программирования Python, а также многих сетевых инструментов. Операционная система Raspberry Pi имеет большую популярность в сообществе, и, по мнению многих пользователей, является одной из самых простых систем, обладающих хорошей поддержкой. Существуют и другие ОС для Pi, однако упомянутая система является наилучшим вариантом для тех, кто только начинает работать с Raspberry Pi.

## ЗАПИСЬ ОБРАЗА ОС RASPBERRY PI НА SD-КАРТУ

Чтобы запустить ОС Raspberry Pi на микрокомпьютере, необходимо записать образ системы на microSD-карту.

Программа Imager предназначена для упрощения этого процесса. Сейчас я предлагаю вам скачать эту программу и установить на карту правильный образ ОС.

1. Посетите страницу загрузки дополнительного ПО на официальном сайте Raspberry Pi по адресу <https://www.raspberrypi.com/software/>. Загрузите Imager для своей операционной системы как показано на рис. 3.3 (как правило, нужная вам версия выделяется в виде кнопки).

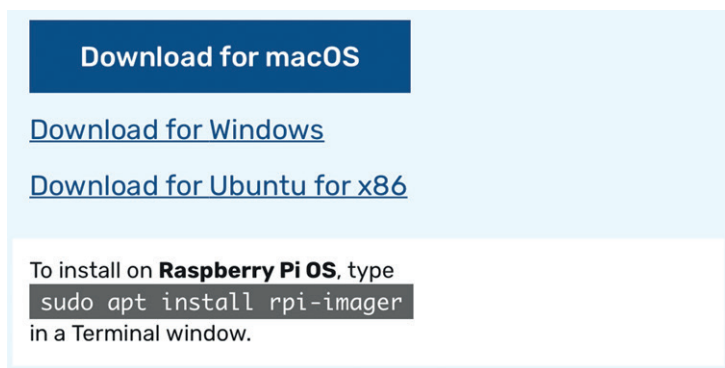


Рис. 3.3. Загрузка Imager для Raspberry Pi

2. Установите программу согласно инструкциям.
3. Вставьте microSD-карту в порт на ноутбуке. Вам может понадобиться переходник.
4. Запустите Imager и нажмите кнопку **CHOOSE OS** (выбор ОС), (см. рис. 3.4).

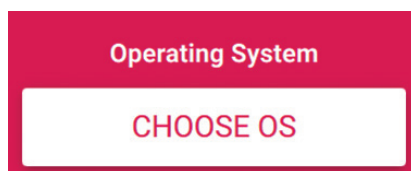


Рис. 3.4. Кнопка CHOOSE OS

5. Далее вы увидите список операционных систем, доступных для записи на карту. На рис. 3.5 показан список предлагаемых ОС. Выберите **Raspberry Pi OS (other)**.

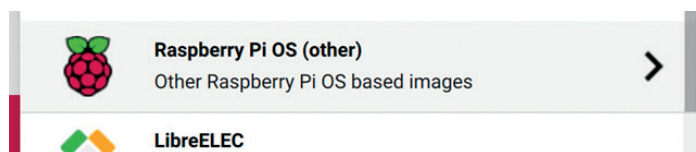


Рис. 3.5. Список доступных ОС

6. В открывшемся меню вы увидите доступные версии Raspberry Pi OS (см. рис. 3.6). Среди них нас интересует **Raspberry Pi OS Lite (32-bit)**.

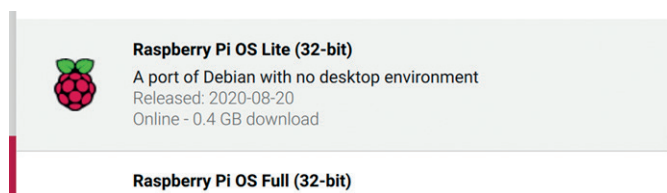


Рис. 3.6. Выбор версии Raspberry Pi OS

7. Далее нажмите кнопку **CHOOSE SD CARD** (выберите SD-карту) (см. рис. 3.7)

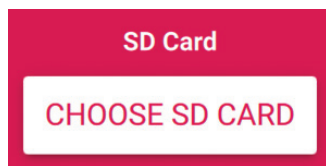


Рис. 3.7. Кнопка CHOOSE SD CARD

8. Далее появится список, который должен содержать используемую вами SD-карту (см. рис. 3.8). Выберите ее, чтобы продолжить.

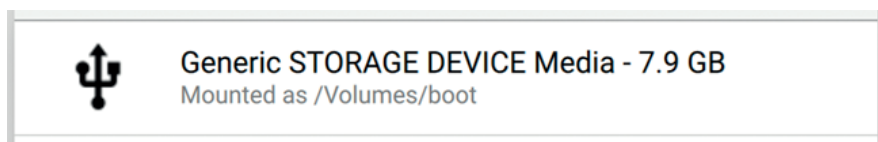


Рис. 3.8. Выбор SD-карты

9. На этом этапе необходимо записать образ. Нажмите кнопку **WRITE** (запись) (см. рис. 3.9).



Рис. 3.9. Кнопка WRITE

10. Если программа спросит, уверены ли вы, нажмите **YES** (да) чтобы продолжить. Дождитесь завершения загрузки и записи образа.

Вы можете загрузить эту программу на Raspberry Pi (при наличии монитора и клавиатуры), но, прежде чем вы сможете использовать ее, необходимо внести изменения в SD-карту на вашем компьютере. Этот вопрос мы обсудим в следующей главе.

## Выводы

В этой главе вы узнали больше о Raspberry Pi и подключении внешних устройств.

Вы узнали о том, что Raspberry Pi OS основана на дистрибутиве Linux. Затем мы рассмотрели, как записать образ этой системы на microSD-карту для использования в Raspberry Pi.

В следующей главе реализуем автономное использование карты. В результате для связи с Raspberry Pi вам больше не понадобится монитор, клавиатура или мышь.

## ЗАДАНИЕ

1. В нашей книге мы будем использовать только Raspberry Pi 3A+. Подумайте, в чем состоят плюсы и минусы других моделей? При этом учитывайте аспекты стоимости, размера, энергопотребления и скорости вычислений.
2. Попробуйте установить другую версию Raspberry Pi OS или другой дистрибутив (для этого может понадобиться клавиатура и мышь). Прежде чем продолжить чтение, убедитесь, что вы вернулись к версии Raspberry Pi OS Lite.
3. В этой главе мы говорили о разъемах камеры (CSI), питания и портах GPIO. Рассмотрите другие порты Raspberry Pi и подумайте, для чего они могут использоваться.

## ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ:

С дополнительными материалами вы можете ознакомиться по следующим ссылкам.

- Руководство Raspberry Pi Foundation по установке операционных систем Raspberry Pi: <https://www.raspberrypi.org/documentation/installation/installing-images/README.md>.
- Ознакомиться с альтернативными возможностями ОС для Raspberry Pi и множеством других интересных проектов можно в книге «*Raspberry Pi By Example*», Ашвин Паджнкар (Ashwin Pajankar) и Аруш Каккар (Arush Kakkar), Packt Publishing.
- На сайте <https://pinout.xyz/> описано расположение выводов GPIO для Raspberry Pi и подключение разных плат к контактам микрокомпьютера. Эта информация крайне полезна, поскольку подключение большинства плат осуществляется именно с использованием этой информации.

# Глава 4

## Автономное управление роботом с помощью Raspberry Pi

В этой главе вы узнаете, почему контроллер Raspberry Pi в роботизированной системе должен быть беспроводным и автономным. Затем мы рассмотрим, что представляет собой автономное управление и почему эта концепция так важна для робототехники. Вы реализуете автономное управление на Raspberry Pi, научитесь подключать микрокомпьютер к сети, а также отправите на него первые инструкции. К концу главы у вас будет готовый к использованию Raspberry Pi, которому для выполнения различных задач в системе не потребуется монитор, клавиатура и подключение к проводной сети.

В этой главе мы рассмотрим следующие темы:

- сущность автономных систем и их преимущества;
- настройку Wi-Fi и SSH на Raspberry Pi;
- поиск IP-адреса Raspberry Pi;
- соединение с Raspberry Pi посредством PuTTY и SSH;
- настройку конфигурации ОС Raspberry Pi.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам понадобятся:

- Raspberry Pi, предпочтительно модели 3A+ (также подойдут модели 3 и 4);
- блок питания с выходом USB на 2,1А и кабель Micro-USB;
- microSD-карта с образом ОС;
- компьютер на Windows, Linux или macOS с доступом к интернету, способный считывать/записывать информацию на SD-карту;
- текстовый редактор кода (например, отличным мультиплатформенным вариантом является VS Code);
- программа PuTTY, если вы используете компьютер на Windows (в десктопных версиях macOS и Linux предустановлен клиент SSH).

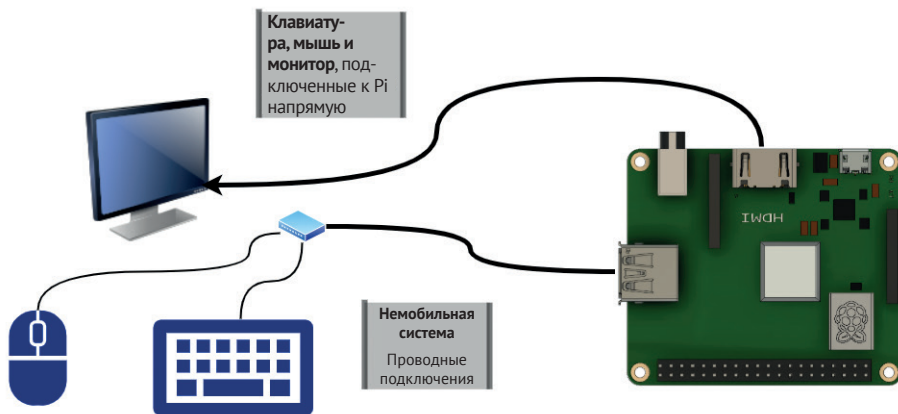
Найти код на GitHub вы можете по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter4>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/3bEr11I>.

## СУЩНОСТЬ АВТОНОМНЫХ СИСТЕМ И ИХ ПРЕИМУЩЕСТВА

**Автономные системы** предполагают возможность управления ими с помощью другого компьютера через сеть, когда управление с помощью монитора, компьютера или мыши невозможно. Очень часто такой тип управления реализуется для серверных вычислительных систем, а также при создании роботов и гаджетов.

На рис. 4.1 представлена система, для управления которой необходимо участие пользователя. Таким системам требуется подключение монитора, клавиатуры и мыши, ввиду чего они обладают низкой мобильностью. Разумеется, в случае необходимости все периферийные устройства можно отключить, но все же это не совсем удобно. Некоторые устройства, как на рис. 4.1, можно подключать к Raspberry Pi, но при движении робота все эти устройства либо должны будут передвигаться вместе с ним, либо их нужно будет отключить.



**Рис. 4.1.** Raspberry Pi с подключенными монитором, клавиатурой и мышью

На некоторых мероприятиях я видел роботов, оснащенных крошечными бортовыми экранами и управляемых с помощью беспроводных клавиатуры и мыши. Мы в свою очередь откажемся от этой идеи, и конструкция нашего робота не будет предусматривать прямого участия человека в управлении.

На рис. 4.2 представлен робот с автономной системой управления на основе Raspberry Pi. К нему не подключены монитор и клавиатура, поскольку их заменяет другой компьютер. Код, инструкции и другая информация отправляются на Raspberry Pi с ноутбука по беспроводной сети. Исполнение многих примеров кода может осуществляться автономно, т. е. компьютер может запускать/останавливать этот процесс самостоятельно. В главе 9 мы добавим нашему роботу светодиодные индикаторы. В главе 17 научимся управлять роботом с мобильного телефона, с помощью которого можно будет запускать и останавливать автономный режим, получать информацию о состоянии робота и управлять

им без необходимости подключения ноутбука. Такой Raspberry Pi будет полностью свободен от монитора и клавиатуры.



Рис. 4.2. Raspberry Pi в роботизированной автономной системе

### Совет

Несмотря на то что монитор и клавиатура не являются обязательными атрибутами, они все же пригодятся, если вы потеряете доступ к Raspberry Pi и не сможете связаться с ним через сеть. В такой ситуации можно подключиться к нему с помощью монитора и клавиатуры и выяснить причину проблемы.

Для реализации автономного управления мы будем использовать протокол **Secure Shell (SSH)**. SSH-клиент предоставляет командную строку для отправки инструкций на Pi, а также систему передачи файлов.

Благодаря автономности робот становится меньше и мобильнее, поскольку ему больше не требуются громоздкие монитор и клавиатура. Кроме того, такой подход побуждает вас, создателя, продумать аспекты автономного режима, поскольку вы не всегда сможете управлять роботом посредством ввода команд.

## НАСТРОЙКА Wi-Fi и SSH НА RASPBERRY PI

Теперь, когда вы узнали о преимуществах автономных систем, давайте скорректируем данные на нашей SD-карте таким образом, чтобы Raspberry Pi мог поддерживать такой режим. Для начала необходимо настроить Wi-Fi.

1. Извлеките и снова вставьте microSD-карту (используемую в предыдущих главах) в ваш компьютер, чтобы он распознал новый накопитель.
2. Далее вы увидите, что карта отображается как два диска. Один из дисков называется boot. Если Windows спросит, хотите ли вы отформатировать другой диск, нажмите **Cancel** (Отмена). В этой части SD-карты хранится файловая система для Linux, которую Windows не может прочитать.
3. Теперь создайте два файла в boot (на этом этапе я предлагаю использовать редактор VSCode, поскольку он хорошо подходит для редактирова-



ния простых текстовых файлов, просмотра расширений файлов и создания пустых файлов):

- ♦ ssh: пустой файл без расширения;
- ♦ wpa\_supplicant.conf: файл, содержащий конфигурацию вашей сети Wi-Fi:

```
country=GB
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="<your network ssid>"
    psk="<your network password>"
}
```

Давайте рассмотрим каждую строку этого файла.

Первая строка содержит код страны в формате ISO/IEC alpha2. Вы можете найти код вашей страны на <https://datahub.io/core/country-list>. Если ОС Raspberry Pi не обнаружит верный адрес адаптера Wi-Fi, он будет отключен. Я нахожусь в Великобритании, поэтому код моей страны – GB.

Следующие две строки позволяют другим инструментам обновлять конфигурацию.

Последние 4 строки файла определяют сеть Wi-Fi, к которой будет подключаться ваш робот и Raspberry Pi. Здесь вместо заполнителей необходимо вставить ваши сетевые данные. **Предварительный ключ** (PSK – Pre-Shared Key) – это пароль сети Wi-Fi, который вы используете при подключении к сети ноутбука или телефона. Я рекомендую сохранить копию файла wpa\_supplicant.conf на вашем компьютере для использования на других SD-картах.

### Важное примечание

Файл ssh не должен иметь расширения. Недопустимы варианты ssh.txt или подобные.

4. Извлеките microSD-карту. Не забудьте воспользоваться «безопасным извлечением». Это гарантирует, что запись файлов будет завершена до извлечения карты.
5. Теперь, когда два необходимых файла созданы, вы можете использовать microSD-карту для загрузки Raspberry Pi. Вставьте карту microSD в слот на нижней части Raspberry Pi. Убедитесь, что вставляете ее правильно.
6. Наконец, подключите Micro-USB-кабель к разъему на боковой стороне Raspberry Pi, а затем к источнику питания. При запуске индикаторы на корпусе микрокомпьютера начнут мигать.

### Важное примечание

Для Raspberry Pi вам понадобится блок питания мощностью не менее 2,1А.

На этом этапе завершается настройка системы автономного управления на Raspberry Pi. Мы делаем важный шаг на пути к мобильности. Теперь необходимо определить IP-адрес компьютера и установить соединение с ним.

## ПОИСК RASPBERRY PI В СЕТИ

Если введенные SSID и PSK верны, то Raspberry Pi будет зарегистрирован в сети Wi-Fi. Теперь необходимо найти адрес компьютера в сети. Raspberry Pi использует *динамические IP-адреса (DHCP, Dynamic Host Configuration Protocol – протокол динамической настройки узла)*. Они представляют собой уникальные значения, которые меняются при каждом подключении устройства к сети. Узнать IP-адрес вашего компьютера можно через интерфейс администратора маршрутизатора Wi-Fi. Однако в некоторых случаях это невозможно.

К счастью, Raspberry Pi использует технологию, известную как **mDNS (Multicast Domain Name System – многоадресная система доменных имен)**, что позволяет устройствам в локальной сети обнаружить его. Клиентский компьютер отправляет запрос на поиск устройства с именем `raspberrypi.local` в локальной сети, а Raspberry Pi отправляет свой адрес в ответ. Такая технология также известна под названиями Zeroconf и Bonjour. Первое, что вам необходимо сделать, – убедиться, что ваш компьютер поддерживает ее.

На компьютерах с macOS программа Bonjour, поддерживающая mDNS, установлена по умолчанию. Также она предустановлена на десктопных версиях Ubuntu и Fedora. На компьютере с Linux вам необходимо будет установить Zeroconf или Avahi, хотя в последних версиях они также установлены по умолчанию.

Если вы пользуетесь Windows, потребуется программа Bonjour. Давайте посмотрим, как ее установить.

## Установка Bonjour для Windows

Если вы недавно обновляли Skype или iTunes, скорее всего, программа уже установлена на вашем компьютере. Узнать, как проверить наличие программы на компьютере и запустить ее, можно в руководстве <https://smallbusiness.chron.com/enable-bonjour-65245.html>.

Также вы можете проверить наличие программы, введя в командную строку следующую команду:

```
ping raspberrypi.local
```

Если вы увидите это, значит, программа уже установлена на компьютере:

```
PING raspberrypi.local (192.168.0.53) 56(84) bytes of data.  
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=1 ttl=64  
time=0.113 ms  
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=2 ttl=64  
time=0.079 ms
```

Если вы увидите это, значит, программу необходимо установить:

```
Ping request could not find host raspberrypi.local. Please check the name and  
try again.
```

Для установки перейдите по адресу <https://support.apple.com/downloads/bonjour-for-windows> и загрузите Bonjour для Windows, а затем установите **службы печати Bonjour для Windows** (Download Bonjour Print Services for Windows). После запуска программы Windows сможет запрашивать mDNS устройства по имени.

## Тест системы

Когда зеленый индикатор на Raspberry Pi перестает мигать и остается гореть только красный индикатор питания, это означает, что микрокомпьютер завершил загрузку и установил соединение с сетью.

Если вы используете Windows, вызовите командную строку нажатием клавиши Windows и введите CMD в строке поиска. Если вы используете Linux или macOS, откройте приложение «Терминал» (Terminal). С его помощью можно проверить связь с Raspberry Pi, т. е. найти его IP-адрес, отправить короткое сообщение и получить ответ:

```
ping raspberrypi.local
```

Если все идет правильно, то компьютер покажет, что он установил соединение с Pi:

```
$ ping raspberrypi.local
PING raspberrypi.local (192.168.0.53) 56(84) bytes of data.
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=1 ttl=64
time=0.113 ms
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=2 ttl=64
time=0.079 ms
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=3 ttl=64
time=0.060 ms
64 bytes from 192.168.0.53 (192.168.0.53): icmp_seq=4 ttl=64
time=0.047 ms
```

Что делать, если вы не смогли установить соединение с Raspberry Pi? В следующем разделе мы рассмотрим устранение неполадок.

## Устранение неполадок

Если Raspberry Pi не отвечает на ping-запрос (запрос проверки связи), следует предпринять следующие шаги.

1. Еще раз проверьте подключения. Вы должны увидеть несколько вспышек зеленого светодиода, а красный должен гореть постоянно. В противном случае отключите питание, проверьте положение SD-карты и блок питания (он должен выдавать не менее 2,1A), а затем повторите попытку.
2. Зайдите в настройки точки доступа Wi-Fi и проверьте, является ли Raspberry Pi клиентом.
3. Если вы обнаружили Raspberry Pi в настройках Wi-Fi-роутера, это может означать, что mDNS на вашем компьютере работает неправильно. Если вы не установили данный протокол, то вернитесь назад и сделайте это. В Windows различные версии служб печати Bonjour, Bonjour для Skype и iTunes могут конфликтовать. На Windows в панели управления **Программы и компоненты** проверьте количество экземпляров Bonjour на вашем компьютере. Если их больше одного, удалите все и установите официальную версию Bonjour заново.
4. Отключите питание, извлеките SD-карту и вставьте ее в свой компьютер. Далее проверьте, присутствует ли файл `wpa_supplicant.conf`, а также убедитесь, что он содержит верные данные Wi-Fi и код региона. Наиболее частые ошибки в этом файле:

- a) неверные данные Wi-Fi (имя сети и пароль);
  - b) неверные знаки пунктуации или их отсутствие, а также отсутствие ка-вычек;
  - c) неверный код региона (или его отсутствие);
  - d) неверный регистр ключевых слов (ключевые слова должны быть введены строчными буквами, а код страны – прописными).
5. Файл SSH удаляется при запуске Raspberry Pi. Если вы уверены, что он был удален, это означает, что Pi действительно загрузился.
  6. Может потребоваться подключить к Pi монитор и клавиатуру и попытаться диагностировать проблему. С их помощью вы сможете обнаружить другие ошибки в `wpa_supplicant.conf` или иные неполадки. Попробуйте найти решение проблем в интернете. Ввиду разнообразия потенциальных проблем представить все их решения в этой книге невозможно. Можете задать вопрос в Twitter с тегом `#rasberrypi`, а также на Stack Overflow или на форуме Raspberry Pi по адресу <https://www.raspberrypi.org/forums/>.

После того как мы устранили неполадки с подключением Raspberry Pi к сети и нам удалось обнаружить его с помощью ping-запроса, можем подключиться к нему.

## ПОДКЛЮЧЕНИЕ К RASPBERRY PI ПОСРЕДСТВОМ PuTTY и SSH

При загрузке Raspberry Pi мы добавили файл с именем `ssh`. Он активирует службу SSH на Pi. Как упоминалось ранее, SSH – это протокол, обеспечивающий безопасный доступ к сети. Помимо надежного шифрования, он предоставляет возможность отправки команд и файлов на Raspberry Pi по удаленной сети без физического доступа к нему.

### Важное примечание

Если у вас уже установлен какой-либо SSH-клиент, обратите внимание, что не все SSH-клиенты командной строки для Windows поддерживают mDNS.

PuTTY – это удобный SSH-клиент, доступный для Windows, Linux и Mac. Информацию об установке клиента для различных операционных систем можно найти по адресу <https://www.ssh.com/ssh/putty/>.

После того как вы установили PuTTY, давайте установим соединение с вашим Raspberry Pi. Для этого необходимо выполнить следующие шаги.

1. Запустите PuTTY. Вы должны увидеть то, что изображено на рис. 4.3. В поле **Host Name** (Имя хоста) (или **IP address** (IP-адрес)) введите `raspberrypi.local`. Затем нажмите **Open** (Открыть), чтобы войти в свой Pi.
2. Если вы делаете это впервые, PuTTY выведет на экран предупреждение системы безопасности и попросит внести ключ Pi в список доверенных ключей. Нажмите **Yes** (Да). Такое предупреждение может появиться снова только в том случае, если другое устройство с тем же именем хоста (например, новый Raspberry Pi) будет использовать другой ключ.

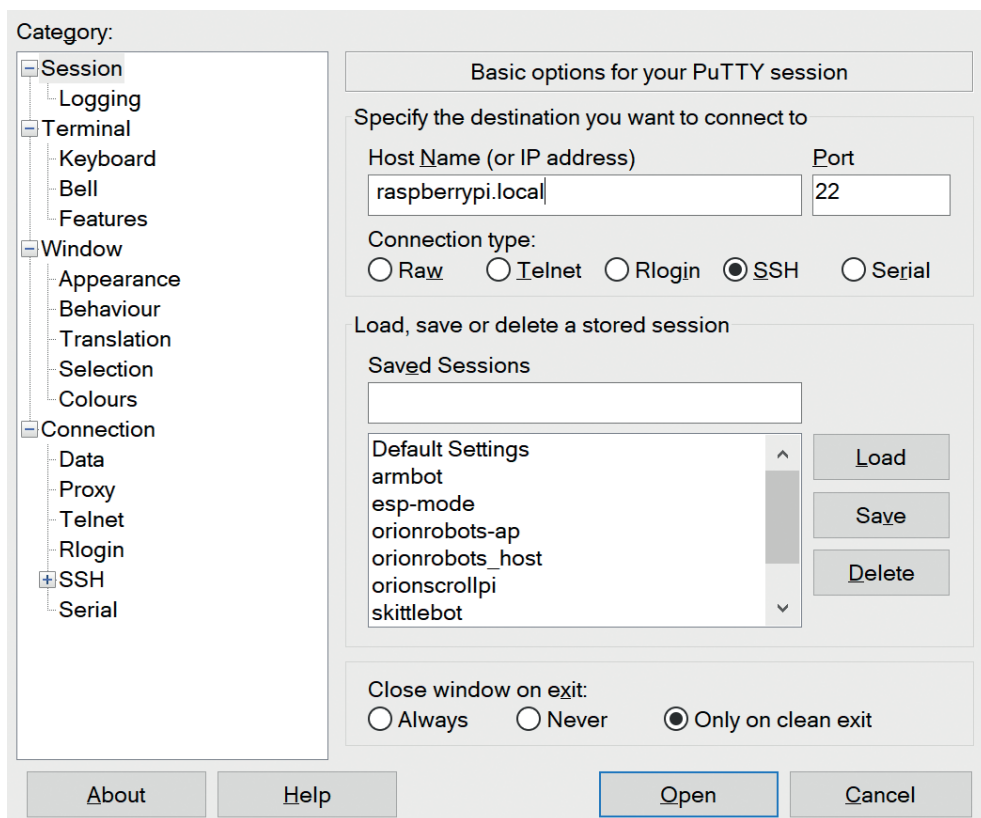


Рис. 4.3. Соединение с Pi

3. В поле **Login as** (Войти как) введите `pi`, нажмите **Ввод** и введите пароль `raspberrypi`. После этого вы увидите то, что изображено на рис. 4.4. Это означает, что вы установили соединение с Pi.

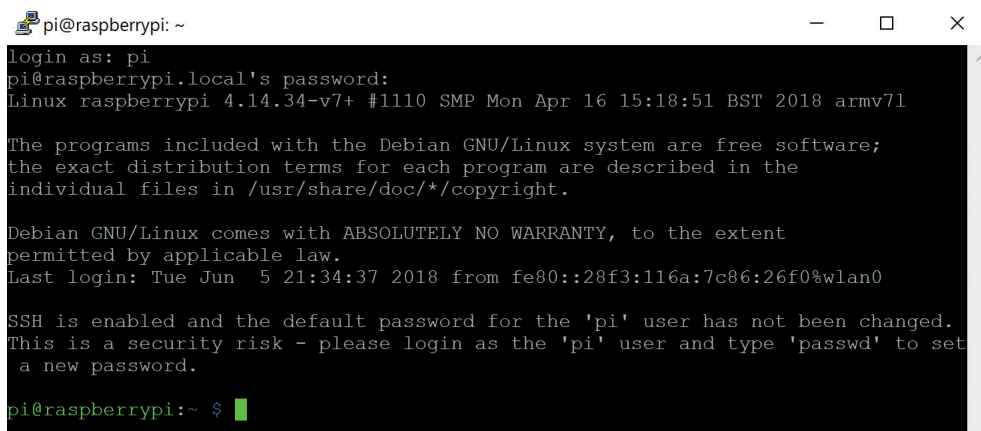


Рис. 4.4. Соединение установлено

Вы этом разделе мы подключились к Raspberry Pi с помощью SSH-клиента PuTTY (или другого предпочитаемого клиента). Теперь можно настраивать его конфигурацию, отправлять команды и взаимодействовать с ним. Далее мы рассмотрим настройку конфигурации.

## НАСТРОЙКА КОНФИГУРАЦИИ ОС RASPBERRY PI

Теперь, когда мы установили соединение с Raspberry Pi, необходимо подготовить его к дальнейшему использованию. В целях безопасности нужно изменить пароль и имя хоста.

Для большинства подобных задач мы можем использовать инструмент `raspi-config`, который представляет собой диалоговую систему управления с меню для настройки ОС Raspberry Pi. Для этого нам понадобится другой инструмент – `sudo`, который запускает `raspi-config` от имени главного пользователя (пользователя `root`). Для запуска необходимо ввести следующую команду:

```
sudo raspi-config
```

Далее появится интерфейс `raspi-config`, как на рис. 4.5.

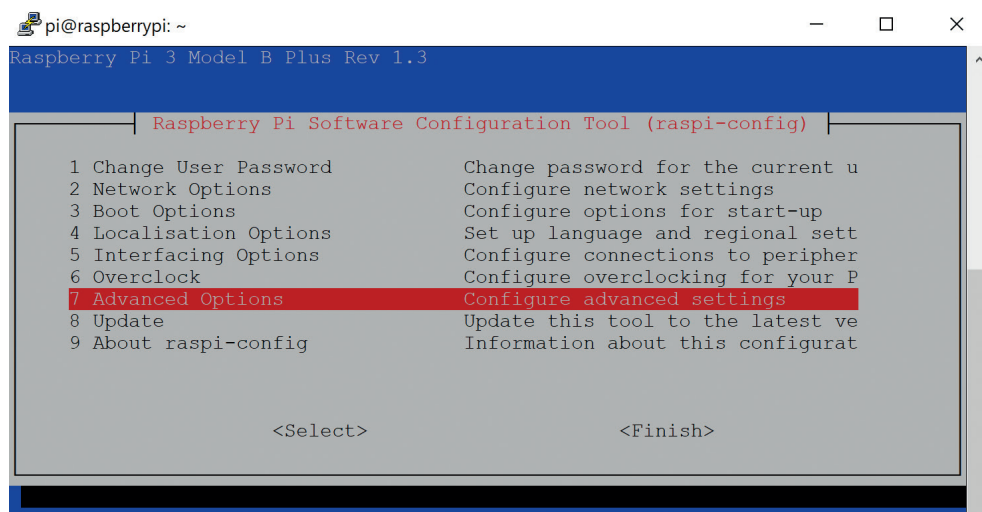


Рис. 4.5. Инструмент `raspi-config`

Теперь вы имеете доступ к `raspi-config` и можете использовать его для изменения настроек Raspberry Pi.

## Изменение имени Raspberry Pi

По умолчанию новый образ Raspberry Pi называется **raspberrypi**. Если в локальной сети будет несколько компьютеров с таким именем, вы не сможете найти свой. Для этого необходимо придумать новое имя. Я предлагаю использовать `myrobot`, но вы можете придумать что-нибудь получше. Можете изменить имя в любой момент. Оно может состоять только из букв, цифр и тире. Для того чтобы изменить имя выполните следующие шаги.

1. В `raspi-config` выберите **Network Options** (сетевые опции), как показано на рис. 4.6. Чтобы выбрать запись, выделите ее с помощью стрелок на клавиатуре и нажмите **Ввод**.

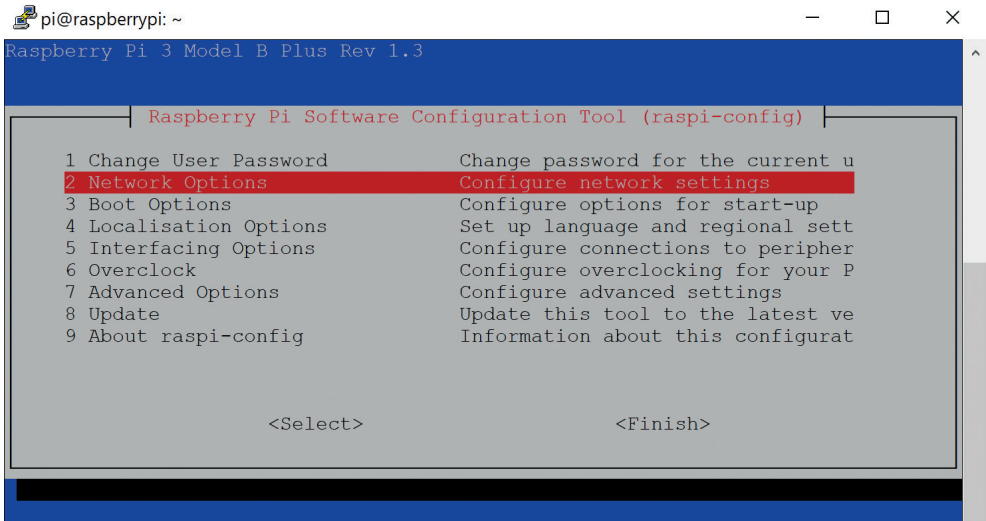


Рис. 4.6. Network Options (Сетевые опции)

2. Выберите **Hostname** (Имя хоста), как показано на рис. 4.7.

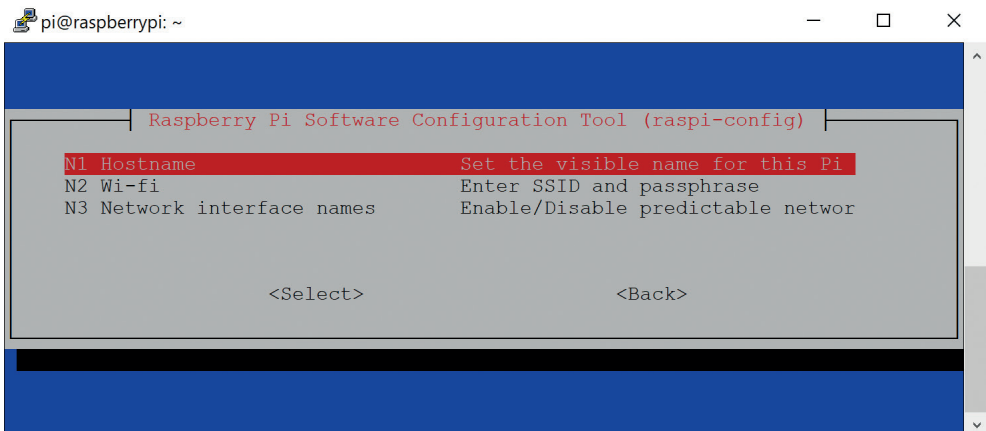


Рис. 4.7. Изменение имени хоста

3. Вы должны перейти на экран для ввода имени хоста. Введите имя вашего робота (своего я назвал `myrobot`) и нажмите **Ввод**. Я надеюсь, что вы придумаете имя поинтереснее.

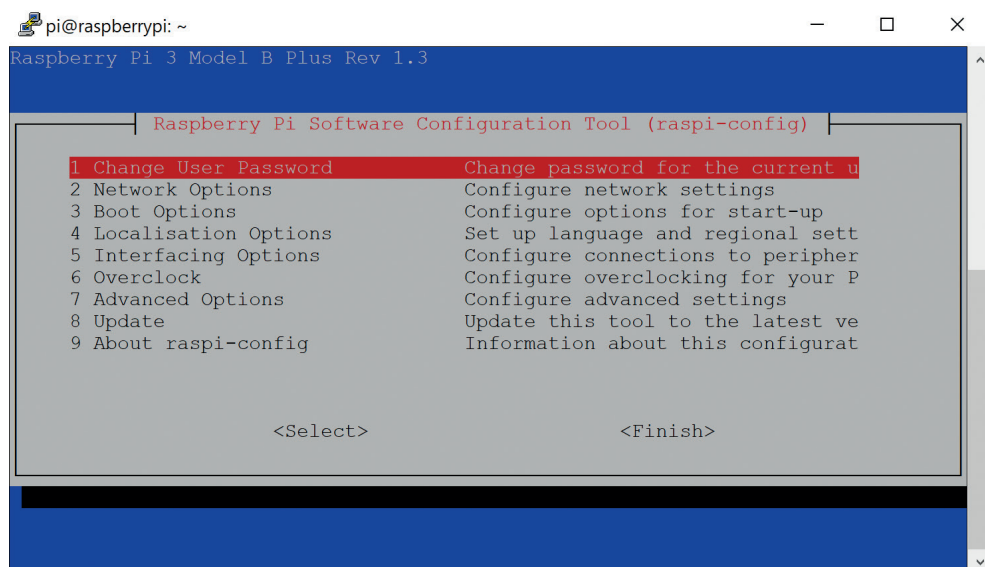
Вы задали роботу новое имя. Этот шаг особенно важен, если у вас есть другие Raspberry Pi. Также это позволяет персонализировать наш микрокомпьютер. Теперь давайте изменим пароль.



## Защита Raspberry Pi

На данном этапе у нашего Raspberry Pi установлен пароль по умолчанию, одинаковый для всех новых устройств. Для того чтобы изменить его, выполните следующие шаги.

1. В верхнем меню `raspi-config` выберите **Change User Password** (Изменить пароль пользователя) (см. рис. 4.8).



**Рис. 4.8.** Изменение пароля

2. Введите новый пароль. Он должен быть более сложным, чем `gaspberrу`, но вместе с тем запоминающимся. Не рекомендуется использовать пароль от электронной почты или банковских сервисов.

Изменение пароля позволяет персонализировать ваш Raspberry Pi и защитить его от злоумышленников. Для того чтобы внесенные в конфигурацию изменения вступили в силу, необходимо перезагрузить компьютер.

## Перезагрузка и повторное подключение

Пришло время завершить настройку и перезапустить Pi, для этого выполните следующее.

1. Нажатием клавиши **Tab** перейдите к элементу **Finish** (Завершить) (см. рис. 4.9), затем нажмите **Ввод**.

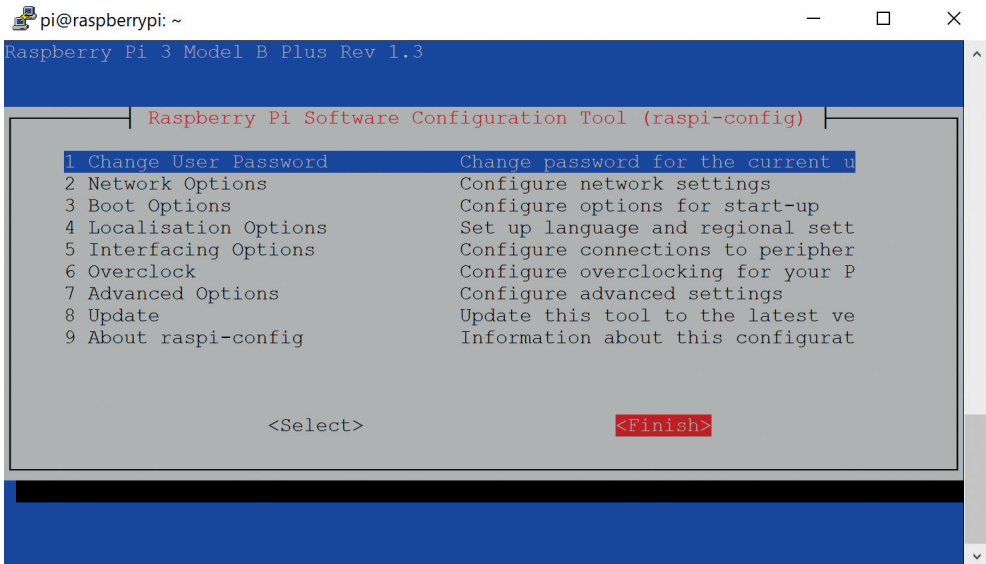


Рис. 4.9. Завершение настройки

2. Затем вы увидите всплывающее окно перезагрузки Pi. Выберите **Yes** (Да) и нажмите **Ввод** (см. рис. 4.10).

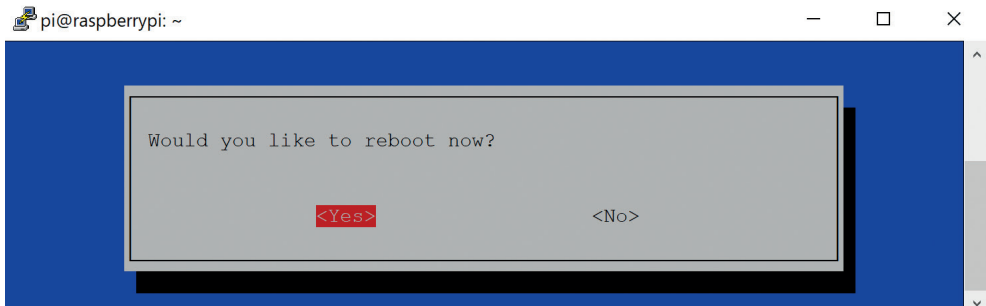


Рис. 4.10. Окно перезагрузки

После этого запустится процесс перезагрузки Raspberry Pi, а связь с PuTTY будет прервана (см. рис. 4.11). Подождите несколько минут. Зеленый индикатор на Pi должен начать мигать, а затем погаснуть. PuTTY сообщит вам, что соединение с компьютером прервано. Теперь Pi выключен. Красный индикатор будет продолжать гореть, пока вы не отключите питание.

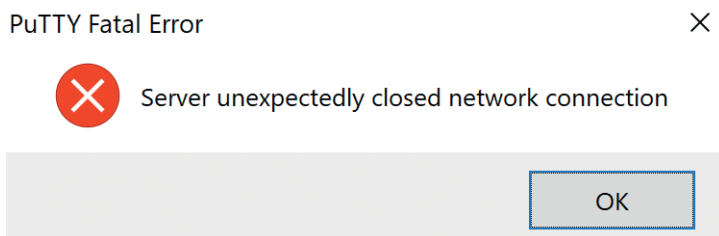


Рис. 4.11. PuTTY сообщает, что соединение с Pi прервано

PuTTY предназначен только для отправки команд роботу и получение ответов от него, поэтому он не понимает, что эта команда выключила Pi. Он не ожидает закрытия соединения, поэтому появляется окно ошибки. Для того чтобы закрыть его, нажмите **ОК**.

3. Снова подключитесь к Raspberry Pi с помощью PuTTY, используя новое имя хоста (в моем случае `myrobot`) с окончанием `.local` и пароль (см. рис. 4.12).

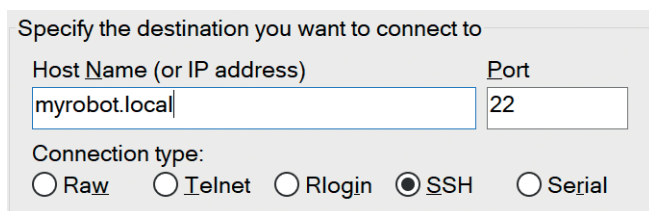


Рис. 4.12. Повторное соединение с Raspberry Pi

4. Теперь вы можете войти по приглашению **pi@myrobot** (здесь должно быть имя вашего Pi), как показано на рис. 4.13.

После того как мы снова подключились к Raspberry Pi, можно попробовать отправить простую команду Linux, чтобы посмотреть, сколько места на SD-карте мы использовали. Очень часто команды Linux представляют собой аббревиатуры.

Команда `df`, показанная на рис. 4.13, запускает анализ пространства накопителей, подключенных к Raspberry Pi. Если добавить к команде `-h`, результат будет преобразован в удобный для чтения численный формат. Суффиксы `G`, `M` и `K` означают гигабайт, мегабайт и килобайт соответственно. Введите команду `df -h`, как показано на рис. 4.13, и вы увидите, что большую часть объема памяти SD-карты занимает `/dev/root`, а остальное принадлежит другим устройствам.

```

pi@myrobot: ~
login as: pi
pi@myrobot.local's password:
Linux myrobot 4.14.34-v7+ #1110 SMP Mon Apr 16 15:18:51 BST 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jun 6 20:54:12 2018 from fe80::28f3:116a:7c86:26f0 wlan0
pi@myrobot:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        15G 1022M   13G   8% /
devtmpfs         460M    0   460M   0% /dev
tmpfs            464M    0   464M   0% /dev/shm
tmpfs            464M   12M   452M   3% /run
tmpfs            5.0M   4.0K   5.0M   1% /run/lock
tmpfs            464M    0   464M   0% /sys/fs/cgroup
/dev/mmcblk0p1   43M   22M   21M  51% /boot
tmpfs            93M    0    93M   0% /run/user/1000
pi@myrobot:~ $

```

Рис. 4.13. Повторное соединение с Raspberry Pi установлено

## Обновление программного обеспечения на Raspberry Pi

Теперь необходимо убедиться, что на вашем Raspberry Pi установлена последняя версия программного обеспечения. Этот процесс займет некоторое время. Введите команду `sudo apt update -y && sudo apt upgrade -y`. Вы должны увидеть нечто похожее на:

```

pi@myrobot:~ $ sudo apt update -y && sudo apt upgrade -y
Hit:1 http://raspbian.raspberrypi.org/raspbian buster InRelease
Hit:2 http://archive.raspberrypi.org/debian buster InRelease
.
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  bluez-firmware curl libcurl3 libcurl3-gnutls libprocps6
pi-bluetooth procs
  raspi-config wget
9 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,944 kB of archives.
After this operation, 16.4 kB of additional disk space will be
used. Get:1 http://archive.raspberrypi.org/debian buster/main
  armhf bluez-firmware all 1.2-3+rpt6 [126 kB]
.
.
.

```

Не выключайте компьютер до завершения процесса. По завершении вы увидите приглашение `pi@myrobot:~ $`.

**Совет**

В шаблонах некоторых команд, представленных в примерах, вы можете видеть директиву `sudo`. Она нужна для того, чтобы указать Raspberry Pi выполнить следующую команду (например, `raspbpi-config` или `apt`) от имени пользователя *root*, суперпользователя/администратора Linux. Чаще всего такого пользователя называют *суперпользователем* (*super-user*), а директива `sudo` образована от `superuser do` (выполнить от имени суперпользователя). Упомянутая директива требуется для программ, которые могут вносить изменения в систему или получать обновления. Для обычных пользовательских программ она, как правило, не требуется.

Обновление необходимо производить ежемесячно на протяжении всего периода работы над проектом. Обновив Raspberry Pi, вы сможете установить дополнительное программное обеспечение для кода робота. При устаревшем ПО вы не сможете воспользоваться инструментом `apt-get`. Эта проблема решается только обновлением.

## Завершение сеанса Raspberry Pi

В Raspberry Pi сеанс завершается посредством следующей команды:

```
pi@myrobot:~ $ sudo poweroff
```

Дождитесь, когда зеленый индикатор на корпусе погаснет. После этого PuTTY обнаружит потерю соединения с Raspberry Pi. Теперь вы можете безопасно отключить питание.

**Важное примечание**

Внезапное отключение питания может привести к потере файлов или повреждению SD-карты. Крайне важно проводить процедуру выключения Raspberry Pi правильно.

## Выводы

В этой главе вы узнали, как реализовать систему автономного управления в Raspberry Pi и избавиться от необходимости подключения монитора и клавиатуры. Теперь можете подключаться к компьютеру посредством Wi-Fi через SSH. Вы научились менять имя хоста и пароль с помощью инструмента `raspi-config` и разобрались с тем, как это работает в системе Linux. Также убедились, что на вашем Raspberry Pi установлено новейшее программное обеспечение. И наконец, вы узнали, как правильно завершать сеанс Pi, не повредив при этом файловую систему.

Сейчас вы уже знаете, как реализовать систему автономного управления в Raspberry Pi. Вы научились обновлять ПО и устанавливать соединение с сетью. Теперь на основе вашего Raspberry Pi можно создать какой-либо гаджет, в том числе робота.

В следующей главе рассмотрим проблему сохранения кода или настроек конфигурации при возникновении ошибок. Мы узнаем, какие ошибки могут происходить, а также рассмотрим, как использовать Git, *SFTP* (*SSH File Transfer Protocol* – *протокол передачи данных SFTP*) и SD-карты для создания резервных копий ваших проектов.

## Задание

1. Какие гаджеты или проекты могут быть созданы на основе автономного Raspberry Pi?
2. Попробуйте дать своему Raspberry Pi другое имя хоста и подключиться к нему в локальной сети с помощью PuTTY и mDNS.
3. Попробуйте другие команды Linux на Raspberry Pi, такие как `cd`, `ls`, `cat` и `map`.
4. По окончании работы завершите сеанс Raspberry Pi в соответствии с правилами.

## Дополнительные материалы

Дополнительную информацию вы можете найти здесь:

- «*Internet of Things with Raspberry Pi 3*», Маниш ПАО (Maneesh Rao), Packt Publishing: эта книга рассказывает о различных способах подключения Raspberry Pi и экспериментах с ним.

# Глава 5

## Создание резервных копий с помощью Git и SD-карты

Написание и модификация кода, благодаря которому ваш робот сможет делать удивительные вещи, займет много времени. Однако, если не соблюдать меры предосторожности, вся проделанная вами работа может просто исчезнуть. Речь идет не только о программах, ведь помимо этого вы уже начали настраивать конфигурацию ОС Raspberry Pi для будущего робота. Поэтому крайне важно иметь возможность сохранить код и конфигурацию в случае возникновения непредвиденных проблем, а также вернуться к предыдущей версии, если вы внесли нежелательные изменения.

В этой главе мы поговорим о том, какие ошибки при написании и изменении кода могут привести к потере данных, а также о других проблемах, с чем вы можете столкнуться. Затем рассмотрим три стратегии, которые обеспечат сохранность кода.

В этой главе мы рассмотрим следующие темы:

- причины возникновения проблем при написании и изменении кода;
- стратегию 1 – хранение кода на ПК и его последующую передачу;
- стратегию 2 – создание копий с помощью Git;
- стратегию 3 – резервное копирование данных на SD-карту.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам потребуется:

- Raspberry Pi и SD-карта, которую вы подготовили в предыдущей главе;
- блок питания с выходом USB и USB-кабель;
- компьютер на Windows, Linux или macOS с доступом к интернету, способный считывать/записывать информацию на SD-карту;
- программное обеспечение: FileZilla и Git;
- для Windows: Win32DiskImager.

Найти код на GitHub вы можете по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter5>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/3bAm94l>.



## ПРИЧИНЫ ВОЗНИКНОВЕНИЯ ПРОБЛЕМ ПРИ НАПИСАНИИ И ИЗМЕНЕНИИ КОДА

Написание кода и настройка конфигурации требуют много времени и сил. Для запуска кода требуется настройка конфигурации, например конфигурация ОС Raspberry Pi, дополнительное программное обеспечение и необходимые информационные файлы. Для разработки, реализации, тестирования и отладки всего вышеупомянутого требуется хорошая техническая подготовка.

Критические ошибки или другие обстоятельства могут привести к потере кода. Такое случилось и со мной всего за неделю до того, как нужно было отвезти роботов, над которыми я работал долгое время, на выставку. На собственном горьком опыте я научился относиться к этому достаточно серьезно. Итак, что же может случиться с вашим кодом?

### Повреждение SD-карты и потеря данных

При повреждении SD-карты вы можете потерять все данные, хранящиеся на ней, включая код, ОС Raspberry Pi и др. Файлы могут стать нечитаемыми, или же сама карта может стать непригодной для использования. Информация, хранящаяся на SD-карте, может быть потеряна безвозвратно.

Одной из причин повреждения SD-карты и потери данных является внешнее отключение питания Raspberry Pi. Также карта может повредиться при перегреве микрокомпьютера. Очень часто Raspberry Pi перегревается при обработке видеоданных. Еще одной причиной потери данных являются проблемы с электроникой, например повреждение контактов GPIO или источника питания. В конце концов, крошечную microSD можно просто потерять.

### Изменение кода или конфигурации

Людям свойственно ошибаться. При написании кода возникают ситуации, когда какая-то его часть перестает работать. При этом возникает необходимость просмотреть предыдущие версии кода и найти изменения, которые привели к ошибке. Для крайних случаев требуется возможность вернуться к последней рабочей версии кода.

Робот может стать бесполезным при неправильной настройке конфигурации, например Pi может перестать подключаться к сети или запускаться. Также может возникнуть ошибка при обновлении системных пакетов, что приведет к тому, что код не будет работать, и для его восстановления потребуются значительные изменения.

В совокупности эти проблемы могут привести к настоящему кошмару. Я видел, как ошибки в коде приводили к тому, что робот начинал функционировать неправильно, что влекло за собой неисправности в системе и повреждение SD-карты. Однажды во время обновления пакетов операционной системы я случайно отключил кабель питания, что вызвало повреждение SD-карты и данных конфигурации ОС Raspberry Pi. Ситуация произошла за две недели до события, для которого предназначался робот. Восстанавливать все потерянные данные было очень тяжело. Это был урок, который я запомнил на всю жизнь.

Настоятельно рекомендую вам создать резервные копии кода и конфигурации SD-карты. Далее в этой главе мы рассмотрим стратегии, позволяющие защитить ваше ПО при возникновении непредвиденных проблем.

## СТРАТЕГИЯ 1 – ХРАНЕНИЕ КОДА НА ПК И ЕГО ПОСЛЕДУЮЩАЯ ПЕРЕДАЧА

Протокол **SFTP** (*SSH File Transfer Protocol – протокол безопасной передачи файлов по SSH*) позволяет передавать файлы с компьютера на Pi. Вы сможете написать код на компьютере, а затем загрузить его в Raspberry Pi. В качестве расширения можно выбрать удобный редактор. Протокол позволяет сохранять несколько копий файлов.

### Важное примечание

Что такое редактор? Это специальное ПО для редактирования кода. Для кода на Python подойдут такие программы, как Mu, Microsoft VS Code, Notepad++ и PyCharm.

SFTP позволяет копировать и передавать файлы из Raspberry Pi (и обратно) по надежному и безопасному SSH-соединению. Рассмотрим, как это реализовать.

1. Создайте на своем ПК папку для хранения файлов с кодом для вашего робота (например, `my_robot_project`).
2. Внутри этой папки с помощью выбранного вами редактора создайте текстовый файл с одной строкой кода внутри. Назовем этот файл `hello.py`:  

```
print("Raspberry Pi is alive")
```
3. Скопируйте этот файл в систему робота и запустите его. Вы можете сделать копию с помощью SFTP-инструмента FileZilla. Для этого скачайте его с <https://filezilla-project.org> и установите согласно инструкциям.
4. Подключите Raspberry Pi и запустите его. На панели в правом нижнем углу экрана FileZilla (рис. 5.1) вы увидите надпись **Not connected** (Соединение не установлено).
5. В поле **Host** (Хост) введите имя локального хоста, которое вы дали своему Pi при настройке автономного управления, с префиксом `sftp://`, например `sftp://myrobot.local`.
6. В поле **Username** (Имя пользователя) введите `pi`, а в поле **Password** (Пароль) введите пароль, который вы установили ранее.
7. Нажмите кнопку **Quickconnect** (Быстрое соединение), чтобы установить соединение с Raspberry Pi.
8. После установки соединения на панели справа под названием **Remote site** (Удаленный сайт) (рис. 5.2) вы увидите файлы, хранящиеся на Raspberry Pi.

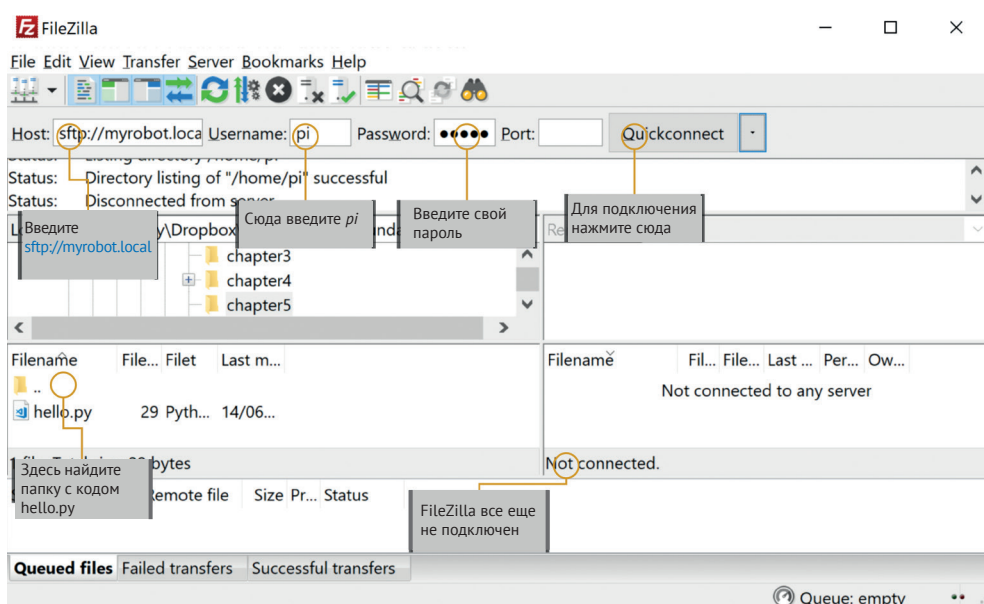


Рис. 5.1. FileZilla

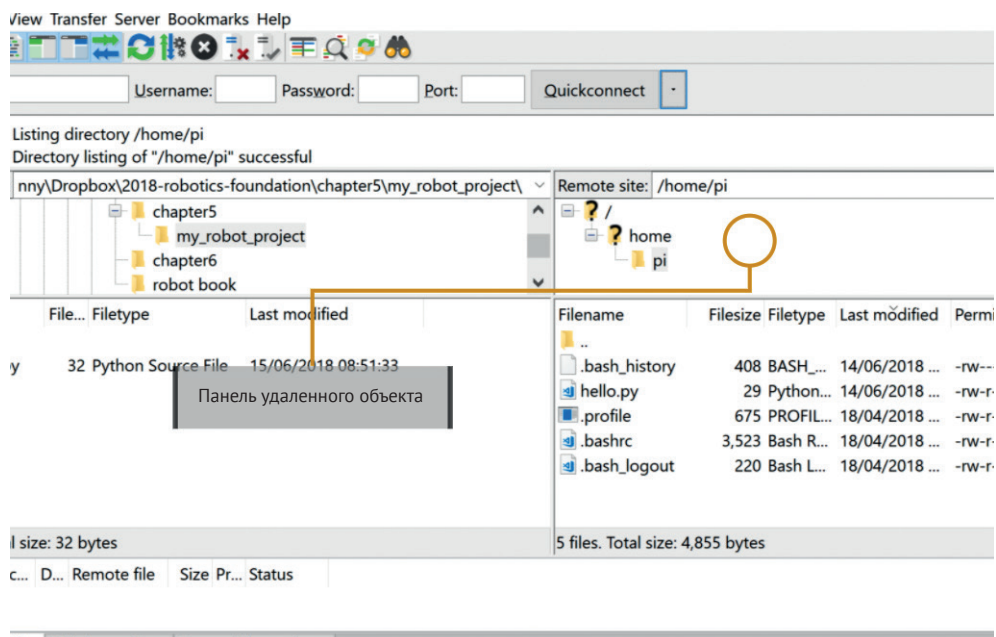


Рис. 5.2. Соединение с Raspberry Pi установлено

9. Перейдите на панель **Local site** (Локальный сайт), расположенную слева, где находятся файлы с кодом, хранящиеся на вашем компьютере.
10. Чтобы переместить файл на Raspberry Pi, нажмите на hello.py (в левом верхнем углу на рис. 5.3) и перетащите его в правую панель.

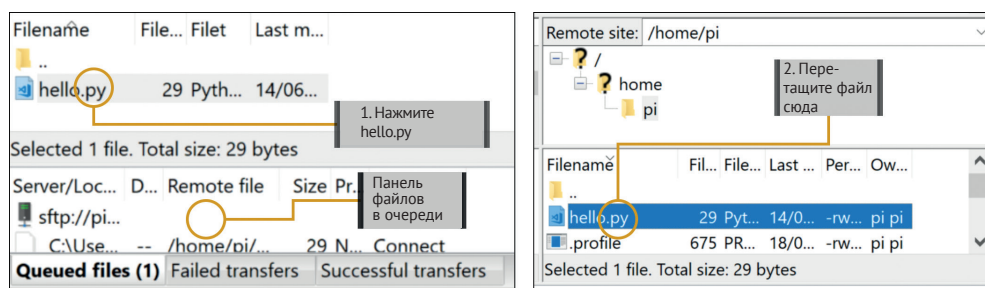


Рис. 5.3. Перемещение файла

11. Как показано на рис. 5.3, при перетаскивании файла он должен появиться в разделе **Queued files** (Файлы в очереди). Поскольку этот файл небольшой, он появится в этом поле лишь на мгновение. По такому принципу можно перемещать целые папки. Вскоре файл появится на панели **Remote site** (рис. 5.3).
12. Чтобы запустить этот код, войдите в Pi с помощью PuTTY и введите следующую команду:

```
pi@myrobot:~ $ python3 hello.py
Raspberry Pi is alive
```

Описанная стратегия хороша в качестве первого шага на пути к безопасности вашего кода. Она гарантирует вам наличие копий файлов как на вашем ноутбуке/ПК, так и на Raspberry Pi. Также у вас появилась возможность использовать любой редактор кода на ПК и обнаруживать ошибки еще до того, как неисправный код попадет на Raspberry Pi. Теперь, когда вы научились делать копии файлов, давайте рассмотрим, как мы можем отслеживать изменения в коде, и посмотрим, что конкретно мы изменили.

## СТРАТЕГИЯ 2 – СОЗДАНИЕ КОПИЙ С ПОМОЩЬЮ GIT

*Git* – популярная система управления версиями, которая позволяет хранить историю изменений, внесенных в код. Система позволяет возвращаться к внесенным изменениям, просматривать их, восстанавливать более ранние версии кода, а также вести журнал с комментариями, где можно указывать, почему вы внесли те или иные изменения. *Git* позволяет хранить код в нескольких местах на случай повреждения жесткого диска. *Git* хранит код и его историю в репозиториях. Система позволяет создавать *ветви* – полные копии кода, в которых вы можете экспериментировать с вашим кодом, а затем объединять их с основной ветвью.

В этом разделе мы рассмотрим лишь малую часть возможностей *Git*. Давайте начнем.

1. Установите *Git* на ваш компьютер согласно инструкциям на <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>.

**Совет**

Если вы используете Windows или macOS, я рекомендую воспользоваться приложением GitHub для упрощения настройки.

- Git потребует идентифицировать вашу личность посредством введения следующей команды в командной строке вашего компьютера:

```
> git config --global user.name "<Ваше имя>"
> git config --global user.email <Ваш email>
```

- Чтобы поместить проект в систему управления версиями, необходимо инициализировать его и выполнить операцию фиксации изменений первого бита кода. Убедитесь, что вы находитесь в папке, где хранится ваш код (`my_robot_project`), а затем введите в командной строке следующую команду:

```
> git init .
Initialized empty Git repository in C:/Users/danny/workspace/my_robot_
project/.git/
> git add hello.py
> git commit -m "Adding the starter code"
[master (root-commit) 11cc8dc] Adding the starter code
1 file changed, 1 insertion(+)
create mode 100644 hello.py
```

`git init` указывает, что нужно создать папку в репозитории Git. Команда `git add` сообщает, что вы хотите сохранить файл `hello.py` в Git. Команда `git commit` сохраняет это изменение, а `-m <message>` помещает сообщение в *историю коммитов*. Затем Git отвечает, что процесс успешно завершен.

- Посредством ввода команды `git log` мы можем просмотреть историю коммитов:

```
> git log
commit 11cc8dc0b880b1dd8302ddda8adf63591bf340fe (HEAD -
master)
Author: ваше имя <ваш@email.com>
Date: <сегодняшняя дата>
Adding the starter code
```

- Теперь вместо кода в файле `hello.py` введите следующее:

```
import socket
print('%s is alive!' % socket.gethostname())
```

Если вы скопируете файл с этим кодом на Pi с помощью SFTP, то получите ответ `mugobot is alive!` (или ваше имя хоста). Однако нас интересует поведение Git. Стоит отметить, что на более продвинутом уровне Git также позволяет передавать код на Raspberry Pi, но это выходит за рамки этой главы. Давайте посмотрим, чем текущий код отличается от предыдущего:

```
> git diff hello.py
```

```
diff --git a/hello.py b/hello.py
index 3eab0d8..fa3db7c 100644
--- a/hello.py
+++ b/hello.py
@@ -1,2 @@
-print("Raspberry Pi is alive")
+import socket
+print('%s is alive!' % socket.gethostname() )
```

Представленный выше код позволяет просмотреть различия в Git. Git интерпретирует изменения как удаление строки `print` и добавление вместо нее строки `import`, а затем `print`. Мы можем добавить эти изменения в Git и создать новую версию кода, а затем с помощью команды `git log` снова посмотреть обе версии:

```
> git add hello.py
> git commit -m "Show the robot hostname"
[master 912f4de] Show the robot hostname
1 file changed, 2 insertions(+), 1 deletion(-)
> git log
commit 912f4de3fa866ecc9d2141e855333514d9468151 (HEAD
-> master)
Author: ваше имя <ваш@email.com>
Date: <время следующего коммита>
Show the robot hostname
commit 11cc8dc0b880b1dd8302ddda8adf63591bf340fe (HEAD ->
master)
Author: ваше имя <ваш@email.com>
Date: <сегодняшняя дата>
Adding the starter code
```

Посредством описанного метода вы можете вернуться к предыдущим версиям кода или просто сравнить версии и получить возможность защитить код от внесения нежелательных изменений. Однако это далеко не весь функционал Git. В разделе «Дополнительные материалы» указан источник, где рассказывается о том, как создавать ветви, использовать удаленные службы, выполнять откат к предыдущим версиям, а также предоставляется информация об инструментах для просмотра кода в истории Git.

Теперь мы можем «перемещаться во времени», по крайней мере в отношении нашего кода, что позволяет нам свободно экспериментировать с ним. Просто не забывайте создавать коммиты – особенно после внесения удачных изменений. Далее мы рассмотрим, как сохранить конфигурацию и установленные пакеты.

## СТРАТЕГИЯ 3 – РЕЗЕРВНОЕ КОПИРОВАНИЕ ДАННЫХ SD-КАРТЫ

Git и SFTP отлично подходят для обеспечения сохранности кода, но они не помогут вам переустановить и перенастроить ОС Raspberry Pi на SD-карте. Эта процедура сильно различается в Windows, Linux и macOS. Однако все варианты основываются на одном алгоритме: вставить SD-карту, посредством специального инструмента клонировать карту и поместить все данные в *файл образа*, который в случае необходимости можно восстановить с помощью *balenaEtcher*.

**Важное примечание**

Восстанавливать образы можно только на карту с тем же объемом памяти, что и исходная (или больше). Записать образ на устройство с меньшим объемом памяти, скорее всего, не удастся. Это может привести к повреждению SD-карты.

Прежде чем мы начнем, правильно завершите сеанс Raspberry Pi, извлеките из него SD-карту и вставьте ее в свой компьютер. Чистые образы представляют собой файлы большого размера, поэтому не следует помещать их в git-репозитории. Это выходит за рамки книги, но я все же советую найти способ сжать их, поскольку на этом этапе они, как правило, пусты. Во всех случаях из-за размера образа эта операция может занять 20–30 мин.

**Windows**

Для Windows мы будем использовать программу Win32DiskImager. Для начала следует установить ее. Выполним следующие шаги.

1. Скачайте программу установки на <https://sourceforge.net/projects/win32diskimager/>.
2. Запустите программу установки и следуйте инструкциям.

**Совет**

Поскольку мы приступим к работе с программой сразу после установки, рекомендую не снимать флажок **Launch immediately** (Запустить по завершении установки).

3. Справа на рис. 5.4 вы можете видеть поле **Device** (Устройство), в котором автоматически должна появиться ваша SD-карта. Левее находится значок папки. Нажмите на него, чтобы выбрать, где будет храниться файл образа.

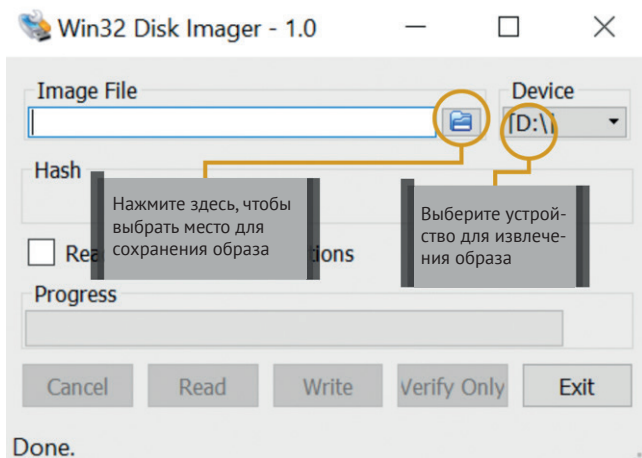


Рис. 5.4. Win32 Disk Imager



- В поле **File name** (Имя файла) необходимо ввести название образа (у меня myrobot.img), как показано на рис. 5.5. Затем нажмите **Open** (Открыть).

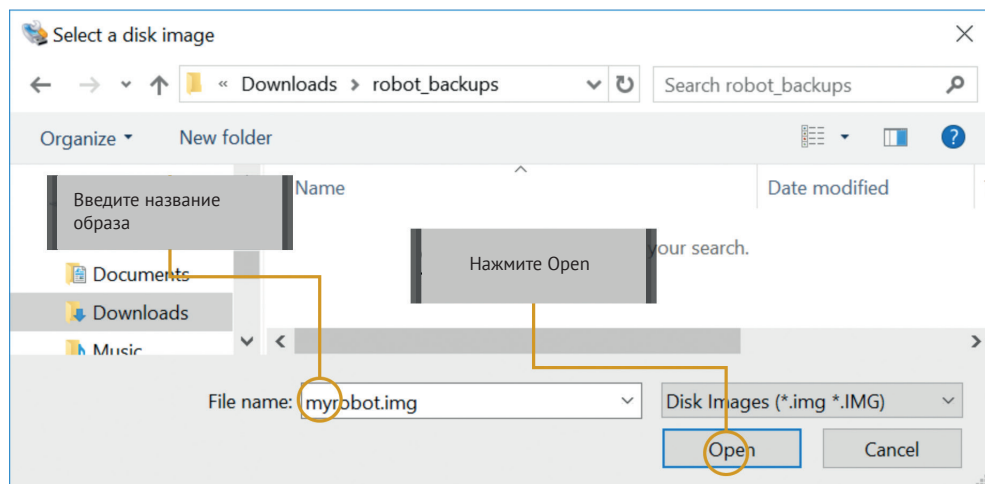


Рис. 5.5. Выбор местоположения файла

- После нажатия **Open** вы увидите, что в поле **Image File** (Файл образа) отображается выбранное местоположение (рис. 5.6 слева). Чтобы начать копирование образа, нажмите **Read** (Читать). Вы можете отслеживать процесс чтения образа по индикатору выполнения. Также программа показывает оставшееся время. Когда образ будет готов, Win32 Disk Imager сообщит вам, что чтение прошло успешно. После этого вы можете закрыть программу.

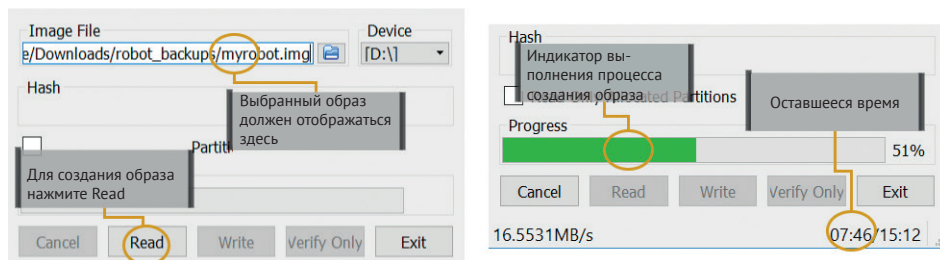


Рис. 5.6. Чтение образа

Теперь у вас есть полная копия данных, хранящихся на SD-карте. При повреждении карты или возникновении проблем с конфигурацией вы сможете записать этот образ на другую SD-карту и восстановить данные.

## Mac

На macOS X предустановлен инструмент для создания образов SD-карт и дисков – **Disk Utility** (Дисковая утилита). Давайте посмотрим, как он работает.

1. Запустите инструмент **Disk Utility**. После запуска вы увидите окно, показанное на рис. 5.7.

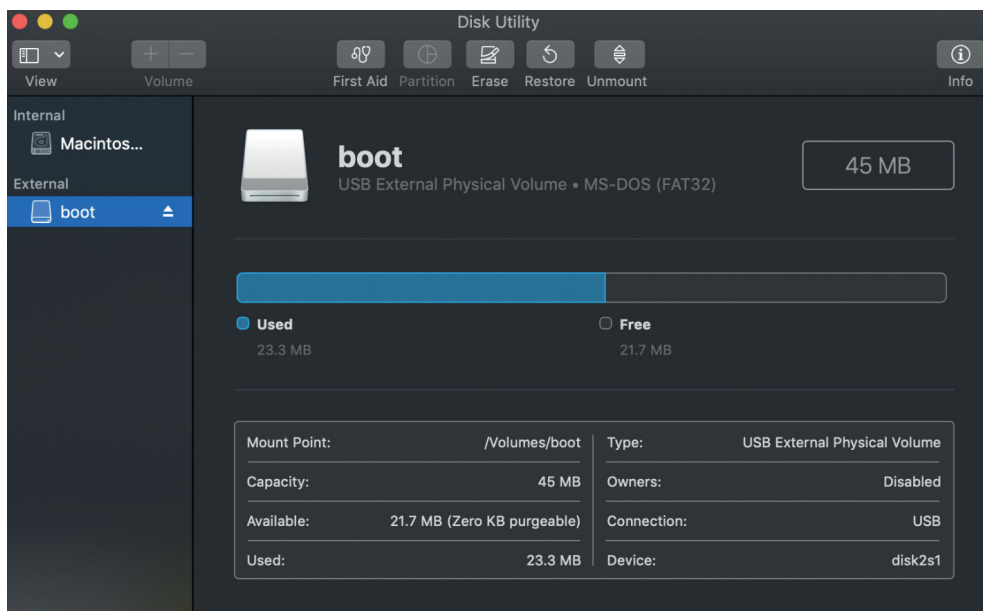


Рис. 5.7. Инструмент Disk Utility

2. Нажмите на **View** (Вид). Вы увидите панель меню, как на рис. 5.8.

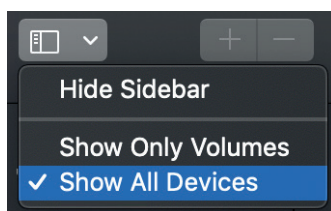


Рис. 5.8. Панель меню View

3. Выберите вариант **Show All Devices** (Показывать все устройства).
4. Вы должны увидеть экран, аналогичный показанному на рис. 5.9. Выберите устройство, содержащее загрузочный том.

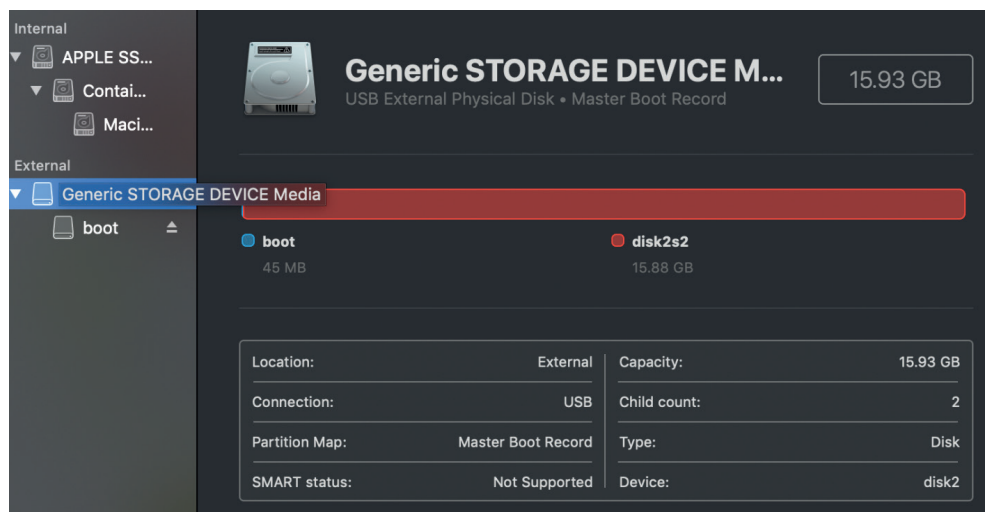


Рис. 5.9. Disk Utility с включенным параметром Show All Devices

- В панели меню выберите **File** (Файл), затем **New Image** (Новый образ) (рис. 5.10).

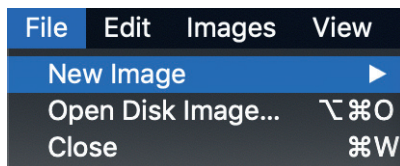


Рис. 5.10. Меню New Image

- В следующем открывшемся меню выберите **Image from <your storage device>** (Образ диска из <ваше устройство хранения>) (рис. 5.11).

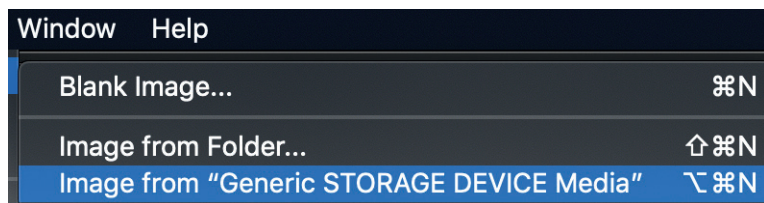


Рис. 5.11. Образ диска из STORAGE DEVICE

- Далее появится диалоговое окно (рис. 5.12). Задайте имя и местоположение файла, а в поле **Format** (Формат) выберите **DVD/CD master** (Мастер DVD/CD).

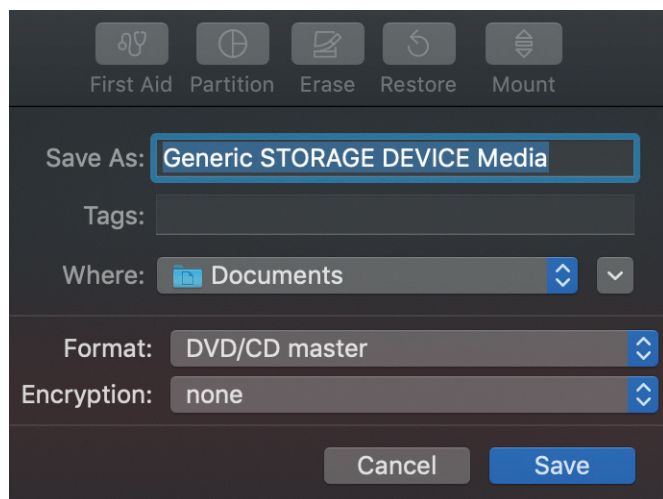


Рис. 5.12. Диалоговое окно сохранения

8. По умолчанию файлы в формате **DVD/CD master** (Мастер DVD/CD) имеют расширение **.cdr** (рис. 5.13).

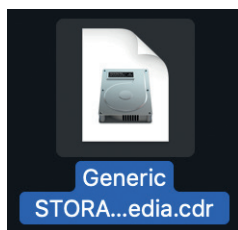


Рис. 5.13. Файл с расширением .cdr

9. Измените расширение файла на **.iso** (рис. 5.14).

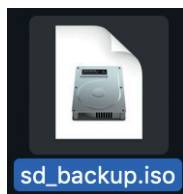


Рис. 5.14. Файл с расширением .iso

10. Подтвердите изменение расширения (рис. 5.15).

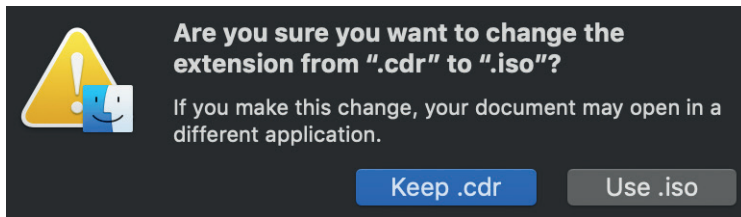


Рис. 5.15. Подтверждение изменения расширения

Теперь вы умеете создавать образы SD-карт, готовые для записи посредством balenaEtcher в macOS.

## Linux

В Linux резервное копирование SD-карт выполняется посредством ввода команды `dd` в командной строке. Прежде чем мы увидим, как это работает, сначала необходимо определить местоположение устройства. Давайте начнем.

1. Чтобы определить местоположение устройства, вставьте карту и введите следующую команду:

```
$ dmesg
```

2. Эта команда выведет много информации, но нас интересует только одна строка ближе к концу, которая выглядит так:

```
sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

Значение в квадратных скобках – `[sdb]` – это SD-карта. На вашем компьютере оно может отличаться. Местоположение SD-карты будет выглядеть следующим образом: `/dev/<местоположение накопителя>`, например `/dev/sdb`.

### Важное примечание

Убедитесь, что вы указали правильное расположение. В противном случае есть вероятность, что содержимое SD-карты или жесткого диска вашего компьютера будет уничтожено. Если вы **не уверены** в этом, **не используйте** описанный метод.

3. Определив расположение SD-карты (например, `/dev/sdb` или `/dev/disk1`), вы можете запустить клон с помощью команды `dd`. Она предназначена для копирования и чтения данных на дисках:

```
$ sudo dd if=/dev/sdb of=~/.myrobot.img bs=32M
Password:
474+2 records in
474+2 records out
15931539456 bytes (16 GB, 15 GiB) copied, 4132.13 s, 3.9
MB/s
```

Параметр `if` – это **входной файл**, которым в данном случае является ваша SD-карта. Параметр `of` – это **выходной файл**, файл `myrobot.img`, куда помещается клон вашей карты.

Параметр `bs` – это **размер блока данных**. Если увеличить его значение и установить его равным, например 32М, время выполнения команды сократится.

Для выполнения команды необходимо ввести пароль пользователя. Команда `dd` создаст файл `myrobot.img` – клон всей SD-карты из домашнего каталога. До завершения выполнения команды `dd` никакие сообщения на экран не выводятся. После выполнения команды вы можете просмотреть статистику операции.

## Выводы

В этой главе вы узнали, как следить за своим кодом и конфигурацией; увидели, какие проблемы могут возникнуть и как при этом сохранить свои данные. Вы создали резервные копии в Git, SFTP и копию SD-карты, благодаря чему появилась возможность свободно экспериментировать с кодом. С помощью SFTP вы можете редактировать код на своем компьютере, используя различные редакторы. Также это позволит хранить копию кода не только на Pi. С помощью Git можете вернуться к предыдущим версиям кода и исправить ошибки или же просто просмотреть внесенные изменения. Резервная копия SD-карты представляет собой полный образ хранилища вашего Raspberry Pi, и в случае повреждения карты все данные можно будет восстановить.

В следующей главе мы приступим к созданию простого робота. Мы соберем шасси робота с двигателями и колесами, определимся с необходимой системой питания, а затем протестируем конструкцию нашего робота. Приготовьте отвертку!

## Задание

- Попробуйте создать файл на своем компьютере – простой образ или текстовый файл. Загрузите его на Raspberry Pi посредством SFTP. Затем с помощью `PuTTY` посмотрите, сможете ли вы вывести содержимое файла с помощью команды `ls`. В файл можно поместить простой скрипт на Python, который можно попробовать запустить на Raspberry Pi.
- Внесите некорректное изменение в файл `hello.py`, а затем сравните версии файлов с помощью команды `diff`. Изучите возможности Git (см. раздел «Дополнительные материалы»), чтобы узнать, как вернуться к предыдущей версии.
- В соответствии с инструкциями создайте резервную копию SD-карты, которую вы используете для Raspberry Pi. Внесите какие-либо изменения в данные в `/home/pi`, а затем восстановите образ с помощью `balenaEtcher`. Также вы можете восстановить резервную копию на другой SD-карте и использовать ее в Pi.
- Дополнительно изучите функционал Git. Узнайте, как еще можно использовать Git для обеспечения сохранности кода или даже для его передачи на Raspberry Pi. В разделе «Дополнительные материалы» вы найдете источники, в которых подробно описываются возможности Git и его внедрение в процесс разработки кода. Git – довольно сложный инструмент, однако изучить его все же стоит.

## Дополнительные материалы

Дополнительную информацию вы можете найти здесь.

- Справочник Git (Git Handbook) на GitHub: <https://guides.github.com/introduction/git-handbook/>. Справочник представляет собой исчерпывающую информацию о том, что такое Git, какие проблем он решает, а также сведения о предоставляемых возможностях.
- Управление версиями с помощью Git на практике (Hands-On Version Control with Git): <https://www.packtpub.com/application-development/hands-version-control-git-video>. Видеоурок по использованию Git.
- Руководства GitHub (GitHub Guides): <https://guides.github.com/>. Серия руководств по максимально эффективному использованию Git и GitHub.
- Основы GitLab (GitLab Basics): <https://docs.gitlab.com/ee/gitlab-basics/>. GitLab – отличная альтернатива GitHub с большим сообществом и хорошими руководствами по использованию Git.



## Часть 2

# Создание автономного робота – подключение датчиков и двигателей к Raspberry Pi

В этой части мы создадим робота и заставим его перемещаться с помощью двигателей. Мы подключим датчики и двигатели к контроллеру, а также разработаем базовый автономный поведенческий сценарий.

Часть включает в себя следующие главы:

- главу 6. Сборка основания – колеса, питание и электропроводка;
- главу 7. Движение и повороты – код на Python для управления двигателями;
- главу 8. Код на Python для работы с датчиками расстояния;
- главу 9. Код на Python для работы со светодиодами;
- главу 10. Код на Python для управления сервоприводами;
- главу 11. Код на Python для работы с энкодерами;
- главу 12. Код на Python для работы с IMU.



# Глава 6

## Сборка основания – колеса, питание и электропроводка

В этой главе мы начнем собирать робота. Мы подберем набор для изготовления шасси с колесами и двигателями, а также контроллер двигателя и источника питания. Мы обсудим преимущества и недостатки и рассмотрим, чего стоит избегать новичку. Сначала подберем подходящие компоненты, а затем приступим к сборке робота. К концу главы у вас будет готова базовая конструкция робота.

Прежде чем приобрести компоненты, важно изучить их преимущества и недостатки, а также продумать расположение компонентов в конструкции. Так вы убедитесь, что все компоненты подходят, и сможете собрать рабочую конструкцию, над которой впоследствии можно будет экспериментировать.

В этой главе мы рассмотрим следующие темы:

- выбор набора шасси;
- выбор платы контроллера двигателя;
- питание робота;
- проверку размещения компонентов;
- сборку основания;
- подключение двигателей к Raspberry Pi.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам понадобится:

- компьютер с доступом в интернет;
- Raspberry Pi и SD-карта;
- набор отверток: M2,5, M3 Phillips, часовые отвертки;
- длинноносые плоскогубцы. По желанию – набор метрических гаечных ключей;
- электрические провода;
- текстильная застежка «липучка»;
- графический редактор, например [app.diagrams.net](http://app.diagrams.net), Inkscape, Visio или другой;
- набор нейлоновых стоек под резьбу M2,5 и M3;
- изоляционная лента;
- четыре заряженные батарейки AA.

**Важное примечание**

Не приобретайте шасси, контроллер двигателя и батарейный блок до того, как прочитаете эту главу.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/3oLofCg>

## ВЫБОР НАБОРА ШАССИ

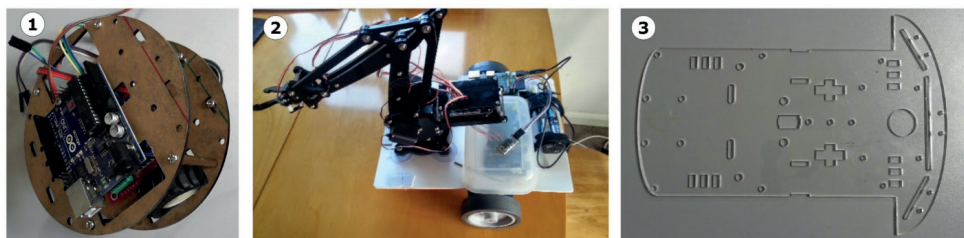
Наряду с выбором контроллера выбор шасси является фундаментальным решением при создании робота. Вы можете сделать шасси самостоятельно, например распечатать на 3D-принтере или разобрать какую-нибудь игрушку, но проще всего воспользоваться готовым набором для сборки. Такие наборы включают в себя компоненты, которые вам понадобятся на начальном этапе сборки. Конструкцию шасси можно изменить, но для этого робота нужно будет перестроить.

В интернете представлено множество наборов для сборки шасси. Как же выбрать среди них нужный?

## Размер

Немаловажную роль играет размер. Если вы сделаете робота слишком маленьким, для дальнейшей работы с ним потребуются навыки миниатюризации электроники. Если же робот будет слишком большим, ему потребуется более серьезная система питания. Новичкам лучше избегать таких вариантов.

На рис. 6.1 представлено сравнение шасси для роботов разных размеров.



**Рис. 6.1.** Сравнение размеров шасси

- Диаметр **шасси 1** составляет 11 см, чего недостаточно для размещения контроллера. Построить робота на таком маленьком шасси будет сложно. Для размещения контроллера, источников питания и сенсоров потребуются продвинутое навыки и опыт, которыми вряд ли обладает новичок.
- **Шасси 2** – это шасси довольно крупного робота Armbot. Размер шасси составляет 33 на 30 см, благодаря чему радиус действия манипулятора достигает 300 мм. Этому роботу требуется восемь батареек АА, большие двигатели и мощный контроллер двигателя, что значительно увеличивает расходы на его создание. Ему нужна серьезная система питания, он тяжелее

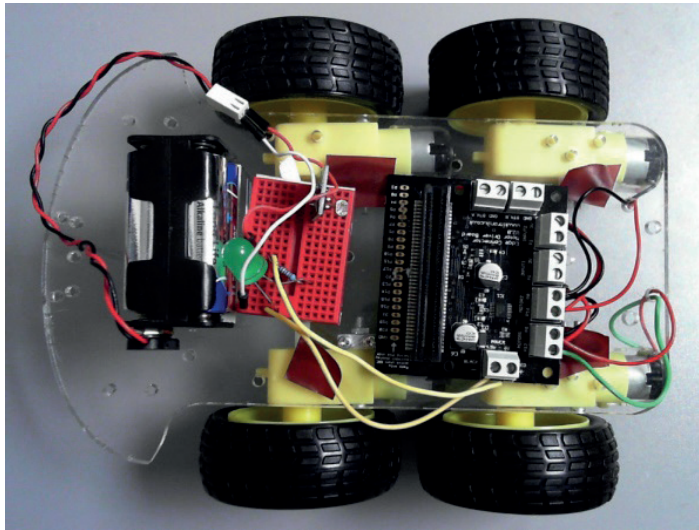
и обладает жесткой конструкцией, что может быть сложным для новичка. Armbot – один из моих самых дорогих роботов, не считая его манипулятора! На **шасси 3** можно разместить Pi, батареи и сенсоры, и при этом он не является громоздким. Размеры составляют 15–20 см в длину и 10–15 см в ширину. На нем можно размещать системы, состоящие не более чем из двух уровней, так как трех- или четырехуровневые системы сильно утяжелят робота и сделают его неустойчивым. На таком шасси можно разместить все необходимые компоненты, и при этом его относительно легко собрать.

Далее поговорим о количестве колес.

## Количество колес

Существуют разные наборы шасси, позволяющие строить роботов с довольно сложными способами передвижения, например с помощью ног, гусениц, колес Mecanum, тройных колес (tri-star wheels) и т. д. Экспериментировать с ними очень увлекательно, но новичкам я рекомендую выбирать более простые варианты. Для первого проекта лучше всего подойдут обычные колеса.

Существуют наборы с приводом на четыре колеса (см. рис. 6.2) и на шесть колес. Такие варианты являются более мощными, и для них необходимы контроллеры двигателя большего размера. У них более серьезные требования к питанию, ввиду чего может возникать риск перегрузки. Для дополнительных двигателей может потребоваться более сложная электропроводка.



**Рис. 6.2.** Робот с приводом на четыре колеса

Наиболее простым вариантом является привод на два колеса. Таким конструкциям требуется дополнительное колесо для поддержания баланса. В качестве третьего колеса можно использовать поворотное колесо (рис. 6.3) или роллербол (rollerball). Для крошечных роботов подойдут тефлоновые полозья. Двухколесная конструкция легче управляется, а также в ней не возникает про-

блем, связанных с трением, как это бывает у роботов с четырьмя или большим количеством колес.

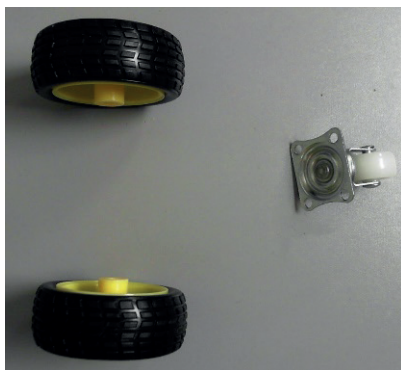


Рис. 6.3. Два ведущих колеса и поворотное колесо

Разумеется, два колеса не смогут обеспечить тяговое усилие как у четырех- или шестиколесного привода. Но для нас самое главное – простота и работоспособность. К тому же этот вариант недорогой.

## Колеса и двигатели

Набор для новичка поставляется с колесами (с простыми резиновыми шинами) и двигателями. На рис. 6.4 показано, как выглядят колеса из стандартного недорогого набора.



Рис. 6.4. Колеса из стандартного недорогого набора

В набор должны входить два двигателя (по одному на каждое колесо), а также винты или детали для крепления на шасси. Я рекомендую выбирать наборы с редукторными ДПТ, поскольку редуктор позволяет поддерживать необходимую скорость вращения, но при этом увеличивает механический крутящий момент приводного механизма робота.

Очень важно, чтобы к двигателям заранее были подсоединены провода, как на рис. 6.5 слева.

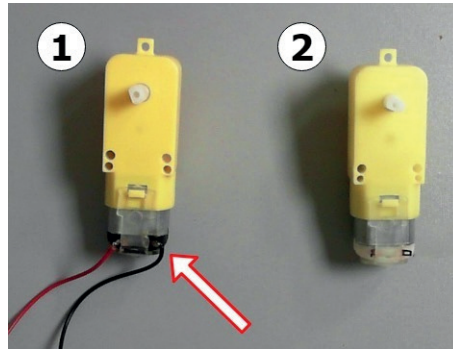


Рис. 6.5. Редукторный ДПТ с проводами и без

Провода сложно припаять или прикрепить к маленьким контактам на двигателях, а плохо подсоединенные провода будут часто отходить. В наборах для новичков все нужные провода к двигателям уже подсоединены, как показано на рис. 6.6.

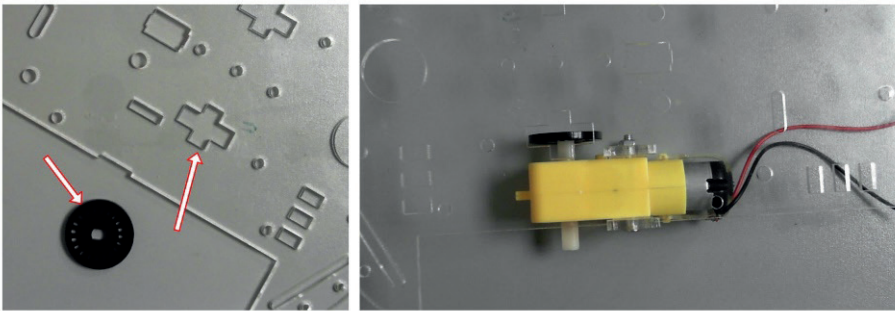


Рис. 6.6. Диск энкодера и прорезь для него крупным планом

В наборе вместе с двигателями должны поставляться диски энкодеров и щелевые оптопары для регистрации их вращения. Вместе диск и оптопара образуют оптический энкодер, который также называют одометром или тахометром.

## Простота

Для первого проекта лучше не использовать сложные в сборке наборы. Повторюсь, что нам необходим набор с приводом на два колеса и двумя двигателями с припаянными проводами. Я не рекомендую вам большие шасси или более сложные системы передвижения не потому, что они несовершенны, а потому, что лучше начать с простого. Однако и здесь есть предел. Готовый собранный набор, который поставляется в закрытом корпусе, не подойдет для обучения или экспериментов. Для работы с таким набором может потребоваться опыт в создании роботов из игрушек.



## Стоимость

Стоимость набора зависит от его сложности и варьируется от 15 долл. до нескольких тысяч. Затраты на создание более крупных и сложных роботов намного выше. В этой книге я стараюсь использовать недорогие варианты и показываю, где можно сэкономить.

## Заключение

Теперь вы знаете, что вам нужен набор шасси, включающий в себя два ведущих колеса, одно поворотное колесо, два двигателя с припаянными проводами, щелевую оптипару и диск энкодера. Прорези на таких шасси, как правило, выполняются лазером. Подобные наборы стоят не дорого. Приобрести их можно в популярных интернет-магазинах по запросу *Smart Car Chassis* с пометкой *2WD* (два ведущих колеса).

Я использую наборы (без Raspberry Pi), как на рис. 6.7.

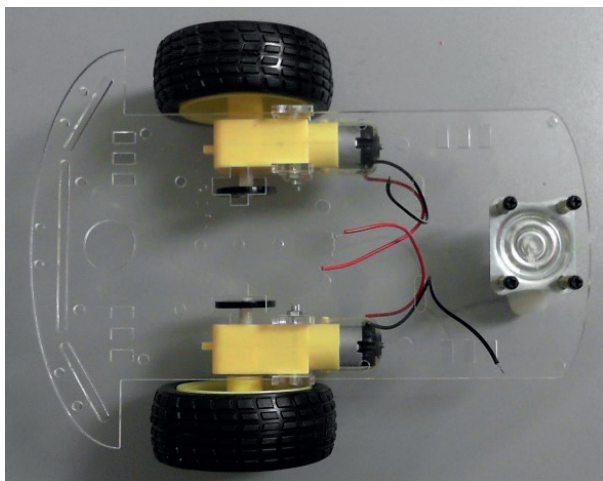


Рис. 6.7. Набор для сборки, который использую я

Итак, мы выбрали набор для сборки шасси. Наше шасси среднего размера, поэтому роботу будет достаточно небольшого источника питания. При этом на нашем шасси можно разместить все необходимые компоненты. Далее выберем подходящий контроллер.

## Выбор платы контроллера двигателя

Следующий важный компонент – это контроллер двигателя. Подключать двигатели напрямую к Raspberry Pi нельзя ввиду несовпадения рабочих напряжений, а большой потребляемый ток может повредить контакты GPIO. Платы контроллеров двигателей могут служить интерфейсом для других устройств, таких как сенсоры и другие типы двигателей.

Плата контроллера двигателя – это важнейший компонент робота, который определяет последующие решения. Как и при выборе двигателя, перед

покупкой необходимо ознакомиться с преимуществами и недостатками различных плат.

На рис. 6.8 показано несколько примеров плат контроллера двигателя, подходящих для нашего проекта.

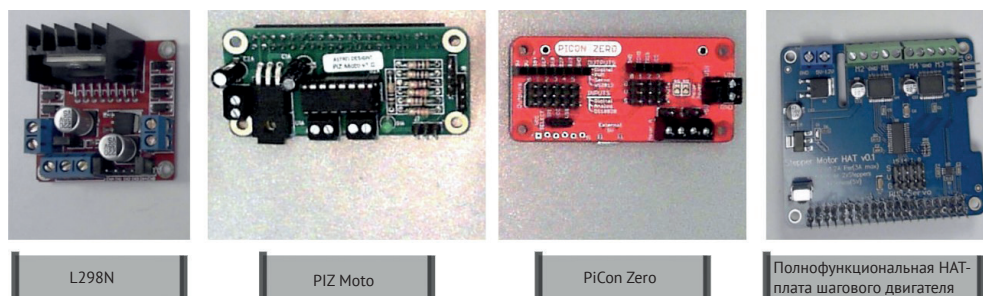


Рис. 6.8. Примеры плат контроллера двигателя

## Степень интеграции

Некоторые контроллеры, например L298N, управляют двигателем (или, как правило, двумя) посредством преобразования управляющих сигналов в токи для безопасного управления двигателем. Такие контроллеры не предназначены для установки непосредственно на Raspberry Pi и должны подключаться к контактам ввода/вывода.

Такие контроллеры, как PiZMoto, устанавливаются поверх платы Raspberry Pi, что позволяет использовать меньше проводов. В данном контроллере есть только одна цепь для управления двигателем. Также у него есть контакты для подключения дополнительных устройств, таких как датчики линии. Такие контакты выступают в качестве распределительных устройств, но об этом мы поговорим позже. Как L298N, так и PiZMoto нуждается в импульсах ШИМ, генерируемых микрокомпьютером.

Полнофункциональный HAT-контроллер шагового двигателя (Full Function Stepper HAT) имеет более высокую степень интеграции. У него есть микросхема, отвечающая за ШИМ-модуляцию, благодаря чему он может управлять не только ДПТ, но также и несколькими сервоприводами. Такая HAT-плата взаимодействует с Raspberry Pi посредством I2C, благодаря чему контакты ввода/вывода остаются свободными.

PiConZero обладает наибольшей степенью интеграции среди всех представленных плат. Она управляет ДПТ и сервоприводами с помощью микроконтроллера, подобного Arduino, способного управлять источниками света и получать выходные данные от различных сенсоров. Как и предыдущий вариант, PiConZero подключается к Pi посредством шины I2C.

## Использование контактов

Если вы выбираете контроллер двигателя в виде платы HAT, важно обратить внимание на то, какие контакты ввода/вывода она будет использовать. Недопустимо чтобы одни и те же контакты использовались для несовместимых функций.

**Важное примечание**

Несмотря на то что плата HAT подключается ко всем контактам, обычно она использует лишь несколько из них.

Чтобы узнать больше о расположении выводов GPIO и выбрать подходящую плату, посетите сайт <https://pinout.xyz>.

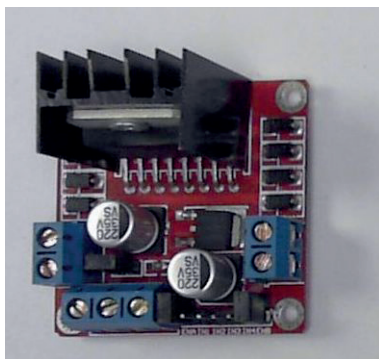
L298N потребуется четыре контакта ввода/вывода для ДПТ и дополнительные контакты для подключения сенсоров или сервоприводов. PiZMoto потребуется четыре контакта для ДПТ, а также один контакт для детектора линии, два контакта для датчика расстояния и два для светодиодов. В общей сложности PiZMoto задействует девять контактов GPIO. Для подключения серводвигателей потребуются дополнительные контакты.

PiConZero и полнофункциональный HAT-контроллер шагового двигателя подключаются к Pi через интерфейс I2C. Для подключения I2C или последовательной шины требуется всего два контакта, а остальные остаются свободными. Кроме того, шина может использоваться одновременно несколькими устройствами. PiConZero может потребоваться дополнительный контакт для подключения ультразвукового сенсора. В полнофункциональном HAT-контроллере шагового двигателя через один интерфейс I2C могут подключаться ДПТ и пять серводвигателей. Эта плата является одной из наиболее функциональных.

**Размер**

Выбор размера контроллера двигателя зависит от размера шасси и двигателей. Проще говоря, чем больше шасси, тем большего размера вам нужен контроллер. Предельно допустимая мощность контроллера двигателя измеряется в амперах (А). Для робота, подобного представленному на рис. 6.7, достаточно 1–1,5 А на один канал.

Слишком низкая мощность может привести к тому, что робот будет двигаться медленно или даже загорится. Большой контроллер потребует больше места для размещения и утяжелит конструкцию робота. Кроме того, большие контроллеры стоят дороже. Для охлаждения контроллера потребуется радиатор и, как показано на рис. 6.9, такое решение будет занимать намного больше места.



**Рис. 6.9.** L298N с радиатором

Размер также зависит от степени интеграции. Крошечная плата, которая устанавливается на Pi, занимает гораздо меньше места, чем отдельная плата.

Плата не должна закрывать доступ к порту камеры Raspberry Pi. Такие платы, как Pimoroni Explorer HAT Pro, закрывают его полностью. На некоторых платах есть специальная прорезь для порта камеры, например на плате полнофункционального HAT-контроллера шагового двигателя. Другие же платы, такие как PiConZero и PizMoto, относятся к платам половинного размера (pHat) и не закрывают порт камеры.

Наш робот будет оснащен камерой, поэтому следует выбирать либо плату с прорезью для доступа к порту, либо плату половинного размера.

## Пайка

При выборе контроллера стоит обратить внимание, что некоторые из них поставляются в виде наборов, где компоненты необходимо припаивать. Даже если у вас есть опыт в этом вопросе, пайка может потребовать много времени. Вместо этого лучше выбрать плату с уже припаянными линейными разъемами. Чтобы собрать плату, которая поставляется в виде набора, у вас уйдет не один вечер. Также может потребоваться последующая отладка.

### Совет

Пайка – важный навык для опытного робототехника, но для новичка это не обязательно. Именно поэтому при сборке первого робота я рекомендую выбирать модули, которые поставляются в собранном виде.

## Силовой источник питания

Двигатели могут работать от источника питания Raspberry Pi, который при этом должен обеспечивать напряжение не менее 5 В. Также двигатели могут потреблять **ток высокого напряжения**. Более подробно об источниках питания мы поговорим чуть позже, но стоит отметить, что для Raspberry Pi и двигателей лучше использовать разные источники питания. Все платы на рис. 6.8 имеют отдельные входы для питания двигателя (силового источника).

PiConZero и L298N позволяют питать Raspberry Pi от источника питания двигателя, но это может привести к сбросу и отключению микрокомпьютера. Некоторые платы интерфейса двигателя, такие как Adafruit Crickit и Pimoroni Explorer HAT Pro, используют один сильноточный источник питания на 5 В как для двигателей, так и для Raspberry Pi. Я рекомендую выбирать платы, на которых предусмотрен отдельный вход для питания двигателей.

## Способы подключения

Вопросы пайки и питания имеют прямое отношение к подключению двигателей и аккумуляторов. Я предпочитаю клеммы с винтовым зажимом, как на рис. 6.10.

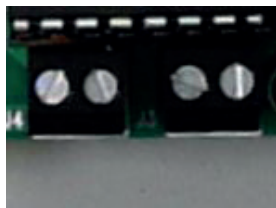


Рис. 6.10. Клемма с винтовым зажимом для подключения двигателя и аккумулятора

Другие типы клемм могут потребовать наличия у двигателей специальных контактов. Или же вам потребуются навыки использования инструмента для обжимных клемм.

## Заключение

В нашем роботе пространство для размещения компонентов ограничено. По этой причине больше всего нам подойдет контроллер типа HAT, который будет устанавливаться на Raspberry Pi. Также мы стремимся сократить количество используемых контактов, поэтому HAT, использующий интерфейс I2C, подойдет как нельзя лучше. На рис. 6.11 показан **полнофункциональный HAT-контроллер шагового двигателя (Full Function Stepper Motor HAT)**. Иногда их называют полнофункциональными платами расширения для роботов (Full Function Robot Expansion Board). Такая плата является мощным контроллером и при этом задействует лишь небольшое количество контактов Pi.

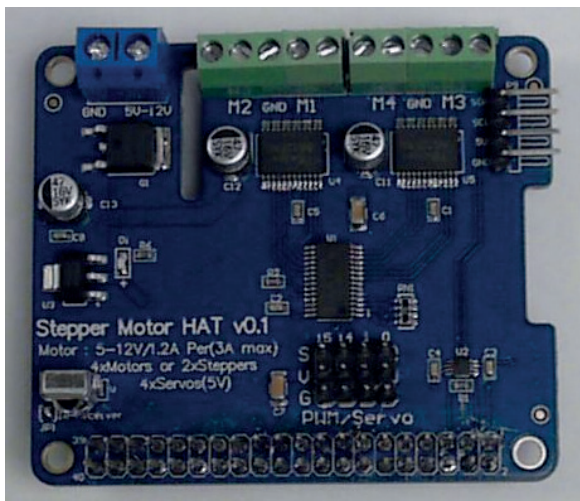


Рис. 6.11. Полнофункциональный HAT-контроллер шагового двигателя (Full Function Stepper Motor HAT)

Упомянутый контроллер можно приобрести в большинстве стран. Он имеет прорезь для доступа к порту камеры и может управлять сервоприводами. Для нашего робота я рекомендую именно этот контроллер. Наш код напрямую совместим с платами на базе микросхемы PCA9685, к которым относится этот



контроллер. Если внести некоторые изменения в код и схему подключения, подойдет и PiConZero от 4tronix.

## ПИТАНИЕ РОБОТА

Всем компонентам робота нужно питание. Рассмотрим две основные системы питания: питание цифровых компонентов, таких как Raspberry Pi и сенсоры, и питание двигателей.

Двигателям нужна отдельная система питания по ряду причин. Во-первых, они потребляют гораздо больше электроэнергии, чем большинство других компонентов робота. Во-вторых, цифровым компонентам и двигателям может требоваться разное напряжение. Я встречал как низковольтные, так и высоковольтные сильнотоочные источники питания. В-третьих, двигатели могут создавать помехи. Четвертая причина заключается в том, что они могут потреблять много энергии, что приведет к провалам напряжения в других цепях. Провалом напряжения является внезапное снижение напряжения в цепи. Продолжительный провал может привести к сбою или сбросу микрокомпьютера. Сброс может привести к повреждению SD-карты на Pi. Помимо вышесказанного, двигатели могут создавать электрические помехи в линии питания, способные привести к сбоям в работе цифровых компонентов.

Существуют две основные стратегии питания робота с двигателями:

- **две батареи:** при таком подходе двигатели и другие компоненты будут питаться от двух независимых комплектов батарей;
- **эмулятор батарей<sup>5</sup>:** при таком подходе используется *BEC (battery eliminator circuit – схема заменителя батареи, модуль вторичного питания)* / регулятор (рис. 6.12).

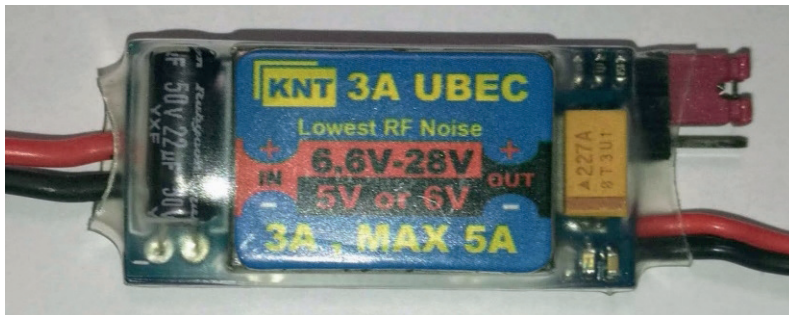


Рис. 6.12. Модуль вторичного питания, заменяющий отдельную батарею

Использование двух батарей – самый надежный способ избежать провалов напряжения, отключения питания или возникновения помех. Однако такая система занимает больше места, чем BEC. Наиболее эффективным подходом является

<sup>5</sup> В электрической схеме питания такой модуль играет роль отдельной виртуальной батареи, обеспечивая надежную развязку и защиту от помех между «грязными» силовыми цепями для питания двигателей и «чистыми» цепями питания цифровых модулей и одновременно является регулятором/стабилизатором напряжения. – Прим. ред.

вариант с двумя источниками питания, где Raspberry Pi питается от внешнего аккумулятора с выходом USB (рис. 6.13). Лучше всего подойдет небольшая по размеру модель с емкостью 10 000 мАч и выходной мощностью не менее 2,1 А.



**Рис. 6.13** Внешний аккумулятор с выходом USB для Armbot (старый и потрепанный, но все еще эффективный)

К внешнему аккумулятору должен прилагаться кабель для подключения к Raspberry Pi. Для Raspberry Pi 3 (A+ и B+) требуется разъем Micro-USB, а для Raspberry Pi 4 – USB-C. Практически все внешние аккумуляторы имеют выход USB-A. Если кабелей в комплекте нет, их нужно приобрести.

В некоторых платах контроллеров двигателей предусмотрены источники питания, которые перенаправляют часть энергии источника питания двигателей на питание Pi. Как правило, выходная мощность у таких источников питания слишком низкая. Они могут быть крайне неэффективными и расходовать много энергии батареи. Если вы не используете аккумуляторные батареи, способные отдавать очень высокий ток, могут возникать провалы напряжения.

Модуль вторичного источника питания легче и занимает меньше места. Существует несколько типов таких модулей: BEC, uBEC, импульсный источник питания и регулятор. Для вторичных источников требуются мощные батареи, например литий-ионные. Если от этой же батареи питаются двигатели, то она должна отдавать достаточно большой ток, чтобы не возникали кратковременные провалы напряжения, способные вызывать сброс контроллера и помехи по цепям питания. Данный аспект важен для импульсных блоков питания Pi и блоков питания, встроенных в контроллер двигателей.

Выход BEC должен выдерживать потребляемый ток не менее 2,1 А (лучше больше). Часто встречаются аккумуляторы на 3,4 и 4,2 А. Также распространены UBEC с выходным током 5 А.

Чтобы упростить работу с роботом и снизить риск самопроизвольного сброса, мы будем использовать стратегию с двумя независимыми батареями, несмотря на то, что это несколько утяжелит конструкцию нашего робота.

Двигателям необходимы четыре батарейки типоразмера AA. Я рекомендую использовать никель-металлогидридные аккумуляторные батареи. Помимо того, что их можно заряжать повторно, они выдают больший ток, чем щелочные батареи. В целях экономии места мы используем конфигурацию «две сверху/две снизу» или «спина к спине», как показано на рис. 6.14.





**Рис. 6.14.** Батарейный блок для двигателя под четыре батарейки AA

Подведем итог. Для двигателей мы используем комплект из четырех металлгидридных батарей AA, а для Raspberry Pi и цифровых компонентов – внешний аккумулятор с выходом USB.

Итак, мы выбрали необходимые компоненты и определили, какие конфигурации шасси, контроллера и питания будем использовать. В предыдущих главах мы выбрали подходящую модель Raspberry Pi, а в этой главе определились с размером двигателей и колес. Перед приобретением выбранных компонентов еще раз убедитесь, что они вам подходят. В следующем разделе мы рассмотрим компоненты конструкции.

## ПРОВЕРКА КОМПОНОВКИ

Прежде чем приобрести компоненты, я рекомендую примерить их друг к другу, чтобы убедиться, что они взаимно совместимы по габаритам. Помимо этого, вы получите примерное представление о том, где будут размещаться компоненты. Этот шаг сэкономит вам время и деньги в дальнейшем.

Примерка компонентов не вызовет затруднений – просто нарисуйте компоненты в натуральную величину на бумаге или в приложении `diagrams.net`. Сначала необходимо найти информацию о размерах всех компонентов. На рис. 6.15 вы можете видеть снимок экрана с сайта Amazon, на котором представлена информация о размерах товара.

### Product details

Colour Name: Black

**Product Dimensions:** 9.7 x 8 x 2.2 cm ; 240 g

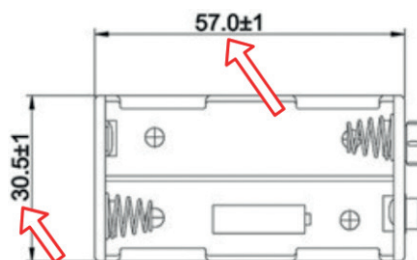
**Boxed-product Weight:** 281 g

**Delivery information:** We cannot deliver certain products outside mainland UK

**Рис. 6.15.** Технические характеристики товара

Чтобы найти технические характеристики всех компонентов, придется потрудиться. Для начала найдите магазин, в котором вы сможете их купить, например Amazon, eBay или другие онлайн-магазины. Затем можете найти или запросить информацию о размерах каждой платы или другого компонента. При этом убедитесь, что вы нашли размеры именно компонента, а не упаковки.

На рис. 6.16 показана схема батарейного блока, на которой отмечены размеры. Вы можете найти в интернете похожие схемы для своих компонентов. В данном случае размеры указаны в миллиметрах. Знаки  $\pm$  показывают производственный разброс характеристик. При проверке компоновки примем значение  $57\pm1$  как 58 мм.



**Рис. 6.16.** Чертеж покупного батарейного отсека с указанием габаритных размеров

Мои компоненты имеют следующие размеры:

- Raspberry Pi 3a +: 65×56 мм;
- шасси: 100×200 мм (имейте ввиду, что это внешние габариты, куда включены колеса);
- контроллер двигателя устанавливается на Pi, поэтому сейчас мы можем считать их одним компонентом. Он сделает нашу конструкцию выше, но учитывать такие аспекты нужно только для многоуровневых шасси;
- батарейный блок для четырех батареек AA: тот вариант, который я предлагал выше, имеет габариты 58×31,5 мм;
- внешний аккумулятор USB: 60×90 мм.

Создадим новую пустую схему на `diagrams.net` и представим наши компоненты в виде прямоугольников в натуральную величину.

Для того чтобы повторить то, что изображено на рис. 6.17, выполните следующие шаги.

1. Создайте несколько прямоугольников, выбрав фигуру на панели слева.
2. Это позволит нам создать понятное обозначение каждого компонента. Дважды щелкните на прямоугольнике, чтобы добавить подпись (название компонента). Затем нажмите **Ввод**. Я добавил надпись на *переднюю часть* шасси.
3. Выберите компонент (он будет выделен синим цветом). Щелкните на панель справа и выберете **Arrange** (Упорядочить).
4. В поля **Width** (Ширина) и **Height** (Высота) введите значения габаритов ваших компонентов (мм меняется на pt).
5. Повторите шаги 1–4 для всех элементов, чтобы скомпоновать их.

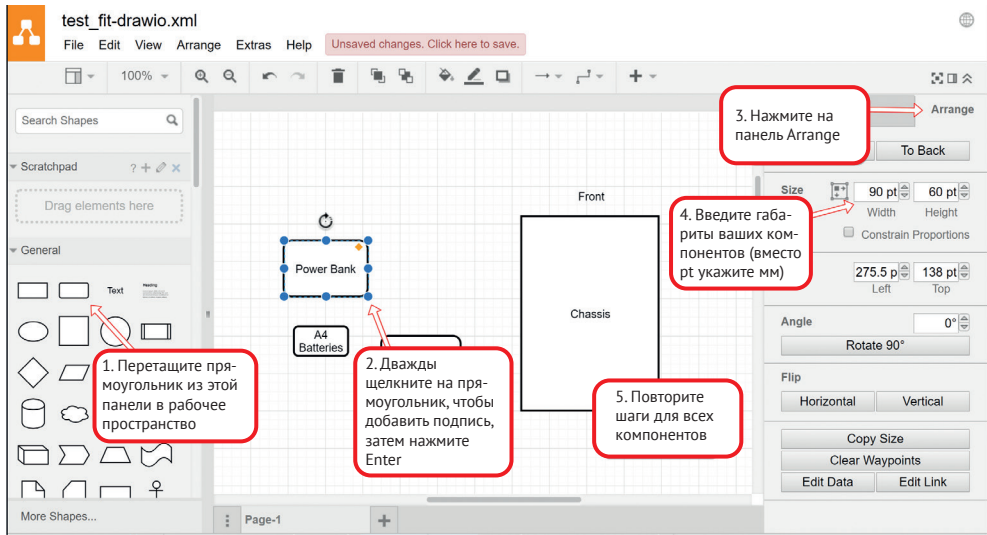


Рис. 6.17. Проверка компоновки в diagrams.net

Pi должен располагаться в передней части робота, так как позже мы будем размещать здесь сенсоры, а также к нему будут подключаться провода двигателей. На рис. 6.18 я скомпоновал все элементы. Сделать это не составит труда, поскольку при перемещении объектов появляются синие направляющие линии, служащие для центрирования и выравнивания элементов. Я поместил внешний аккумулятор ближе к задней стороне, а батарейный блок ближе к Pi, для удобного подключения проводов к контроллеру двигателя.

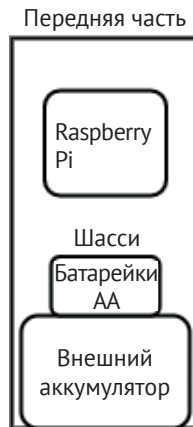


Рис. 6.18. Проверка компоновки

Итак, мы видим, что все компоненты помещаются на свои места. Разумеется, это не является абсолютной гарантией совместимости габаритов, но, скорее всего, такая конфигурация будет работать.

Пришло время приобрести компоненты. Мой список покупок выглядит так:

- набор для сборки шасси;
- полнофункциональный HAT-контроллер шагового двигателя (Full Function Stepper Motor HAT);
- батарейный блок на 4 батарейки AA;
- четыре металлгидридных батарейки AA, а также зарядное устройство (если у вас его еще нет);
- внешний аккумулятор с USB, обеспечивающий ток 3 А и более.

Теперь вы взвесили все плюсы и минусы и выбрали подходящие компоненты. Затем проверили компоновку, чтобы убедиться, что все компоненты подходят, и посмотреть, как они будут расположены. Теперь вы можете приобрести необходимые компоненты и шасси. Как только вы сделаете это, возвращайтесь к книге, и мы начнем сборку.

## СБОРКА ОСНОВАНИЯ

Если вы приобрели шасси, подобное моему, можете руководствоваться инструкцией по сборке, представленной в этом разделе. Для другого шасси я рекомендую обратиться к документации, где вы найдете инструкцию по сборке. Если вы приобрели шасси, которое сильно отличается от рекомендуемого в этой книге, в следующих главах у вас могут возникнуть некоторые затруднения.

Некоторые компоненты могут быть покрыты слоем бумаги (см. рис. 6.19), который защищает пластик от царапин. Снимите его, приподняв край ногтем или монтажным ножом. Это не обязательно, но без него робот выглядит лучше.

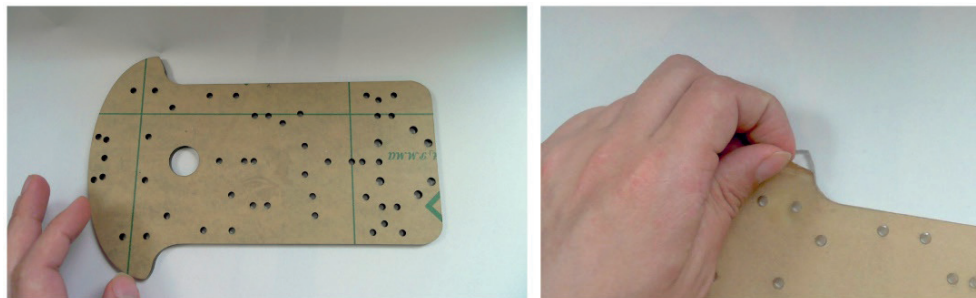
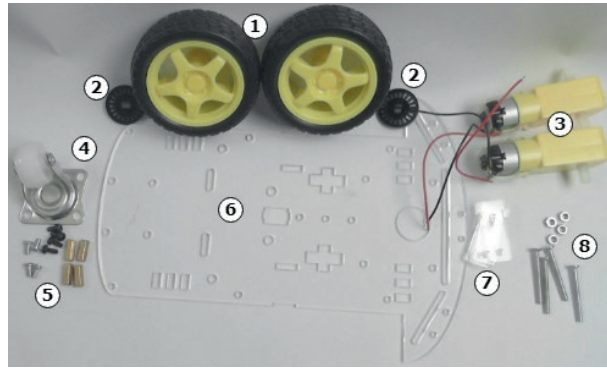


Рис. 6.19. Удаление защитного слоя

В наборы, где вырезы на шасси выполняются лазером, обычно входят желтые двигатели, которые крепятся одним из двух способов. На одних двигателях есть пластиковые крепежные опоры, а на других металлические. Давайте посмотрим, чем они отличаются. Разница играет роль только при сборке, поэтому вы можете купить то, что понравится.

На рис. 6.20 показано, что входит в комплект, где двигатели поставляются с пластиковыми крепежными опорами.

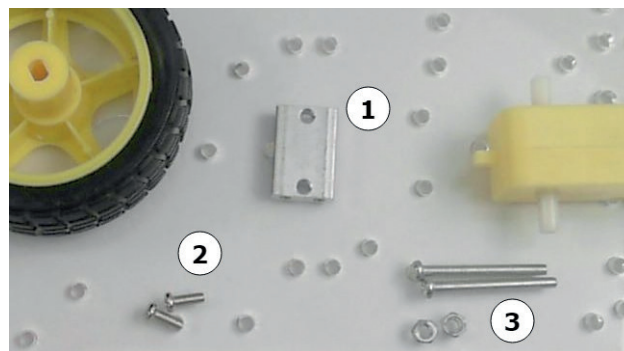


**Рис. 6.20.** Компоненты набора для сборки робота

В набор должны входить следующие компоненты:

- два колеса;
- диски энкодера;
- пара двигателей с подсоединенными проводами;
- поворотное колесо;
- болты и латунные стойки для крепления поворотного колеса. Я заменил один комплект стоек на нейлоновые, поскольку они не проводят ток. Вы можете взять их из комплекта, в который изначально входят нейлоновые стойки;
- шасси;
- пластиковые опоры для крепления двигателей. В вашем наборе они могут быть металлическими. Такие опоры крепятся немного иначе и поставляются с четырьмя дополнительными винтами;
- четыре болта и гайки для крепления двигателей.

На рис. 6.21 показаны части металлической опоры.



**Рис. 6.21.** Металлическая опора для крепления двигателя

В комплекте у вас должно быть следующее.

1. Металлическая крепежная опора (здесь заменяет пластиковую).
2. Болты для крепления опоры к шасси – металлическую опору необходимо прикрепить к шасси болтами.

3. Длинные болты для крепления опоры к двигателю (в обоих вариантах одинаковые).

### Важное примечание

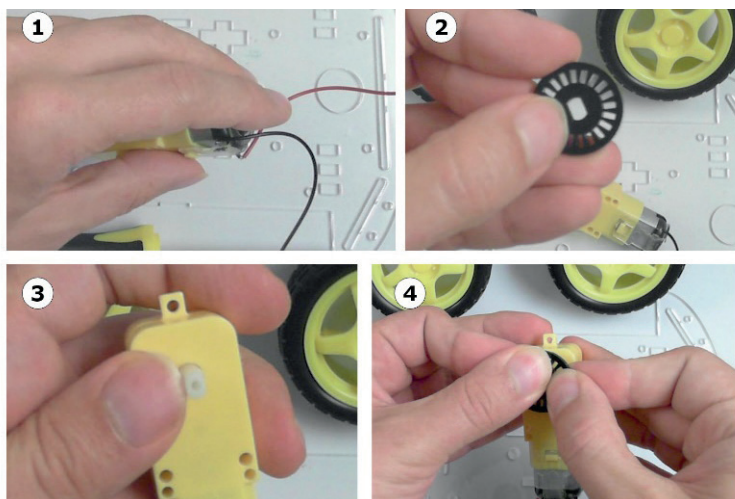
Остальные компоненты, которые не показаны здесь, выглядят во всех наборах примерно одинаково.

Теперь рассмотрим, как установить диск энкодера.

## Установка диска энкодера

Начнем с крепления дисков энкодера к двигателю. Они понадобятся нам в следующей главе, посвященной сенсорам.

Следуйте инструкциям (см. рис. 6.22):



**Рис. 6.22.** Крепление диска энкодера к двигателю

1. Посмотрите, с какой стороны к двигателю подключаются провода. С этой же стороны должен размещаться диск энкодера.
2. В диске энкодера найдите отверстие под ось со сплюснутыми сторонами.
3. Форма осей двигателей соответствует форме этого отверстия.
4. Установите диск энкодера на ось с той же стороны, где находятся провода. Осторожно покрутите диск, вы должны почувствовать небольшое сопротивление. Повторите то же самое для второго двигателя.

Теперь у вас есть два двигателя, на которых диски энкодера установлены на той же стороне, что и провода. Далее мы установим двигатели с помощью опор.

## Установка двигателя с помощью опор

Для крепления на шасси с лазерными прорезями используется два типа опор. Выберите ту инструкцию, которая подойдет вам.



## Установка двигателя с помощью пластиковых опор

### Важное примечание

Если у вас металлические опоры, пропустите этот раздел.

Для того чтобы установить двигатель посредством пластиковых опор, сначала найдите прорезы для крепления, как показано на рис. 6.23.

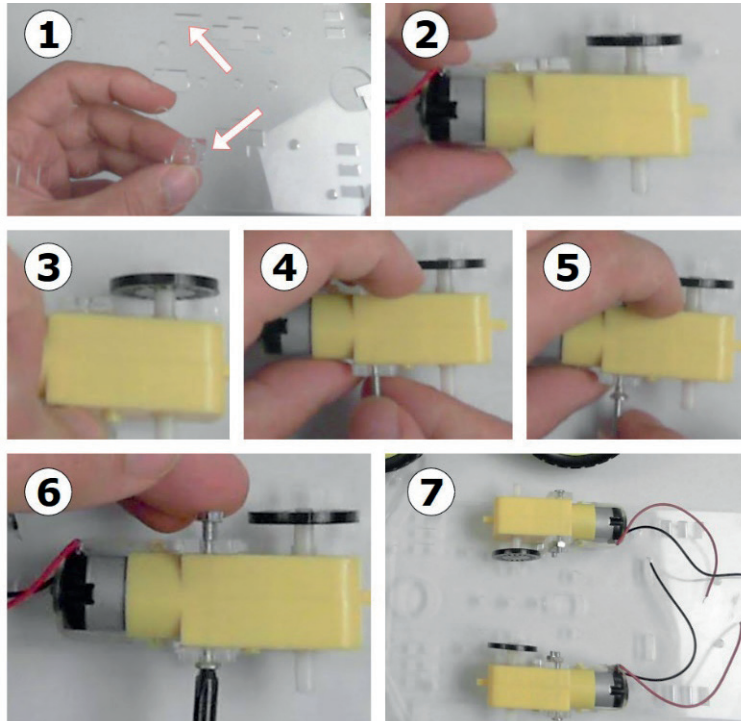


Рис. 6.23. Пластиковая опора двигателя

Для установки двигателя выполните следующие шаги.

1. Стрелки на рис. 6.23 указывают на нужные прорезы. Вставьте пластиковые опоры в прорезы.
2. Прижмите двигатель к опоре. Обратите внимание, что провода и диск энкодера обращены внутрь. Энкодер должен находиться под прорезью в панели шасси.
3. В шасси есть прорезь для внешней опоры. Вставьте еще одну опору в эту прорезь так, чтобы двигатель находился между двумя опорами.
4. Протолкните длинные винты сквозь опоры и редуктор.
5. Затем наденьте гайки на винты. Вкрутите винты с помощью отвертки.
6. У гайки, расположенной ближе к шасси, одна из ее граней должна удерживать ее от проворачивания при затягивании винта. Для удерживания внешней гайки используйте гаечный ключ или плоскогубцы.



7. Повторите то же самое с другой стороны.

Здесь рассказывается о креплении двигателей к шасси с помощью пластиковых опор. Если у вас металлические опоры, перейдите к следующему разделу.

### Установка двигателя с помощью металлических опор

Пропустите этот раздел, если вы уже прикрепили двигатель к шасси с помощью пластиковых опор. Крепление с помощью металлических опор немного отличается. Процесс крепления представлен на рис. 6.24.

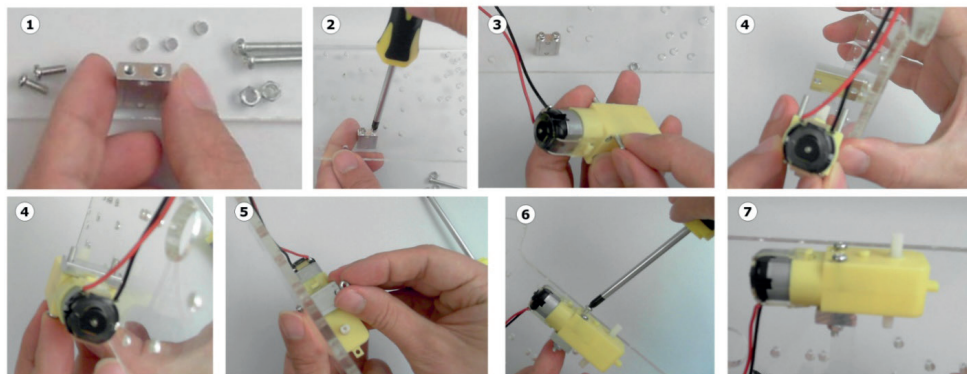


Рис. 6.24. Металлическая опора двигателя

Для того чтобы установить двигатель с помощью металлических опор, выполните следующие шаги.

1. В верхней части опоры находится два небольших отверстия для винтов, в которых нарезана резьба. Для крепления каждой опоры необходимы два коротких винта.
2. В шасси имеются предназначенные для опор отверстия рядом с местом крепления колес. Совместите их с отверстиями в опорах а затем вкрутите короткие винты в опору через отверстия в шасси.
3. Возьмите двигатель так, чтобы провода были направлены от вас. Вставьте длинные винты в отверстия в редукторе двигателя.
4. Затем возьмите двигатель и вставьте длинные винты в отверстия на боковой стороне опоры.
5. Они должны пройти сквозь опору, как показано на рис. 6.24.
6. Теперь наденьте гайки на резьбу, которая выступает с другой стороны опоры.
7. Гайки, находящиеся дальше от шасси, можно затянуть плоскогубцами или гаечным ключом и отверткой. Гайки, которые располагаются ближе к шасси, фиксируются за счет одной из граней, поэтому вам потребуется только отвертка.
8. Вы закрепили один двигатель на шасси. Повторите то же самое с другим.

Теперь на ваше шасси установлены двигатели. Они предназначены для ведущих колес. Но сначала нужно установить поворотное колесо.

## Установка поворотного колеса

Теперь необходимо установить поворотное колесо. Оно уравнивает конструкцию робота. Такое колесо не имеет двигателя, поэтому робот будет тащить его за собой. Важно, чтобы у этого колеса было небольшое сопротивление. На рис. 6.25 показана инструкция по установке поворотного колеса.

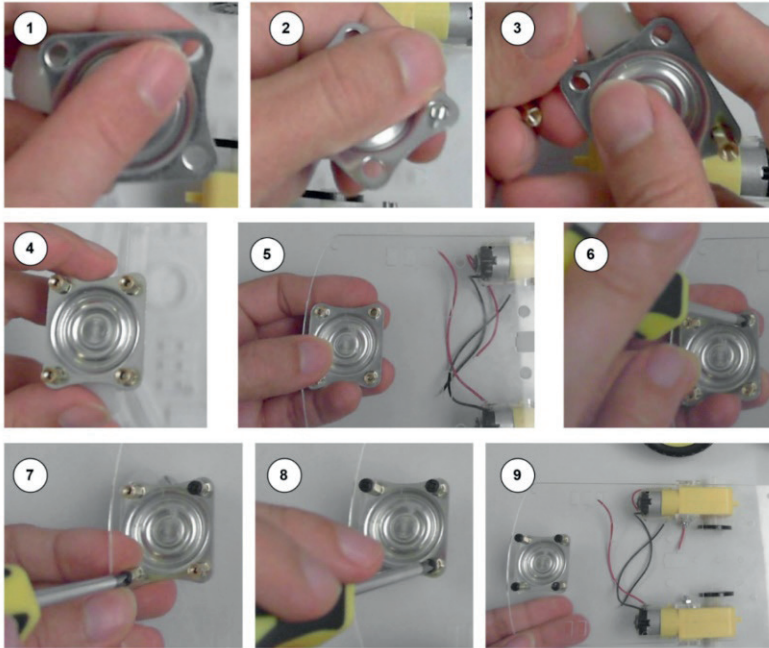


Рис. 6.25. Установка поворотного колеса

1. Это поворотное колесо. В нем есть четыре отверстия для винтов.
2. Вы должны вставить винт в отверстие так, чтобы резьба была обращена в сторону от колеса.
3. Теперь накрутите одну из латунных стоек на этот винт.
4. Повторите то же самое с другими отверстиями.
5. Совместите другую сторону стоек с четырьмя отверстиями на шасси. Обратите внимание, что основание этого поворотного колеса имеет прямоугольную, а не квадратную форму. Колесо должно быть обращено вниз.
6. Закрутите один из винтов.
7. Затем закрутите винт в противоположном углу.
8. Так вам будет легче закрутить оставшиеся винты.
9. Поворотное колесо должно быть прикреплено к роботу, как показано на этой части рисунка.

Прикрепленное поворотное колесо позволит роботу удерживать равновесие, но для полноценного движения ему нужны ведущие колеса. Давайте установим их.

## Установка ведущих колес

Ведущие колеса необходимо закрепить, как на рис. 6.26.

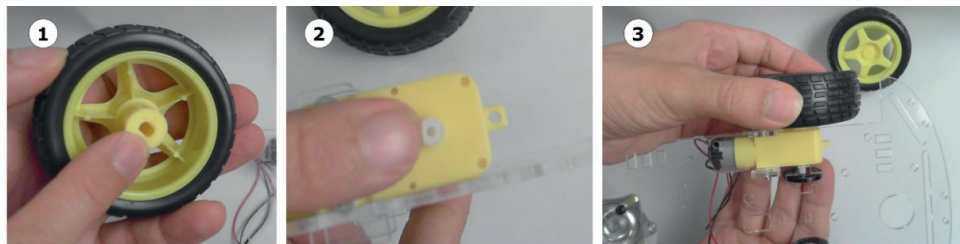


Рис. 6.26. Установка ведущих колес

Выполните следующие шаги.

1. Для начала обратите внимание, что у осевого отверстия ведущего колеса есть две сплюснутые стороны, как у диска энкодера.
2. Совместите колеса с осями и закрепите их. Не прикладывайте усилий к проводам или диску энкодера, так как их легко повредить.
3. Аккуратно проворачивайте колеса, чтобы надеть их на ось. Обязательно придерживайте двигатель с другой стороны. После этого может потребоваться выровнять диски энкодеров.

После установки колес робот становится устойчивым и начинает обретать форму. Пока вы можете катать его только вручную, но скоро он сможет перемещаться самостоятельно.

## Прокладка проводов

Следующим шагом сборки является прокладка проводов. Контроллер двигателя будет расположен на верхней части робота, поэтому провода также должны оказаться с верхней стороны шасси.

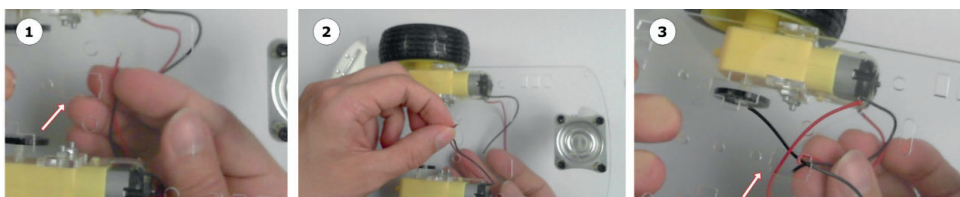


Рис. 6.27. Прокладка проводов

Для этого выполните следующие шаги.

1. Сначала возьмите два провода от одного двигателя. Найдите небольшое отверстие в середине шасси.
2. Проденьте провода через отверстие.
3. Осторожно протяните их к верхней части шасси, чтобы они немного выступали, как представлено на рис. 6.27. Повторите то же самое с проводами другого двигателя.

На рис. 6. 28 показано, как должен выглядеть робот на данном этапе (крепежные опоры двигателя могут различаться).

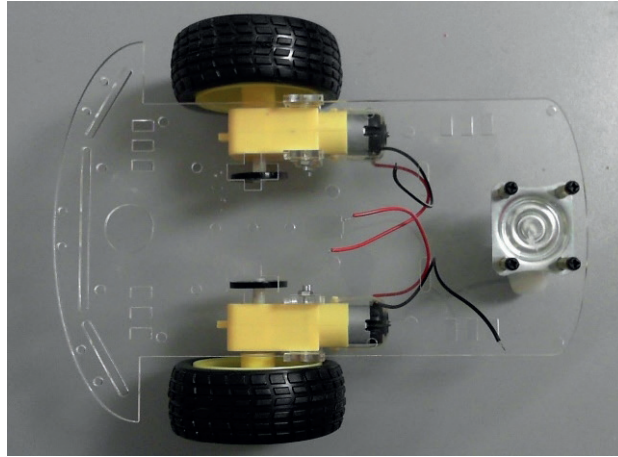


Рис. 6.28. Шасси в собранном виде

Установив двигатели и колеса, мы можем увидеть, как робот будет двигаться. Вы собрали механическую часть робота. Все провода на месте, и теперь можно добавить электронику. Начнем с установки центрального контроллера – Raspberry Pi.

## Установка Raspberry Pi

Установку контроллера двигателя мы рассмотрим в следующей главе, а сейчас установим Raspberry Pi и подготовим его для подключения других плат. Нам необходимо установить стойки на Pi, чтобы прикрутить его к шасси. При этом следует оставить место для креплений двигателя и сенсоров, которые мы установим позже.

Выполните шаги, показанные на рис. 6.29.

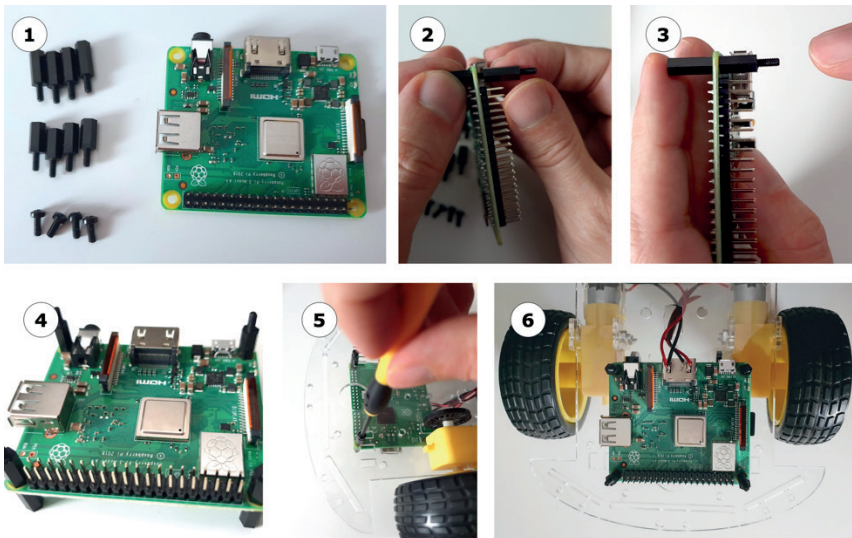


Рис. 6.29. Установка Raspberry Pi

1. Вам понадобится небольшая крестовая отвертка, маленький гаечный ключ или плоскогубцы, четыре винта длиной 5 мм с резьбой M2,5, 4 стойки 8 мм с резьбой M2,5, 4 стойки 12 мм с резьбой M2,5 и Raspberry Pi.
2. Вставьте 8 мм стойку в винтовое отверстие в нижней части Pi.
3. Затем навинтите сверху стойку высотой 10 мм так, чтобы резьба была направлена вверх. При этом придерживайте стойку плоскогубцами или гаечным ключом.
4. Повторите то же самое для всех четырех углов платы.
5. Совместите две стойки с прорезями или винтовыми отверстиями на корпусе и ввинтите в них снизу винты.
6. На шасси, которое использую я, только два отверстия, расположенных на одной линии, поэтому я пропустил винты через них и использовал две остальные стойки, чтобы просто поддерживать Pi.

Теперь у нас есть главный контроллер, который может выполнять код и управлять роботом. Но контроллеру и двигателям требуется питание.

## Установка источников питания

Итак, вы приобрели батарейный блок для четырех AA батареек, как на рис. 6.14 (а также перезаряжаемые металлгидридные батареи), и внешний аккумулятор с литий-ионным элементом, как на рис. 6.13, который заряжается через разъем USB. Мы установим источники питания на задней части робота, чтобы они уравнивали сенсоры, которые мы установим позже.

## Установка внешнего аккумулятора USB

Пока не подключайте аккумулятор к Raspberry Pi (или обязательно войдите в систему и выключите Pi согласно правилам, прежде чем вытаскивать кабель питания).

Чтобы прикрепить внешний аккумулятор к шасси, следуйте инструкции ниже (см. рис. 6.30).

1. Для того чтобы прикрепить аккумулятор, мы используем текстильную ленту «липучку».
2. Обратите внимание, что на одной стороне аккумулятора есть разъем USB. Этот разъем должен быть направлен в левую сторону. Если у аккумулятора есть светодиодный индикатор заряда, он должен быть сверху.
3. Отмерьте два отрезка ленты. Наклейте на робота две полосы одной стороны ленты (шероховатой).
4. Мягкие стороны ленты наклейте на аккумулятор. Затем соедините две части.
5. Прижмите аккумулятор к шасси и слегка пошевелите его, чтобы «липучки» прочно сцепились.
6. Так выглядит аккумулятор, установленный на шасси робота.

В качестве альтернативы «липучке» можно использовать какую-либо клейкую массу (этот вариант дешевый, но менее надежный), кабельные стяжки, двусторонний скотч или эластичные резинки. Все эти варианты подходят для аккумуляторов разных размеров.



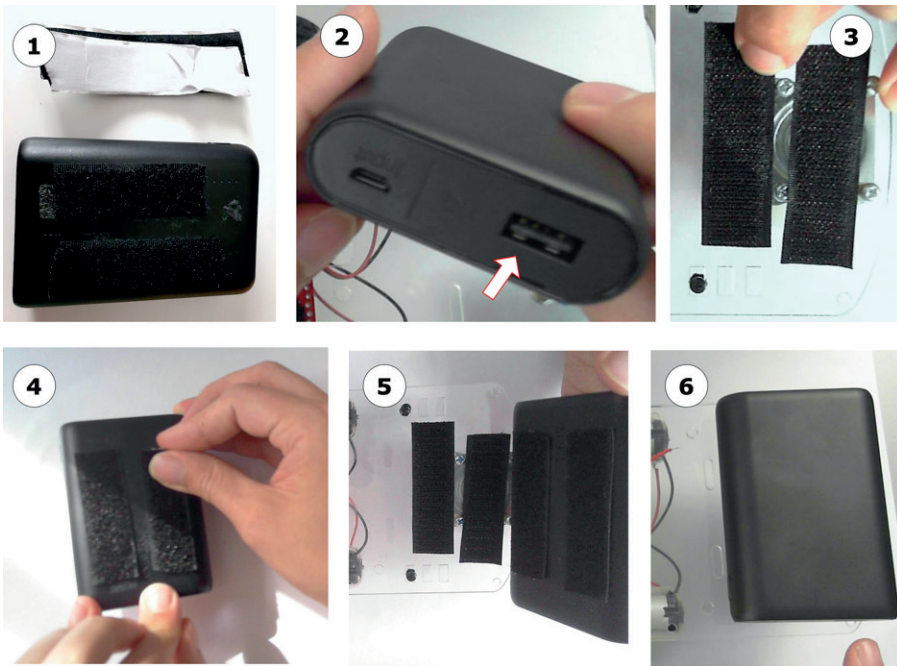


Рис. 6.30. Установка внешнего аккумулятора

### Установка батарейного блока

Батарейный блок для питания двигателей мы также закрепим с помощью «липучки». Для нас важно, чтобы его можно было снять и заменить батареи, и в этом случае «липучка» – отличный вариант. На рис. 6.31 показано, как установить батарейный блок.

1. Убедитесь, что батарейный блок пуст.
2. Отрежьте небольшую полоску ленты и приклейте одну ее сторону к нижней части батарейного блока.
3. Другую сторону ленты наклейте на робота в том месте, где вы хотите установить батарейный блок (на рис. 6.31 USB-аккумулятор снят для наглядности, и делать этого не нужно).
4. Закрепите батарейный блок на корпусе.

В крайнем случае для этого можно использовать любое достаточно липкое вещество, но помните, что батарейный блок должен быть съемным, чтобы можно было заменить батарейки.

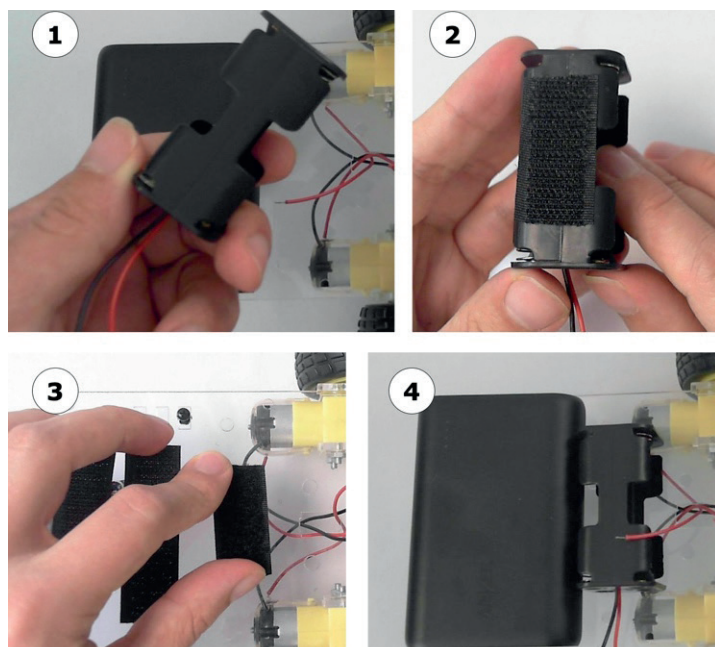


Рис. 6.31. Установка батарейного блока

## Готовое основание для робота

Вы завершили сборку основания робота. Оно должно выглядеть примерно как на рис. 6.32.

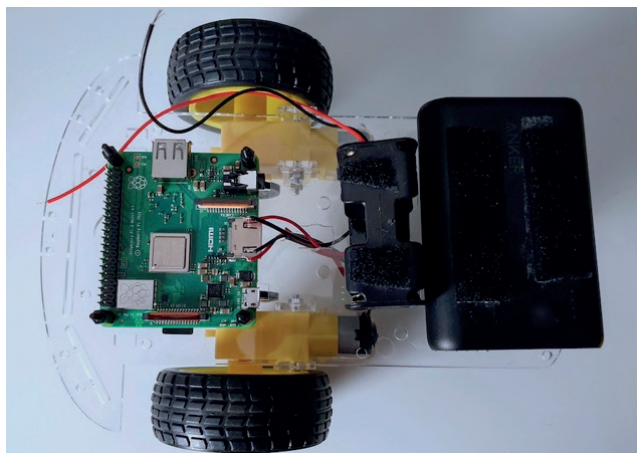


Рис. 6.32. Шасси в собранном виде

Вы собрали свое первое шасси для робота! И я надеюсь, что не последнее. Теперь, когда мы собрали шасси, установили колеса, Raspberry Pi и источники питания, робот почти готов к работе. Чтобы оживить его, необходимо подключить провода к контроллеру двигателя и написать код.



## ПОДКЛЮЧЕНИЕ ДВИГАТЕЛЕЙ К RASPBERRY PI

В этом разделе мы подключим двигатели к Raspberry Pi. После этого можно будет управлять двигателями посредством кода. На рис. 6.33 представлена блок-схема робота, которого мы создаем в этой главе. В качестве контроллера двигателя мы выбрали полнофункциональный HAT-контроллер шагового двигателя. Сокращенно будем называть эту плату HAT-платой двигателей (Motor HAT).

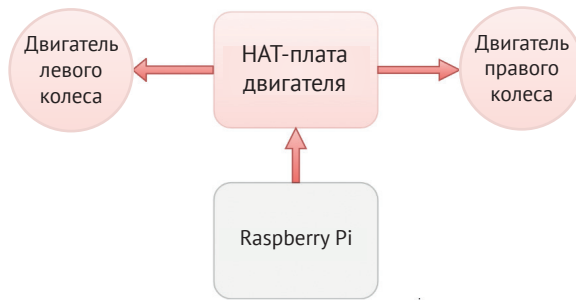


Рис. 6.33. Блок-схема робота

Данная блок-схема похожа на ту, которую мы рассматривали в главе 3. В качестве главного контроллера выбрали Raspberry Pi, который отмечен на блок-схеме серым цветом. К Pi подключена HAT-плата двигателя, куда он отправляет команды. На блок-схеме HAT-плата и двигатели выделены красным цветом. Это значит, что в этой главе мы будем заниматься именно этими компонентами. В других главах упомянутые компоненты выделены серым цветом, а красным выделяются те, которые мы будем подключать. Слева и справа от HAT-платы вы можете видеть стрелки, ведущие к двигателям. Это означает, что плата управляет двигателями.

Сначала необходимо установить HAT-плату двигателей на Raspberry Pi. Внешний вид HAT-платы показан на рис. 6.34.

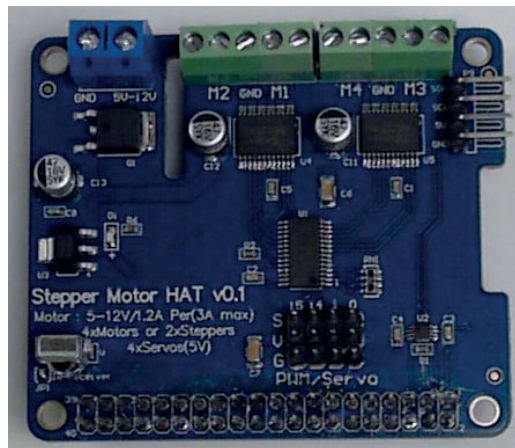


Рис. 6.34. Полнофункциональный HAT-контроллер для управления двумя шаговыми или четырьмя коллекторными двигателями

Давайте установим НАТ-плату на нашего робота и подключим ее, чтобы начать программировать нашего робота.

Выполните шаги, показанные на рис. 6.35.

1. Совместите разъем платы двигателя с линейным разъемом Pi. Резьба на стойках Pi должна войти в отверстия в плате.
2. Осторожно надавите на плату, чтобы закрепить ее на стойках Pi. Надавливайте равномерно, пока плата не будет прочно зафиксирована на линейном разъеме GPIO.
3. Теперь робот должен выглядеть, как показано на рисунке.

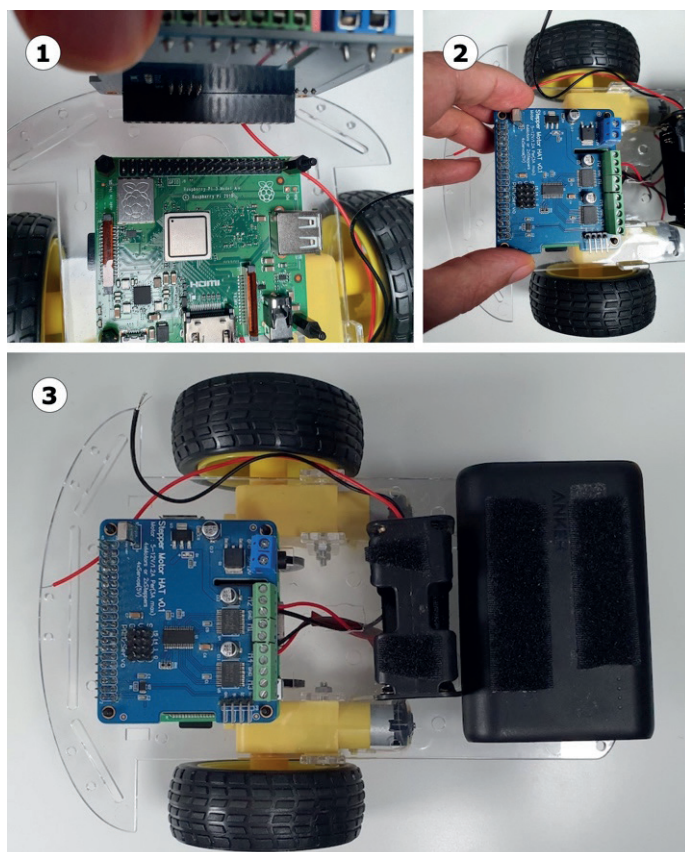


Рис. 6.35. Размещение платы управления моторами

## Подключение проводов к НАТ-плате

Теперь необходимо подключить к НАТ-плате двигатели и источник питания для них. Последнее необходимо для того, чтобы двигатели имели независимое питание, что позволит избежать сбоев, связанных с недостатком питания для Pi. Чтобы контроллер двигателя мог питать и управлять двигателями, их выводы необходимо подключить к контроллеру.

На рис. 6.36 показано, как подключить двигатели. Не подключайте «землю» питания (черный провод) батарейного блока до того, как будете готовы к подаче питания. Пока что этот провод можно закрепить небольшим кусочком изоленты на пластиковой части шасси. Не подключенный провод можно использовать как импровизированный выключатель питания.

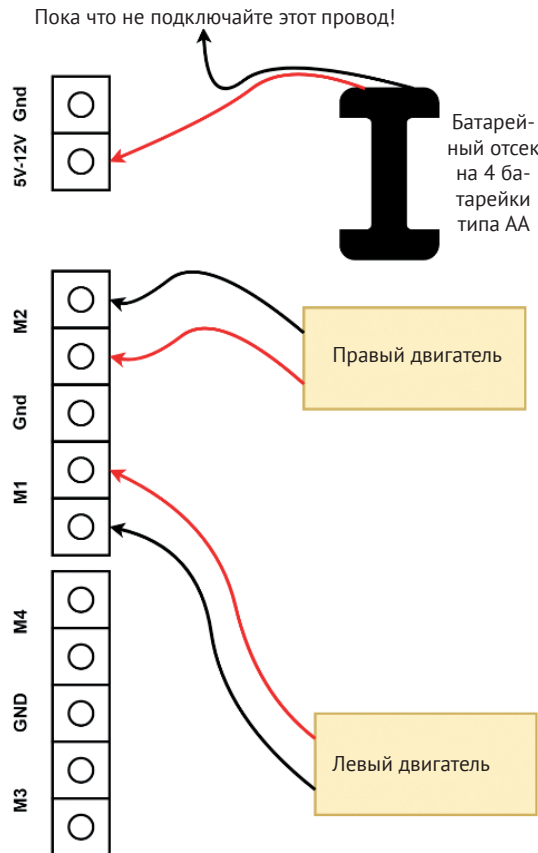


Рис. 6.36. Подключение двигателей и батарейного блока

#### Важное примечание

Черный провод батарейного блока – отрицательный провод питания, который может обозначаться как «земля» (ground), или GND. Красный – положительный, может быть обозначен как **(+ ve)**, **vIn (voltage In)** или в соответствии с номинальным входным напряжением, например 5 или 12 В.

На рис. 6.37 подключение проводов показано поэтапно.

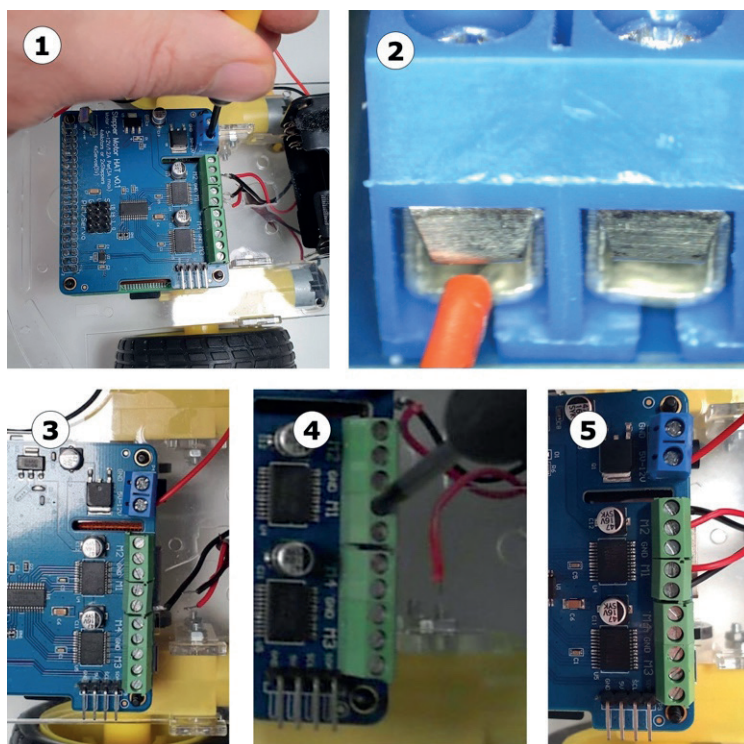


Рис. 6.37. Поэтапное подключение проводов

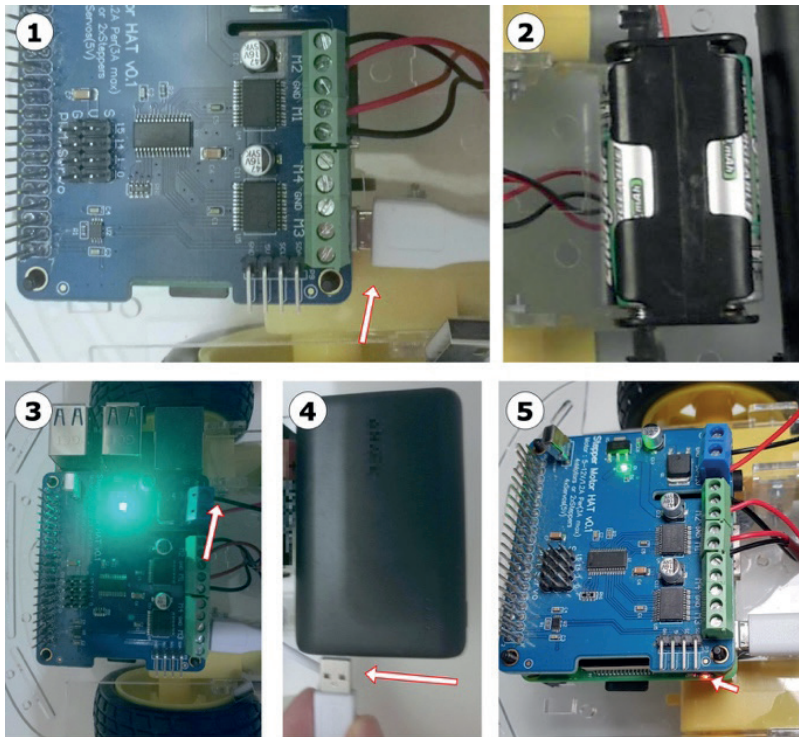
1. Ослабьте винтовые клеммы с маркировками 5V-12V , GND, M2 (2 клеммы) и M1.
2. Вставьте красный провод от батарейного блока в винтовую клемму с маркировкой 5V-12V, чтобы металлическая часть провода (токопроводящая жила) оказалась под металлическим зажимом клеммы.
3. Прочно зажмите провод, чтобы его нельзя было вытащить из клеммы. Убедитесь, что вы зачистили изоляцию с конца провода и не зажали в клемму изолированный провод.
4. Повторите то же самое с клеммами двигателя, как на картинке 4.
5. В результате вы получите то, что изображено на картинке 5.

Итак, мы подключили к контроллеру двигателя, чтобы он мог управлять ими. Мы частично подключили питание от батареи, но оставили один провод свободным, поэтому можем использовать его как выключатель питания. Попробуем запустить робота.

## Автономное питание

На этом этапе мы уже установили компоненты, которые обеспечивают автономную работу робота, и можем запустить его. Двигатели питаются от батареек AA, а Raspberry Pi от внешнего аккумулятора USB. Индикаторы на Pi показывают, что микрокомпьютер получает питание.

Выполните шаги, показанные на рис. 6.38.



**Рис. 6.38.** Переход к автономному питанию

1. Подключите кабель Micro-USB к разъему на Pi, обозначенному стрелкой.
2. Откройте батарейный блок, вставьте четыре батарейки AA и закройте его.
3. Теперь вы можете подать питание на HAT-плату двигателей. Подключите черный провод от батарейного блока к клемме GND, обозначенной стрелкой, рядом с разъемом 5V-12V. Когда вы это сделаете, на плате двигателя загорится индикатор.
4. Включите Pi, подключив кабель USB A (широким концом) к аккумулятору. При этом Micro-USB отключать не нужно.
5. Теперь на Raspberry Pi и плату двигателя подается питание, как показано на картинке 5.

Поздравляем, ваш робот теперь работает от автономного источника питания, передавая мощность двигателям. Теперь Raspberry Pi не нуждается в сетевом источнике питания.

На рис. 6.39 показано, как выглядит наш робот сейчас. На шасси установлены двигатели и Raspberry Pi с платой контроллера двигателя. У робота есть источник питания для Raspberry Pi, который сейчас включен. Источник питания для двигателей сейчас отключен. При этом черный провод заземления закреплен на шасси изолянтной, чтобы не мешать остальной части робота. Двигатели подключены к плате контроллера. Такой робот может двигаться самостоятельно когда вы подадите питание на плату контроллера двигателей.



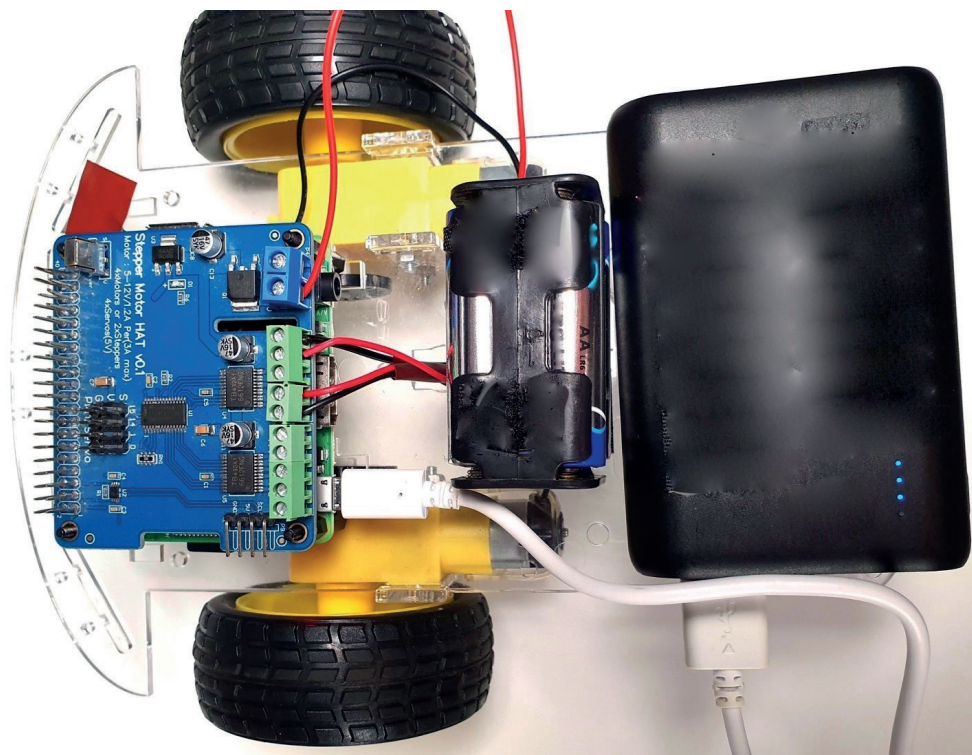


Рис. 6.39. Готовый робот

**Важное примечание**

Внезапное отключение питания Raspberry Pi может привести к повреждению SD-карты. Для того чтобы выключить Pi, войдите в систему через PuTTY и введите команду `sudo poweroff`.

Двигатели готовы к работе, а Raspberry Pi может работать без подключения к розетке. Сочетание такой системы питания и автономного управления по Wi-Fi позволяет управлять роботом с вашего компьютера.

## Выводы

В этой главе вы узнали, как выбрать компоненты для робота, а также приняли важные решения относительно конструкции. Перед приобретением деталей с помощью простого инструмента мы убедились, что все выбранные компоненты совместимы по размерам. Затем вы приобрели необходимые компоненты и собрали основание для робота.

В процессе вы приобрели навыки, которые пригодятся при планировании любого проекта, включая поиск характеристик компонентов на официальных сайтах с информацией или на сайтах магазинов, а также создание простого

эскиза, который показывает, как компоненты будут взаимодействовать друг с другом. Вы поняли, как размер робота влияет на выбор двигателя и контроллера, и узнали, что компоненты конструкции можно сделать съемными, прикрпив их с помощью ленты «липучки», а также многими другими способами.

Благодаря автономной системе питания и двигателям робот может передвигаться независимо от розетки. Теперь ему не хватает лишь кода. В следующей главе мы начнем писать код, чтобы привести робота в движение!

## Задание

- В главе 2 вы создали блок-схему для другого робота. Подумайте, какое шасси, источник питания и другие компоненты ему потребуются. Поищите подходящие компоненты в интернет-магазинах.
- Подумайте, какая НАТ-плата Raspberry Pi подойдет для этого робота. Узнать больше о расположении выводов вы можете на сайте <https://pinout.xyz>.
- Подумайте, какие системы питания подойдут для Raspberry Pi и устройств вывода в этом роботе.
- Есть ли в этом роботе компоненты, которые должны быть съемными? Как можно это реализовать? Какая может быть альтернатива ленте «липучке»? Главное – не усложняйте.
- Проверьте взаимную совместимость всех компонентов робота. Нарисуйте эскиз, где компоненты будут представлены в натуральную величину.

## Дополнительные материалы

- Узнать больше о конструкции шасси вы можете в книге «*Raspberry Pi Robotic Blueprints*», Д-р Ричард Гримметт (*Dr. Richard Grimmett*), Packt Publishing. В этой книге рассказывается, как сделать робота из радиоуправляемой машинки.
- Узнать больше о других типах шасси для роботов можно на сайте <https://www.instructables.com>, где представлено множество различных вариантов. Некоторые из них более интересные и продвинутые, чем наш робот.



# Глава 7

## Движение и повороты – код на Python для управления двигателями

В этой главе мы подключим двигатели к Raspberry Pi и напишем для них код на Python, с помощью которого будем управлять движением робота. Используя методы программирования, мы создадим программный уровень-прослойку между аппаратной частью робота и кодом, описывающим его поведение, благодаря чему впоследствии сможем обходиться минимальными доработками кода. Посредством кода мы заставим робота двигаться! После того как робот будет запрограммирован, проведем небольшой тест, где робот должен будет проехать по заданной траектории. Мы напишем поведенческие сценарии для робота, а на примере упомянутого выше теста рассмотрим их в действии.

Эта глава призвана раскрыть следующие темы:

- написание кода для проверки работы двигателей;
- рулевое управление робота;
- создание объекта Robot – кода для взаимодействия с роботом;
- разработку сценария для следования по заданной траектории.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам потребуется:

- компьютер с доступом к интернету;
- шасси, которое мы собрали в главе 6;
- контроллер двигателя из главы 6;
- участок поверхности размером 2×2 м без препятствий, по которому будет передвигаться робот.

#### Важное примечание

Со стола робот может упасть! Лучше всего запустить его на полу.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/39sHxWL>.

## РАЗРАБОТКА КОДА ДЛЯ ПРОВЕРКИ ДВИГАТЕЛЕЙ

Прежде чем мы углубимся в работу и перейдем к самой увлекательной части, двигатели нужно включить и протестировать. Это позволит нам убедиться, что они работают как надо, и избежать дальнейших проблем.

Для работы с выбранным контроллером двигателя необходимо скачать библиотеку. Для многих компонентов, кроме самых простых, существуют библиотеки интерфейсов, посредством которых можно управлять двигателями и другими устройствами, подключенными к плате. Давайте войдем в систему Raspberry Pi через PuTTY.

### Подготовка библиотек

Сейчас нам необходимо скачать код из проекта на GitHub и загрузить его на Raspberry Pi с помощью Git. Сначала необходимо установить Git на Pi. Для работы с шиной I2C на Python нам потребуются пакет `i2c-tools` и модуль `python3-smbus`. Для управления пакетами потребуется `pip`. Введите следующую команду:

```
pi@myrobot:~ $ sudo apt-get install -y git python3-pip python3-smbus i2c-tools
```

Библиотека для контроллера двигателя называется `Raspi_MotorHAT`. Вы можете скачать ее с GitHub и загрузить посредством Git. Для того чтобы ей можно было пользоваться в любом из ваших скриптов, введите следующую команду:

```
pi@myrobot:~ $ pip3 install git+https://github.com/orionrobots/Raspi_MotorHAT
Collecting git+https://github.com/orionrobots/Raspi_MotorHAT
  Cloning https://github.com/orionrobots/Raspi_MotorHAT to /tmp/pip-c3sFoy-build
Installing collected packages: Raspi-MotorHAT
  Running setup.py install for Raspi-MotorHAT ... done
Successfully installed Raspi-MotorHAT-0.0.2
```

Итак, мы подготовили библиотеки для запуска робота. Немногочисленную документацию и примеры использования библиотеки `Raspi_MotorHAT` вы можете найти по адресу [https://github.com/orionrobots/Raspi\\_MotorHAT](https://github.com/orionrobots/Raspi_MotorHAT).

### Тест – обнаружение HAT-платы двигателя

Raspberry Pi подключается к HAT-плате двигателя через **шину I2C**. Через такие шины можно отправлять и получать данные. Кроме того, одну шину может использовать несколько устройств сразу. Чтобы включить I2C, используйте `raspi-config`. Также необходимо включить **шину SPI (Serial Peripheral Interface – последовательный периферийный интерфейс)**. Она потребуется нам для подключения других плат и сенсоров. Введите следующую команду:

```
$ sudo raspi-config
```

Далее выполните следующие шаги с помощью инструмента конфигурации `raspi-config` (рис. 7.1).

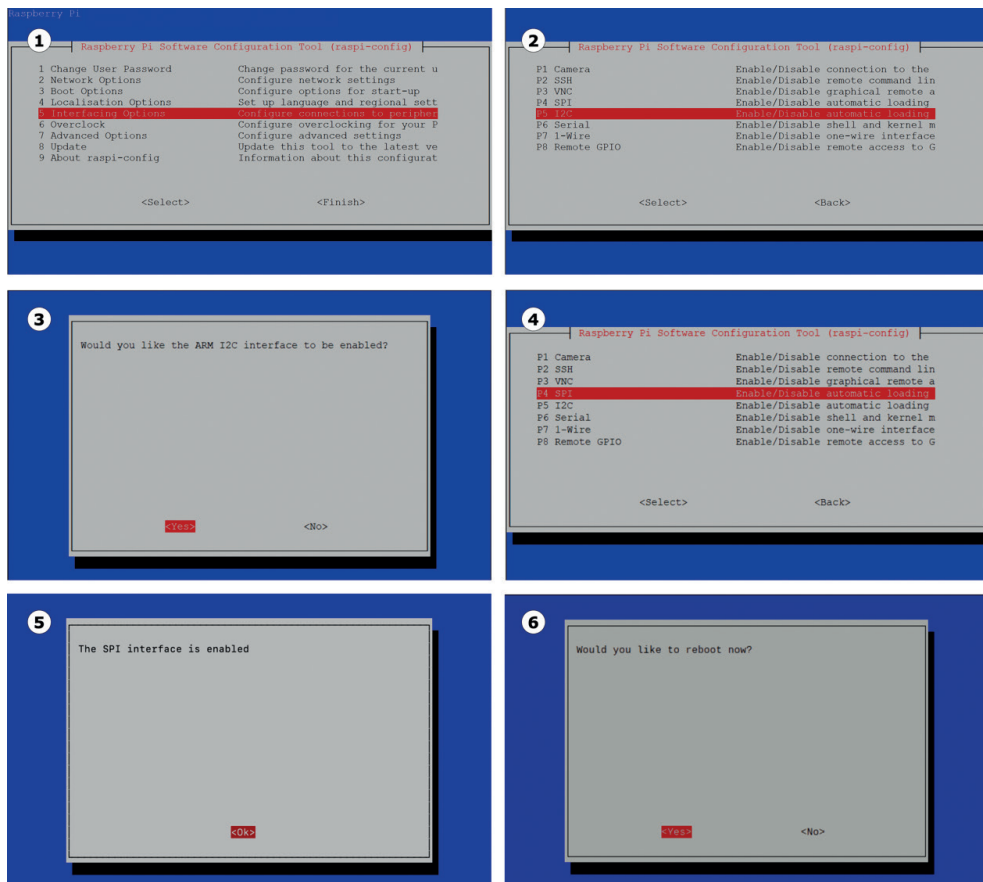


Рис. 7.1. Включение SPI и I2C с помощью инструмента конфигурации raspi-config

1. Выберите **Interfacing Options** (Параметры интерфейса).
2. Далее выберите **I2C**.
3. Pi спросит, хотите ли вы включить интерфейс. Выберите **<Yes>** (Да).
4. Вернитесь к начальному экрану, снова выберите **Interfacing Options**, затем **SPI** и нажмите **<Yes>**.
5. Далее появится экран подтверждения, который сообщит, что SPI включена. Выберите **<Ok>**.
6. Чтобы закрыть raspi-config, дважды нажмите **Esc**. После этого появится экран, где будет предложено перезагрузить Pi. Выберите **<Yes>**, а затем дождитесь завершения перезагрузки и повторного подключения к Pi. Если экран с запросом перезагрузки не появился, перезагрузите Pi с помощью команды `sudo reboot`.

Далее необходимо указать, с каким устройством мы будем взаимодействовать через I2C, т. е. определить адрес платы. Этот адрес играет примерно ту же роль, что и номер конкретного дома на улице, – помогает вам оказаться в нужном месте.

Необходимо убедиться, что Raspberry Pi видит HAT-плату двигателя. Сделать это можно с помощью команды `sudo i2cdetect -y 1`, вывод которой выглядит следующим образом:

```
pi@myrobot:~$ sudo i2cdetect -y 1
 0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- 6f
70: 70 -- -- -- -- -- -- --
```

Эта команда сканирует шину I2C 1 на предмет подключенных к Raspberry Pi устройств, и в выводе мы видим их адреса. Устройство, расположенное по адресам 6f и 70, – наш контроллер двигателя. Если команда не обнаружила его, выключите Raspberry Pi, проверьте подключение устройства и повторите попытку.

Адреса представлены в шестнадцатеричной системе счисления, где используются цифры 0-9 и буквы A-F вместо десяти цифр. Для записи шестнадцатеричных чисел в коде используют префикс 0x (*ноль* и строчная буква x).

Итак, мы включили шину I2C (и SPI), а затем нашли контроллер двигателя с помощью инструмента `i2cdetect`. Мы получили ответ от контроллера и убедились, что он подключен, а также определили его адрес – 0x6f. Теперь можем отправлять ему команды.

## Тест – демонстрация работы двигателей

Необходимо убедиться, что двигатели работают. Для начала создадим текстовый файл. Выполните следующие действия.

1. Создайте файл `test_motors.py` и добавьте в него следующий фрагмент кода:

```
from Raspi_MotorHAT import Raspi_MotorHAT

import time
import atexit

mh = Raspi_MotorHAT(addr=0x6f)
lm = mh.getMotor(1)
rm = mh.getMotor(2)

def turn_off_motors():
    lm.run(Raspi_MotorHAT.RELEASE)
    rm.run(Raspi_MotorHAT.RELEASE)

atexit.register(turn_off_motors)

lm.setSpeed(150)
rm.setSpeed(150)

lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.FORWARD)
time.sleep(1)
```

- Загрузите этот файл на Raspberry Pi, используя методы, описанные в главе 5.

### Важное примечание

Переместите робота со стола на пол, поскольку он может начать двигаться в неожиданном направлении и упасть!

- Чтобы запустить код на Pi через PuTTY, введите команду:

```
pi@myrobot:~ $ python3 test_motors.py
```

Робот должен начать двигаться вперед. Движение может немного отклоняться от прямой линии, но при этом робот не должен поворачивать или двигаться назад. Работать должны оба двигателя.

## Устранение неполадок

При возникновении проблем обратитесь к таблице ниже.

Проблема	Возможная причина и решение
Ошибка ImportError: no module named smbus	Возможно, вы не установили требуемый пакет. Установите его с помощью apt-get и устраните любые связанные с ним ошибки
Ошибки в коде Python	Вернитесь к коду и внимательно проверьте его на наличие ошибок. Проверьте, загрузили ли вы его в Raspberry Pi. Убедитесь, что вы ввели команду python3, а pip установлен правильно
Одно или оба колеса вращаются в обратную сторону	Неправильно подсоединены провода двигателя. Поменяйте местами черный и красный провода на клеммах двигателя (рис. 7.4 и 7.5)
Индикатор светится, но один или оба двигателя не работают	Убедитесь, что провода двигателей надежно подсоединены к клеммам. Для справки обратитесь к соответствующему подразделу («Подключение проводов к HAT-плате») в главе 6
Индикатор не светится, и двигатели не работают	Убедитесь, что вы подключили провода к нужным клеммам (см. раздел «Автономное питание» в главе 6). Проверьте заряд батареек
Робот поворачивает или двигается в противоположную сторону	Робот может слегка отклоняться от нужного направления. Если он двигается в противоположную сторону, проверьте подключение двигателей, а также убедитесь, что колеса и диски энкодера не заедают / не цепляются за шасси

Теперь ваш робот двигается вперед, а вы знаете, как устранять неполадки.

## Как работает код

Итак, наши двигатели работают, а робот управляет ими с помощью кода в test\_motors.py. Но как работает этот код? В этом разделе мы рассмотрим данный вопрос подробнее.

Первые строки кода отвечают за импорт библиотек:

```
from Raspi_MotorHAT import Raspi_MotorHAT
import time
import atexit
```

Импорт делает библиотеки доступными для использования в коде Python. Библиотека `Raspi_MotorHAT` предназначена для взаимодействия с нашими двигателями. Библиотека `time` позволяет работать со временем. В данном случае мы используем эту библиотеку для того, чтобы приостановить работу двигателей на заданное количество секунд. Библиотека `atexit` позволяет выполнить нужный код при закрытии программы.

Следующие строки отвечают за подключение библиотеки к HAT-плате двигателя и самим двигателям:

```
mh = Raspi_MotorHAT(addr=0x6f)
lm = mh.getMotor(1)
rm = mh.getMotor(2)
```

Первая строка создает объект `Raspi_MotorHAT` с I2C-адресом (`addr`) `0x6f`. Этот объект мы видим при сканировании шины. Имя возвращенного объекта `mh` – аббревиатура от `Raspi_MotorHAT`.

Сокращения `lm` и `rm` – это названия левого и правого двигателей соответственно. Мы получаем эти элементы управления двигателем от объекта `mh`. Каждый из них отмечен номером двигателя, указанным на плате. `Motor 1` – левый двигатель, `Motor 2` – правый.

Далее мы объявляем функцию `turn_off_motors`, которая запускает `Raspi_MotorHAT`. В следующем фрагменте кода `RELEASE` – это инструкция, указывающая на остановку двигателей.

```
def turn_off_motors():
    lm.run(Raspi_MotorHAT.RELEASE)
    rm.run(Raspi_MotorHAT.RELEASE)
atexit.register(turn_off_motors)
```

Далее следует команда `atexit.register (turn_off_motors)`, которая запускается в конце файла при завершении процесса Python. Команда `atexit` сработает даже при наличии ошибок. Отсутствие этой команды может привести к неправильному исполнению кода, и робот продолжит движение. В таком случае робот может упасть со стола или врезаться в стену. Если робот предпримет попытку продолжить движение, когда двигатели уже не могут вращаться, это может привести к повреждению двигателей, контроллеров двигателей и аккумуляторов, поэтому важно не допустить такой ситуации.

Частота вращения двигателя для этого контроллера/библиотеки может быть установлена в диапазоне от 0 до 255. Наш код устанавливает частоту вращения каждого двигателя чуть выше среднего значения этого диапазона, а затем запускает режим `Raspi_MotorHAT.FORWARD`, который заставляет каждый двигатель вращать колеса вперед, как показано в следующем фрагменте кода:

```
lm.setSpeed(150)
rm.setSpeed(150)

lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.FORWARD)
```

После этого мы указываем, что необходимо приостановить исполнение кода на 1 с, как показано ниже:

```
time.sleep(1)
```

Эта команда заставит двигатели вращать колеса вперед в течение 1 с. Затем программа завершится. Поскольку мы указали программе останавливать двигатели при завершении кода, двигатели также остановятся.

Итак, мы написали код для тестирования двигателей и разобрались как он работает. Также мы увидели его в действии. На данном этапе у вас есть «жизнеспособный» робот. Помимо этого, вы научились использовать импорты Python. Вы узнали, как пользоваться командой `atexit` для прекращения работы двигателей и управления таймером, благодаря чему робот может запускаться до завершения кода. Далее мы рассмотрим рулевое управление робота.

## РУЛЕВОЕ УПРАВЛЕНИЕ РОБОТА

Мы научили робота двигаться вперед. Но как им управлять? Как заставить его поворачивать? Сначала необходимо разобраться в типах рулевого управления. Давайте рассмотрим некоторые из них и остановимся на том, который подойдет для нашего робота, а затем напомним соответствующий код.

### Типы рулевого управления

Наиболее распространенные способы управления колесным транспортным средством (включая робота) относятся к двум основным категориям – с помощью управляемых и фиксированных колес. Мы обсудим их далее, а затем рассмотрим необычные варианты реализации методов рулевого управления.

#### Управляемые колеса

Такая конструкция предполагает наличие подвижных колес, где направление одного (или нескольких) из них может меняться. Изменение направления колес во время движения заставляет робота поворачивать. На рис. 7.2 представлено два вида такого управления.

Зелеными стрелками обозначено направление движения. Белые стрелки показывают изменение положения робота и угла поворота колес. На рис. 7.2 мы можем отметить следующее.

1. Зачастую используется **реечный механизм управления**. Когда колеса направлены прямо, транспортное средство движется вперед.
2. При перемещении рулевой рейки (отмечено белыми стрелками) транспортное средство поворачивает.
3. Еще один распространенный тип управления – **система управления поворотом оси**. Такое управление часто реализуется в самодельных гоночных картах. Когда колеса направлены прямо, транспортное средство движется вперед.
4. Рулевое управление транспортным средством осуществляется за счет поворота передней оси.



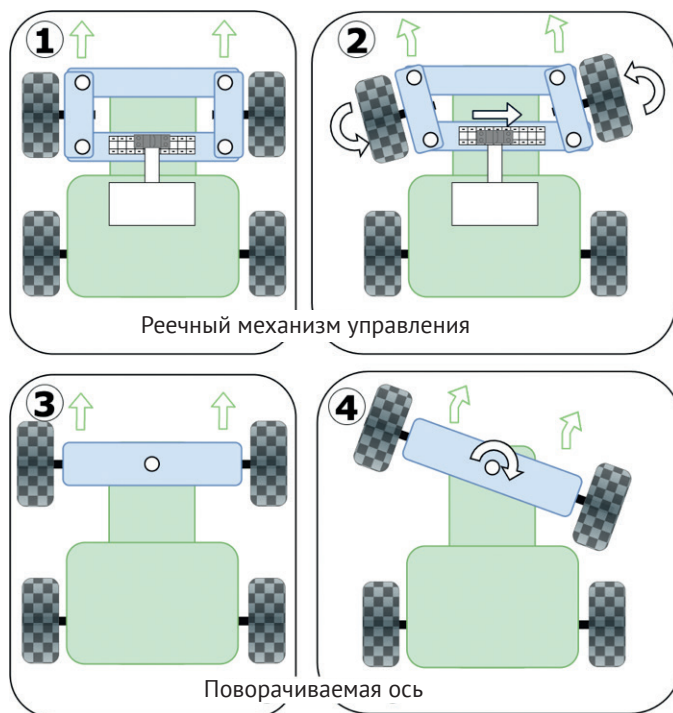


Рис. 7.2. Виды управляемых колес

Существуют и другие системы рулевого управления на основе управляемых колес:

- колеса некоторых роботов могут менять направление независимо друг от друга, благодаря чему робот может двигаться боком;
- принцип Аккермана в рулевом управлении, где каждое колесо может быть повернуто под разным углом;
- механизм с задними управляемыми колесами (rear steering), при котором управляются как передняя, так и задняя колесные пары. Такой механизм подходит для длинных транспортных средств.

Примером реализации системы управления поворотом оси является робот Unotron, показанный на рис. 7.3. Этого робота собрал мой сын на основе шасси Unotron от 4tronix и контроллера Arduino Nano.

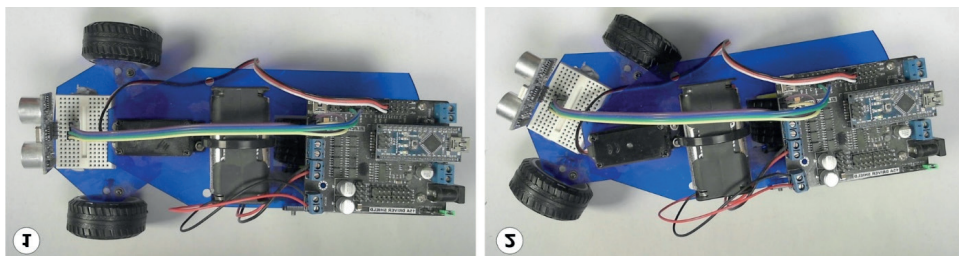


Рис. 7.3. Робот Unotron с системой управления поворотом оси

В конструкции Unotron сзади (под контроллером двигателя) находится одно колесо с приводом от двигателя. Передняя колесная пара управляется сервоприводом, который поворачивает всю переднюю часть.

Такая система обладает рядом недостатков, среди которых ее габариты, вес и сложность конструкции. Шасси, в котором предусмотрено рулевое управление с подвижными колесами, требует больше движущихся частей и места для размещения колес. Unotron прост настолько, насколько это возможно. Как правило, подобные конструкции более сложные и требуют серьезного технического обслуживания.

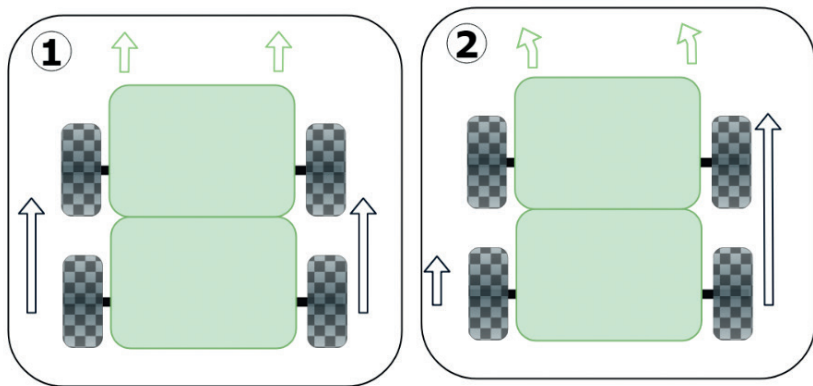
Роботам с управляемыми колесами требуется больше места для поворота (большой радиус поворота), поскольку для изменения направления колес требуется движение вперед/назад.

Для фиксированной оси вам потребуется один большой двигатель, поскольку в таком случае распределять мощность между двумя двигателями невозможно, или же вам понадобятся сложные механизмы для равномерного распределения подводимой энергии. Если механизм не отцентрирован, робот будет поворачивать.

### Фиксированные колеса

Системы рулевого управления с фиксированными колесами предполагают фиксацию колесных осей относительно шасси. Такие системы используются довольно широко. В них направление робота определяет относительная скорость каждого колеса или колесной пары. То есть колеса не поворачиваются относительно направления робота, но если скорость вращения одного колеса (или колесной пары) выше, то робот будет поворачивать. Такую систему часто называют системой бортового поворота или поворотом с подтормаживанием.

На рис. 7.4 показана система бортового поворота в действии. Белые стрелки указывают относительную частоту вращения двигателей. Зеленые стрелки показывают направление робота.



**Рис. 7.4.** Система рулевого управления с фиксированными колесами или система бортового поворота

Здесь мы видим следующее.

1. Частота вращения двигателей одинаковая, поэтому робот движется вперед.
2. Относительная частота вращения двигателей справа выше, чем у двигателей слева. Робот движется вперед и влево.

У такой системы есть несколько преимуществ. Она хорошо подходит для роботов, передвигающихся на гусеницах. С механической точки зрения система довольно проста: все, что нужно для ее реализации, – это приводной двигатель на каждое колесо. Система бортового поворота позволяет роботу выполнять поворот на месте на  $360^\circ$  относительно самой широкой/длинной части робота.

Также у системы есть и недостатки. При повороте колесо может тянуть вбок, что вызывает сильное трение. Кроме того, любые незначительные различия в двигателях, их зубчатой передаче или выходном сигнале контроллера могут привести к изменению направления движения.

### Другие системы рулевого управления

Выбранный нами контроллер позволяет управлять четырьмя каналами двигателей. Этого достаточно, чтобы оснастить робота специальным типом колес – колесами Mecanum. Такие колеса позволяют роботу двигаться в любом направлении, совмещая бортовой поворот и движение вбок. Технически их можно отнести к рулевому управлению с фиксированными колесами. На рис. 7.5 показан робот на колесах Mecanum.



**Рис. 7.5.** Робот Uranus Pod (автор Gwpcsm) на колесах Mecanum [CC BY 3.0 (<https://creativecommons.org/licenses/by/3.0>)]

Эти колеса удивительно функциональные, но при этом сложны с точки зрения механики, ввиду чего требуют особого обслуживания. При этом они довольно тяжелые и сравнительно дорогие. Однако работать с ними очень увлекательно.

### Рулевое управление нашего робота

Наш робот имеет три колеса: два независимых друг от друга ведущих колеса с каждой стороны и одно поворотное. Мы реализуем систему бортового поворота. Управление роботом будет осуществляться посредством изменения ча-

стоты и направления вращения двигателей колес. Благодаря этой системе робот сможет выполнять поворот на 360°. Поворотное колесо позволит избежать проблем, связанных с трением, присущих четырех- и шестиколесным роботам с системой бортового поворота.

Внеся всего одно изменение в код, мы можем заставить робота выполнять поворот на месте. При этом одно колесо будет вращаться назад, а другое вперед. Давайте посмотрим, как это сделать.

1. В `test_motors.py` найдите следующие строки:

```
lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.FORWARD)
```

2. Во второй строке вместе FORWARD введите BACKWARD

```
lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.BACKWARD)
```

3. Запустите код на Pi, введя в `test_motors.py` команду `python3`. Теперь ваш робот вращается по часовой стрелке. Чтобы заставить его вращаться в другую сторону, укажите значение BACKWARD для левого двигателя (lm) и FORWARD для правого (rm).

4. А что насчет более плавных поворотов? В предыдущей версии кода, перед строками, задающими направление, мы добавили строки, устанавливающие частоту вращения для каждого двигателя:

```
lm.setSpeed(150)
rm.setSpeed(150)

lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.FORWARD)
```

Для более плавного поворота необходимо установить для обоих двигателей (lm и rm) значение FORWARD и при этом установить частоту вращения одного двигателя меньше, чем другого, например:

```
lm.setSpeed(100)
rm.setSpeed(150)

lm.run(Raspi_MotorHAT.FORWARD)
rm.run(Raspi_MotorHAT.FORWARD)
```

Такой код заставит робота двигаться вперед и плавно поворачивать влево.

В этом разделе мы рассмотрели методы управления направлением движения робота. В соответствии с конструкцией нашего робота в дальнейшем мы применим один из методов на практике, заставив робота выполнять поворот на месте, двигаться вперед и поворачивать. В следующем разделе мы представим все это в виде отдельного уровня кода, который будет определять поведение робота при различных сценариях.

## Создание объекта Робот – код для взаимодействия с РОБОТОМ

Теперь, когда мы научили робота двигаться и поворачивать, перейдем к следующему уровню программного обеспечения, который объединит несколько

аппаратных функций и позволит сделать **сценарии поведения** робота независимыми от них. Под сценарием поведения я имею в виду код, заставляющий робота вести себя определенным образом, например следовать по линии или избегать столкновений со стенами. Для чего нужно такое разделение на разные уровни кода?

При выборе контроллера двигателя мы тщательно рассмотрели все достоинства и недостатки различных плат, чтобы найти наиболее подходящий вариант. Однако по мере появления новых идей может возникнуть необходимость заменить контроллер. Также новый контроллер потребуется для новых роботов. Несмотря на то что такие функции, как управление частотой вращения и изменение направления вращения двигателей, относятся к одному типу операций, каждый контроллер выполняет их по-разному. Создание дополнительного уровня перед контроллером позволит нам использовать прежние команды основного кода даже при замене контроллера. Этот уровень будет служить «фасадом» или интерфейсом для взаимодействия с функционалом робота.

У каждого контроллера есть свои особенности. Наш, например, позволяет устанавливать режим работы и частоту вращения двигателей. Во многих контроллерах значение 0 означает остановку, которая осуществляется за счет удержания двигателя. В нашем контроллере используется режим RELEASE, который немного отличается. Для движения назад во многих контроллерах значение частоты вращения задают в виде отрицательного числа, а у нашего для этого есть режим BACKWARD. Значение частоты вращения может быть установлено в диапазоне от 0 до 255. В некоторых контроллерах диапазон частоты вращения отличается, например от –128 до 128 или от 0 до 10. Для того чтобы спрятать особенности нашего контроллера за кулисы, мы создадим интерфейсный объект.

## Для чего нужен объект?

Мы разрабатываем интерфейс для того, чтобы получить возможность взаимодействовать с другим кодом. Интерфейс позволяет упростить базовые системы и сделать их более согласованными, ввиду чего основной код будет подходить для всех упомянутых ранее типов контроллера. Также посредством создания интерфейсов можно разделять код на уровни. Деление на уровни позволяет изменять одну часть кода независимо от другой, как показано на рис. 7.6.

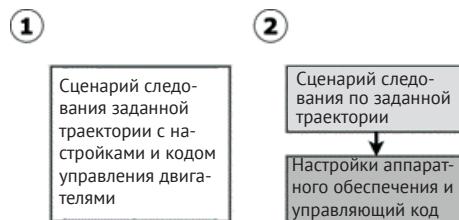


Рис. 7.6. Уровни программного обеспечения

В блоке 1 на рис. 7.6 представлен код, в котором нет разделения систем на уровни. Внести сюда изменения, например добавить новый поведенческий

сценарий или заменить контроллер двигателя, крайне сложно. Таких подходов лучше избегать.

В блоке 2 представлена система, разделенная на два уровня, которые взаимодействуют между собой. Например, они могут взаимодействовать, когда сценарий следования заданной траектории управляется совокупностью аппаратного обеспечения и управляющего кода.

В этой книге мы разработаем множество различных поведенческих сценариев. Время от времени мы будем расширять библиотеку аппаратного контроллера. В конце концов, кому захочется разрабатывать один и тот же код заново? Вместо этого вы сможете использовать этот уровень снова и снова, дополняя его новыми сценариями.

Блок 2 на рис. 7.6 – это блок настройки/управления аппаратного обеспечения нашего объекта *Robot*. Этот объект является интерфейсом, скрывающим за собой особенности платы полнофункционального *HAT-контроллера шагового двигателя*.

Объект обладает стандартным интерфейсом, что означает, что он совместим с другими роботизированными системами, и им будут доступны все поведенческие сценарии. Некоторые робототехники-профессионалы используют интерфейсы в качестве замены реальных контроллеров, чтобы моделировать сложные поведенческие сценарии.

## Что мы поместим в объект?

Объект – это строительный блок для создания интерфейсов в Python. У объектов есть методы, которые мы можем вызывать для выполнения задач. Также объекты могут содержать элементы, некоторые данные или ссылки на другие объекты.

Далее мы напишем код для объекта *Robot*, который позволит делать следующее:

- настраивать *HAT-плату двигателя* и двигатели как элементы с именами `left_motor` и `right_motor`;
- управлять состоянием `exit`;
- останавливать работу двигателей с помощью метода `stop_motors`;
- я предлагаю обозначать значения частоты вращения (от 0 до 100) как проценты. Мы будем пересчитывать значение частоты вращения в соответствии с требованием контроллера;
- режимы специфичны для данного контроллера. В нашем интерфейсе отрицательные значения частоты вращения означают движение назад;
- на более позднем этапе объект *Robot* может действовать как диспетчер шин данных, которым требуется код для блокировки взаимного доступа для них и другого оборудования;
- наш интерфейс (и, следовательно, наш объект) не содержит других сценариев поведения, кроме защитного отключения двигателей при выходе из программы.

В файл `robot.py` необходимо добавить следующий код:

```

from Raspi_MotorHAT import Raspi_MotorHAT
import atexit

class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Настройка HAT-платы в соответствии с переданным адресом
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)

        # получение локальных переменных для каждого двигателя
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)

        # убедимся, что после завершения кода двигатели останавливаются
        atexit.register(self.stop_motors)

    def stop_motors(self):
        self.left_motor.run(Raspi_MotorHAT.RELEASE)
        self.right_motor.run(Raspi_MotorHAT.RELEASE)

```

Данный class имеет специальный метод `__init__`, который создает уровень. Метод `__init__` сохраняет выходные данные методов `getMotor` из библиотеки `Raspi_MotorHAT` в элементах `left_motor` и `right_motor`. Также он регистрирует систему остановки. Я добавил несколько комментариев, чтобы указать, для чего предназначены те или иные строки кода.

Итак, объект `Robot` нашел и настроил HAT-плату двигателя, и теперь он может останавливать двигатели. Полученный нами результат – это тот же самый код, который мы видели раньше, но сейчас он имеет немного другую структуру.

Мы можем протестировать код в другом файле, назовем его `behavior_line.py`:

```

import robot
from Raspi_MotorHAT import Raspi_MotorHAT
from time import sleep

r = robot.Robot()
r.left_motor.setSpeed(150)
r.right_motor.setSpeed(150)
r.left_motor.run(Raspi_MotorHAT.FORWARD)
r.right_motor.run(Raspi_MotorHAT.FORWARD)
sleep(1)

```

Сначала происходит импорт недавно созданного нами файла `robot.py`. В соответствии с представленным здесь фрагментом кода робот будет двигаться вперед 1 с, а затем остановится. Запустите `behavior_line.py` с помощью `python3`.

Необходимо установить определенную частоту вращения двигателей в соответствии с требованиями платы (не более 100). Внесем в `robot.py` соответствующие изменения (выделено жирным шрифтом):

```

from Raspi_MotorHAT import Raspi_MotorHAT
import atexit

class Robot(object):
    def __init__(self, motorhat_addr=0x6f):
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)
        atexit.register(self.stop_motors)

```



```
def convert_speed(self, speed):
    return (speed * 255) // 100

def stop_motors(self):
    self.left_motor.run(Raspi_MotorHAT.RELEASE)
    self.right_motor.run(Raspi_MotorHAT.RELEASE)
```

Теперь мы можем использовать функцию `convert_speed`, чтобы указывать частоту вращения двигателей в диапазоне от 0 до 100. Для нашей HAT-платы функция выводит значения в диапазоне 0 до 255. Для других плат будут вычислены другие выходные значения.

Мы умножаем частоту вращения на 225 и делим полученный результат на 100. Эта формула позволяет представить процент как долю числа 255. Сначала нужно выполнить умножение, потому что мы выполняем целочисленные вычисления. При целочисленном делении 80 на 100 получится 0, но если поделить произведение ( $80 \cdot 225$ ) на 100, мы получим 204.

Полученный в результате код остается неудобным для использования. Для его использования в `behavior_line.py` необходимо внести следующие изменения:

```
import robot
from Raspi_MotorHAT import Raspi_MotorHAT
from time import sleep

r = robot.Robot()
r.left_motor.setSpeed(r.convert_speed(80))
r.right_motor.setSpeed(r.convert_speed(80))
r.left_motor.run(Raspi_MotorHAT.FORWARD)
r.right_motor.run(Raspi_MotorHAT.FORWARD)
sleep(1)
```

Здесь мы по-прежнему используем специфичные для нашей платы контроллера методы `run` и `setSpeed` из библиотеки `Raspi_MotorHAT`. В других платах методы будут другие. Далее можно преобразовать код.

Начнем с изменения метода `convert_speed`. Для робота использовать отрицательные значения для обозначения движения назад довольно удобно. Помимо определения диапазона частоты вращения, также необходимо определить режим работы.

Нам необходимо сделать две вещи:

- определить, является ли частота вращения выше, ниже или равной нулю, и установить режим для функции `run`;
- убрать знак из значения частоты вращения для `setSpeed`, чтобы оно всегда было положительным.

Режимом по умолчанию при нулевой частоте вращения будет `RELEASE`, т. е. остановка вращения. Если частота больше 0, режим будет `FORWARD`, а если меньше, то `BACKWARD`.

Чтобы получить правильную функцию `mode`, воспользуемся простым оператором `if`. Необходимо заменить метод `convert_speed` в классе, чтобы он возвращал режим и положительное значение скорости. Чтобы показать наличие двух частей функции, я добавил в код комментарии. Итак, внесите следующие изменения в `robot.py`:

```
def convert_speed(self, speed):
    # Выбор режима работы
    mode = Raspi_MotorHAT.RELEASE
    if speed > 0:
        mode = Raspi_MotorHAT.FORWARD
    elif speed < 0:
        mode = Raspi_MotorHAT.BACKWARD

    # Определение частоты вращения
    output_speed = (abs(speed) * 255) // 100
    return mode, int(output_speed)
```

Мы добавили в расчет частоты вращения еще одну операцию: `abs(speed)`. Результатом этой операции является абсолютное значение числа без знака. Например, как `-80`, так и `80` будут представлены как `80`, что означает, что метод всегда возвращает положительный результат.

Затем добавим несколько методов, которые позволят устанавливать частоту и направление вращения обоих двигателей напрямую. Они вызывают `convert_speed` и используют возвращенные им данные о режиме и частоте для вызова функций `Raspi_MotorHAT`.

Чтобы использовать преобразованное значение частоты вращения, необходимо изменить методы, отвечающие за работу двигателя.

```
def set_left(self, speed):
    mode, output_speed = self.convert_speed(speed)
    self.left_motor.setSpeed(output_speed)
    self.left_motor.run(mode)

def set_right(self, speed):
    mode, output_speed = self.convert_speed(speed)
    self.right_motor.setSpeed(output_speed)
    self.right_motor.run(mode)
```

Итак, для каждого двигателя мы получаем режим и выходную частоту вращения, затем вызываем `setSpeed` и `run`. На этом этапе файл `robot.py` должен выглядеть следующим образом:

```
from Raspi_MotorHAT import Raspi_MotorHAT
import atexit
class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Установка HAT-платы в соответствии с переданным адресом
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)
        # получение локальных переменных для каждого двигателя
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)
        # убедимся, что после завершения кода двигатели останавливаются
        atexit.register(self.stop_motors)
    def convert_speed(self, speed):
        # Выбор режима работы
        mode = Raspi_MotorHAT.RELEASE
        if speed > 0:
            mode = Raspi_MotorHAT.FORWARD
        elif speed < 0:
            mode = Raspi_MotorHAT.BACKWARD
        # Определение частоты вращения
        output_speed = (abs(speed) * 255) // 100
        return mode, int(output_speed)
```

```

def set_left(self, speed):
    mode, output_speed = self.convert_speed(speed)
    self.left_motor.setSpeed(output_speed)
    self.left_motor.run(mode)

def set_right(self, speed):
    mode, output_speed = self.convert_speed(speed)
    self.right_motor.setSpeed(output_speed)
    self.right_motor.run(mode)

def stop_motors(self):
    self.left_motor.run(Raspi_MotorHAT.RELEASE)
    self.right_motor.run(Raspi_MotorHAT.RELEASE)

```

Простой скрипт `behavior_line.py` теперь состоит всего из нескольких строк:

```

import robot
from time import sleep

r = robot.Robot()
r.set_left(80)
r.set_right(80)
sleep(1)

```

Мы упростили код, и это означает, что в дальнейшем его можно будет использовать для создания других сценариев. Для всех своих роботов я использую один интерфейс и разные версии объекта `Robot`. Я могу запустить код из файла `behavior_lines.py` на `ArmBot` (мы говорили о нем в главе 1) и других моих роботах. Все они могут двигаться вперед 1 с со скоростью, соответствующей 80 % от максимальной частоты вращения двигателя.

## РАЗРАБОТКА СЦЕНАРИЯ ДЛЯ СЛЕДОВАНИЯ ПО ЗАДАННОЙ ТРАЕКТОРИИ

Итак, мы переходим к созданию поведенческого сценария робота. На рис. 7.7 показан набросок пути, по которому наш робот будет следовать.

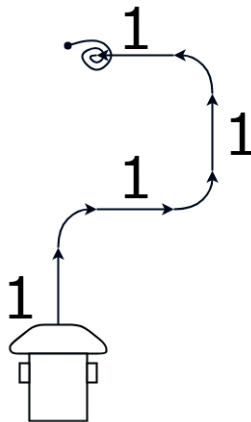


Рис. 7.7. Путь для робота

Прямые линии означают движение вперед; **единицы (1)** означают время – 1 с. На данном этапе мы не можем рассчитать пройденное расстояние, только время. Можно попробовать угадать расстояние, но такой подход не совсем точен. Изогнутые линии – это повороты, на которых частота вращения одного двигателя будет больше, чем другого.

Завиток в конце пути означает «победный» поворот на месте. Его мы реализуем путем запуска двигателей в разных направлениях (один будет вращаться вперед, а другой назад).

Давайте создадим код. Во-первых, нам нужны импорты: `sleep` и `robot`. Но, прежде чем приступить, давайте разработаем для этого сценария несколько вспомогательных функций. Я назвал свой файл `behavior_path.py` и поместил в него следующий код:

```
import robot
from time import sleep

def straight(bot, seconds):
    bot.set_left(80)
    bot.set_right(80)
    sleep(seconds)

def turn_left(bot, seconds):
    bot.set_left(20)
    bot.set_right(80)
    sleep(seconds)

def turn_right(bot, seconds):
    bot.set_left(80)
    bot.set_right(20)
    sleep(seconds)

def spin_left(bot, seconds):
    bot.set_left(-80)
    bot.set_right(80)
    sleep(seconds)
```

Вспомогательные функции пишем на том же языке, на котором написан основной скрипт. У нас есть следующие функции: `straight`, `turn_left`, `turn_right` и `spin_left`. Мы не включили их в объект `Robot`, потому что другие варианты поведенческих сценариев предполагают более продолжительные действия. Здесь вместо `Robot` я называю объект `bot`, поскольку имена переменных, состоящие из одной буквы, например `г`, гораздо труднее найти, прочесть и понять, когда код становится больше.

Каждая вспомогательная функция устанавливает частоту вращения двигателя, а затем бездействует определенное количество секунд. Затем мы создаем объект `Robot` и упорядочиваем упомянутые функции путем добавления в `behavior_path.py` следующего кода:

```
bot = robot.Robot()
straight(bot, 1)
turn_right(bot, 1)
straight(bot, 1)
turn_left(bot, 1)
straight(bot, 1)
```

```
turn_left(bot, 1)
straight(bot, 1)
spin_left(bot, 1)
```

Теперь можем загрузить его в Raspberry Pi и запустить через PuTTY посредством следующей команды:

```
$ python3 behavior_path.py
```

Теперь, если ваш робот похож на моего, вы увидите, как он передвигается и поворачивает. Повороты могут быть неточными, и робот может отклоняться от нужного угла. Мы можем исправить это, уменьшив значение времени для поворота:

```
bot = robot.Robot()
straight(bot, 1)
turn_right(bot, 0.6)
straight(bot, 1)
turn_left(bot, 0.6)
straight(bot, 1)
turn_left(bot, 0.6)
straight(bot, 1)
spin_left(bot, 1)
```

Чтобы робот мог осуществлять повороты на 90°, необходимо тщательно отрегулировать эти значения. Для этого понадобится немало терпения. Как только вы отрегулируете их, заново загрузите код в Pi. Регулирование значений в коде – это грубая форма калибровки, к которой мы прибегаем, чтобы соответствовать конструктивным особенностям нашего робота. Если робот будет перемещаться по разным поверхностям (например, перемещаться с деревянного пола на ковер), то значения времени будут меняться.

Вы можете учесть некоторые отклонения, сделав частоту вращения одного двигателя при движении вперед меньше (отрегулируйте значение для своего робота), например:

```
def straight(bot, seconds):
    bot.set_left(80)
    bot.set_right(70)
    sleep(seconds)
```

Этот код будет более-менее работоспособен, однако значения сложно подобрать. Но по какой причине?

Частота вращения двигателей (даже от одного производителя) может быть разной. На это может влиять диаметр колес, расположение осей, распределение веса, особенности поверхности (например, она может быть скользкой или неровной), сопротивление в проводах и колебания тока в контроллере двигателя. Таким образом, робот, скорее всего, не будет двигаться по идеальной прямой. В зависимости от сенсоров, которые мы используем, это может быть проблемой или же не иметь значения. В следующей главе мы поговорим об этом подробнее и рассмотрим энкодеры/датчики скорости, а также откалибруем их, чтобы сделать сценарий следования по заданной траектории более точным.

Без сенсоров робот не сможет определить, где он находится и не врезался ли он во что-нибудь. Если робот без сенсоров столкнется со стеной, вам придется подойти и переставить его на открытую поверхность.

## Выводы

В этой главе мы узнали, как установить библиотеки для платы двигателя, и продемонстрировали, как работают наши двигатели. Затем начали создавать первый уровень кода, поддерживающий выполнение поведенческих сценариев. Мы выяснили, что можем использовать этот уровень и для других роботов. Мы увидели, как наш робот движется по заданной траектории. При этом мы откалибровали некоторые значения в коде. Также мы узнали, какие проблемы при передвижении возникают у роботов без сенсоров.

Полученные знания вы можете применять для любого аппаратного проекта: сначала протестируйте двигатели / устройства вывода, затем создайте уровень поддержки поведенческих сценариев, чтобы в случае, если вы решите изменить аппаратную конструкцию, вам не пришлось менять весь код. В таком случае понадобится внести изменения только в код для двигателей.

В следующих главах мы рассмотрим сенсоры, на основе чего создадим новые поведенческие сценарии.

## Задание

Чтобы лучше разобраться в полученной информации, выполните следующие задания.

1. Нарисуйте еще одну траекторию и напишите код для робота, который будет по ней следовать. Траектория может быть, например, в форме восьмерки.
2. Какие методы вы бы добавили к объекту `Robot`, если бы у вас был дополнительный выход, которым нужно управлять (например, **светодиод**)?
3. Подумайте, что бы вы добавили в код объекта `Robot` для робота с системой рулевого управления, как у гоночного карта. Какие бы у него были методы? Код писать необязательно, вы можете представить его у себя в голове. Подсказка – скорее всего необходимо будет определять частоту вращения приводного двигателя (для движения) и положение двигателя рулевого управления.

## Дополнительные материалы

Узнать больше об объекте `Robot`, а также об использовании аналогичных интерфейсов и классов вы можете в книге «*Learning Object-Oriented Programming*», Гастон К. Хиллар (*Gastón C. Hillar*), *Packt Publishing*. Автор не только подробно излагает упомянутые концепции для языка Python, но и рассматривает их в более общем плане и показывает, как концепции **объектно-ориентированного программирования (ООП)** применяются в языках C# и JavaScript.

# Глава 8

## Код на Python для работы с датчиками расстояния

В этой главе мы рассмотрим, как датчики расстояния помогают роботу избегать столкновения с окружающими его объектами. Обход препятствий – крайне важная функция для мобильного робота, поскольку столкновения могут привести к его повреждению. К тому же, робот, который умеет это делать, кажется более разумным.

Здесь мы поговорим о различных типах датчиков и выберем тот, который подходит для нашего робота. Затем добавим уровень для доступа к ним через объект `Robot`, а также разработаем поведенческий скрипт, который позволит роботу избегать столкновения со стенами и другими препятствиями.

В этой главе мы рассмотрим следующие темы:

- выбор между оптическими и ультразвуковыми датчиками;
- подключение ультразвуковых датчиков и обработку получаемых ими данных;
- обход стен – разработку скрипта обхода препятствий.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Для выполнения практических экспериментов, описанных в этой главе, вам понадобятся:

- робот с Raspberry Pi и код, который мы создали в предыдущих главах;
- два ультразвуковых датчика HC-SR04P, RCWL-1601 или Adafruit 4007. Датчик должен быть рассчитан на напряжение питания 3,3 В;
- макетная плата;
- одножильный провод 22 AWG или набор жестких перемычек для макетной платы;
- ползунковый однополюсный выключатель, подходящий для установки на макетную плату;
- гибкие соединительные провода с разъемами «гнездо–штекер»;
- две крепежные опоры для датчика;
- крестовая отвертка;
- маленькие гаечные ключи или плоскогубцы.

Найти код на GitHub вы можете по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter8>.

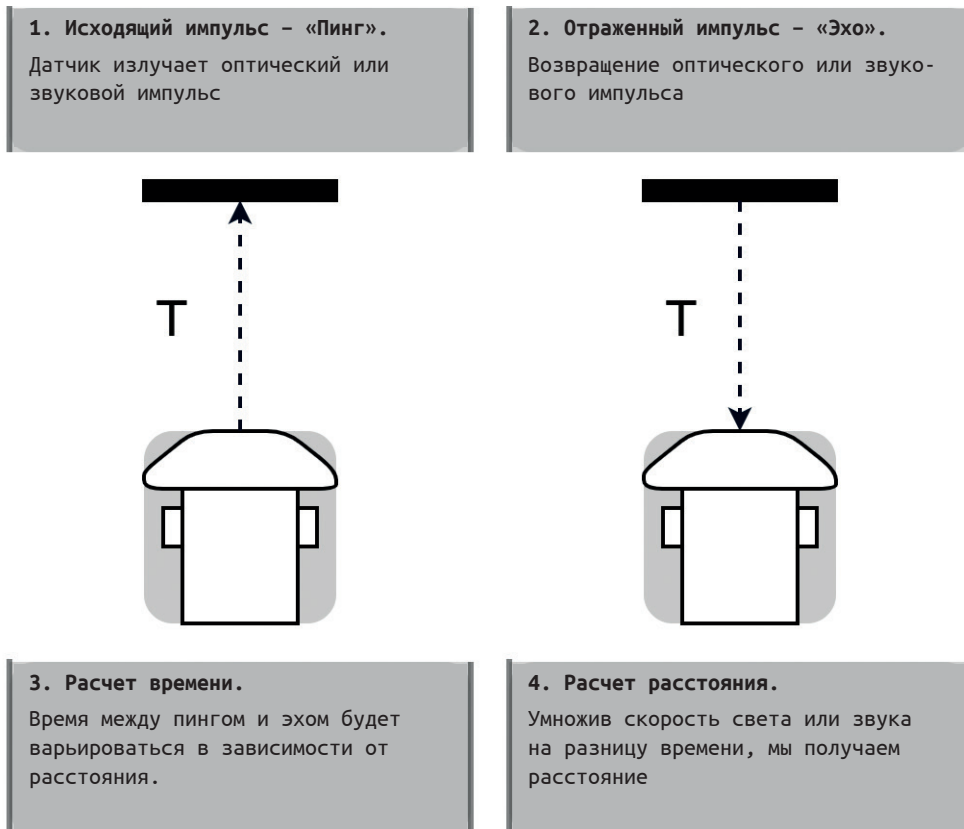


Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/2KfCkZM>.

## ВЫБОР МЕЖДУ ОПТИЧЕСКИМИ И УЛЬТРАЗВУКОВЫМИ ДАЛЬНОМЕРАМИ

Прежде чем перейти к работе с датчиками, необходимо выяснить, что они из себя представляют и как работают. Помимо этого, следует рассмотреть, какие типы датчиков существуют.

Датчики могут определять расстояние до объекта с помощью ультразвука или света. Оба типа работают по следующему принципу: датчик посылает импульсный сигнал и принимает его отражение от объекта. Затем путем измерения интервала времени между отправкой и получением импульса или основываясь на других пространственных измерениях (например, на измерении угла), датчик определяет расстояние до объекта, как показано на рис. 8.1.



**Рис. 8.1.** Определение датчиком расстояния на основе измерения времени, затраченного на прохождение сигнала до объекта и обратно

Нас интересуют дальномеры, которые определяют расстояние до объекта посредством измерения затраченного времени на прохождение сигнала до объекта и обратно (**времени пролета**). На рис. 8.1 принцип работы таких дальномеров показан более подробно.

Теперь, когда мы разобрались, как работают такие датчики, перейдем к детальному рассмотрению оптических и ультразвуковых дальномеров.

## ОПТИЧЕСКИЕ ДАЛЬНОМЕРЫ

Оптические дальномеры, подобные представленному на рис. 8.2, излучают инфракрасное лазерное излучение, невидимое для человека. Как правило, они небольшие по размеру. Такие дальномеры чувствительны к источникам интенсивного естественного и флуоресцентного освещения. Также на точность измерения влияет тип поверхности объекта; например, прозрачные или плохо отражающие свет объекты такой дальномер может не распознать.

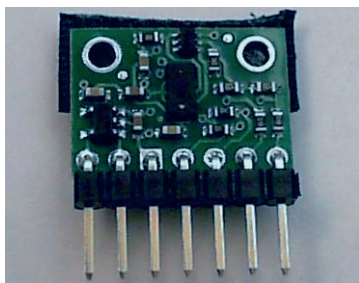


Рис. 8.2. Дальномер VL530L0x на плате модуля

На соревнованиях инфракрасные лучи финишных пунктов, определяющих время прохождения дистанции, могут мешать работе оптических дальномеров. Однако, в отличие от ультразвуковых, при размещении двух оптических дальномеров по разные стороны робота возможность возникновения ложных обнаружений значительно снижается. Они обеспечивают высокую точность измерения, но в ограниченном диапазоне расстояний. Как правило, стоят они дороже ультразвуковых, но можно найти и недорогие модели с фиксированным диапазоном.

## Ультразвуковые дальномеры

Многие звуковые дальномеры используют звуковые сигналы, частоты которых выходят за пределы восприятия человеческого уха. При этом они могут раздражать некоторых животных, например собак. Также эти сигналы могут улавливаться мобильными телефонами и некоторыми камерами, в результате чего возникают помехи («щелчки»). Ультразвуковые дальномеры, как правило, больше оптических, но при этом дешевле. Это обусловлено тем, что звук распространяется медленнее света и его скорость легче измерить. На точность результатов измерения ультразвуковых дальномеров влияет тип поверхности объекта, до которого измеряется расстояние. Например, они плохо обнаруживают мягкие (тканевые) объекты, поверхность которых не отражает звук.

На рис. 8.3 показан стандартный недорогой ультразвуковой дальномер HC-SR04.

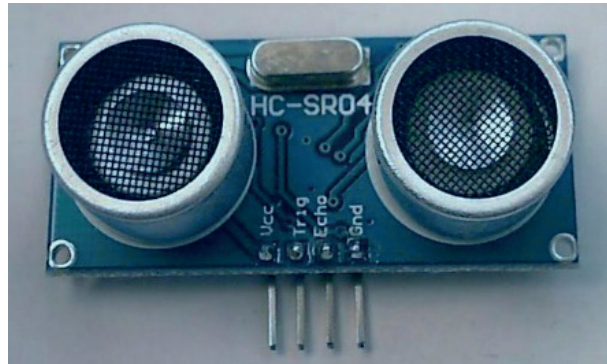


Рис. 8.3. HC-SR04

Диапазон измеряемых расстояний этого дальномера составляет от 2 до 4 м.

Помимо HC-SR04, существуют множество других ультразвуковых дальномеров, но не каждый из них подойдет для нашего проекта. Важным фактором при выборе дальномера являются его логические уровни, о чем мы поговорим в следующем разделе.

## Логические уровни и трансляторы логических уровней

Контакты ввода/вывода на Raspberry Pi работают с напряжением 3,3 В. Также в продаже есть множество устройств с рабочим напряжением (логикой) 5 В для входа или выхода. Давайте рассмотрим более детально, что такое логические уровни и почему важно не нарушать нагрузочные параметры.

**Напряжение** определяет величину воздействия на электроны, заставляющую их двигаться в потоке. Разные электронные устройства рассчитаны на разное напряжение. Слишком высокое напряжение может повредить устройство, а слишком низкого может быть недостаточно для работы датчиков и других подключенных устройств. Логические устройства принимают сигналы в форме высокого или низкого напряжения и относят их к истинному или ложному значению (логической 1 или логическому 0) соответственно. Истинное значение выше порогового значения, а ложное – ниже. Учитывать эти электрические свойства очень важно, поскольку в противном случае устройства либо не будут взаимодействовать, либо вовсе выйдут из строя.

На рис. 8.4 представлен график, содержащий информацию о логических уровнях.

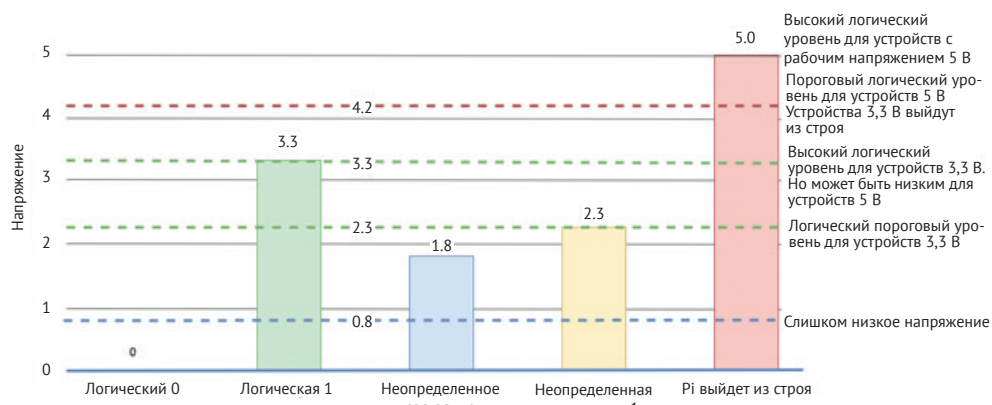


Рис. 8.4. Напряжение и логические уровни

На оси y (слева) отмечены уровни напряжения от 0 до 5 В. На оси x показаны различные рабочие условия. На графике проходят четыре пунктирные линии. Самая нижняя линия представляет уровень напряжения 0,8 В; все, что ниже этого значения, будет принято за логический 0. Следующая линия – напряжение около 2,3 В; многие устройства с рабочим напряжением 3,3 В будут принимать это значение за логическую 1. Линия 3,3 В представляет собой входное/выходное напряжение, которое Raspberry Pi будет принимать за логическую 1. При подаче большего напряжения Pi может выйти из строя. Линия, представляющая напряжение около 4,2 В, – этот тот уровень, который будет принят за логическую 1 устройствами с рабочим напряжением 5 В (хотя некоторые из них принимают за истинное напряжение от 2 В). Такие устройства нельзя подключать к Pi напрямую.

На графике вы можете видеть пять широких вертикальных полос. Первая находится на уровне 0; это означает, что отсутствие напряжения все устройства будут принимать как логический 0. Вторая полоса показывает, что уровень напряжения 3,3 В будет принят Raspberry Pi как четкая логическая 1. Однако для устройств с рабочим напряжением 5 В этого недостаточно, поскольку напряжение ниже 4,2 В. Полоса с подписью «неопределенное состояние» доходит до уровня 1,8 В. В этом случае неизвестно, к какому логическому уровню система отнесет такое значение, поэтому его лучше избегать. Полоса с подписью «**неопределенная логическая 1**» находится чуть выше порогового значения, поэтому здесь также нет гарантии того, как это значение воспримет устройство с рабочим напряжением 3,3 В. Последняя полоса доходит до уровня 5 В, это значение устройства с рабочим напряжением 5 В воспримут как истинное. Однако, подавать такое напряжение на Raspberry Pi без преобразователей уровня напряжения нельзя.

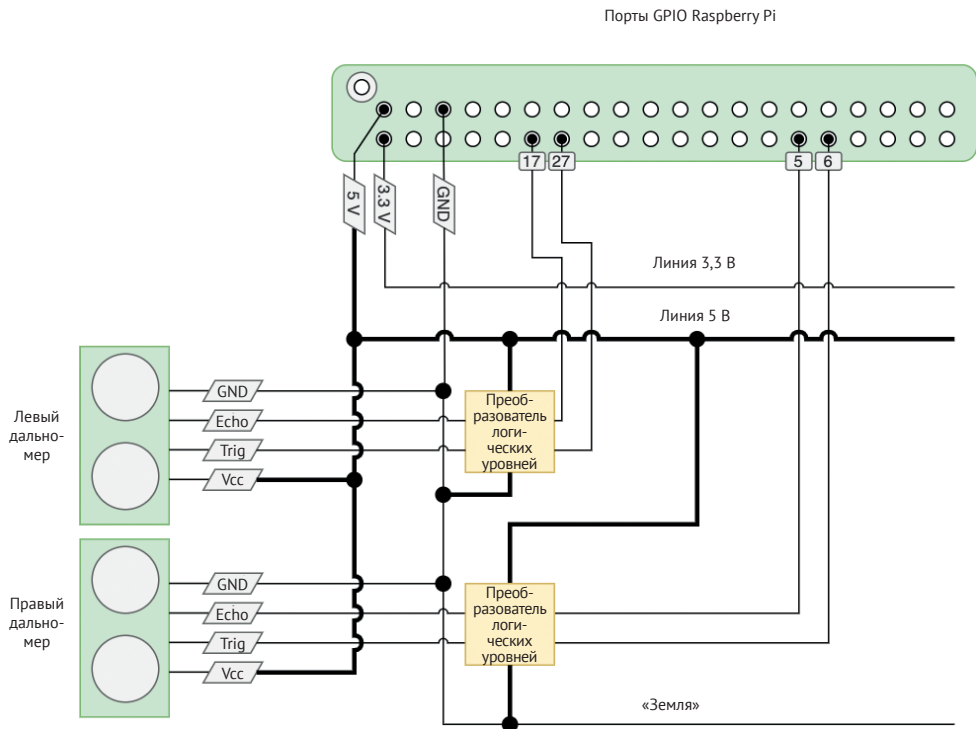
Как показано на рис. 8.4, значения 1,8 и 2,3 В очень близки к логическому порогу, поэтому логическое устройство может воспринимать их неоднозначно. Я рекомендую избегать промежуточных значений, например 1,5 В. Для нашего Pi напряжение должно быть около 3 В.

**Важное примечание**

Подача на контакты напряжения выше 3,3 В может вывести Raspberry Pi из строя. Для подключения к Pi устройств с рабочим напряжением 5 В требуется преобразователь логического уровня.

Для сопряжения с Pi устройств с рабочим напряжением 5 В требуются дополнительные электронные компоненты, которые поставляются на плате и нуждаются в соединительных проводах. Это значительно увеличивает стоимость, сложность и размер робота.

На рис. 8.5 показана принципиальная электрическая схема робота, к которому подключены датчики HC-SR04 с рабочим напряжением 5 В (ввиду чего требуются преобразователи логических уровней). Сверху на схеме вы можете видеть контакты GPIO Raspberry Pi. К трем контактам слева подключаются линии 5 В, 3,3 В (3v3) и «земля» (GND). Ниже параллельно порту GPIO проходят линии 3,3 и 5 В.



**Рис. 8.5.** Подключение датчиков HC-SR04 с помощью преобразователей логических уровней

Ниже линий питания расположены два преобразователя уровней. Они подключаются к контактам GPIO 5, 6, 17 и 27 (справа). На данной схеме соединения отмечены черными точками, а линии, которые не соединяются, – «мостиками».

Снизу на схеме изображена линия земли, отходящая от соответствующего контакта. У дополнительных электронных компонентов должна быть возможность подключения к этой линии.

Слева на схеме расположены два датсонаера, которые подключены к контактам 5V и GND. Каждый из них подключен к трансляторам логических уровней через контакты **trig** и **echo**. Как видите, подключение к Pi других датчиков через преобразователи логических уровней значительно усложняет принципиальную схему робота.

К счастью, у нас есть альтернативные варианты. По возможности следует выбирать устройства с рабочим напряжением 3,3 В или устройства, которые будут правильно воспринимать трехвольтовые логические уровни. Перед покупкой электронных компонентов следует уточнить рабочее напряжение главного контроллера робота (в нашем случае это Raspberry Pi) и убедиться, что выбранные компоненты подойдут.

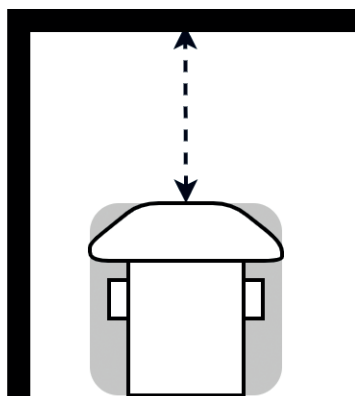
С HC-SR04 поставляются дополнительные компоненты, которые позволят подключить его к Pi. Модели HC-SR04P, RCWL-1601 и Adafruit 4007 с выходным напряжением 3,3 В можно подключать к Raspberry Pi напрямую.

## Зачем использовать два датчика?

Благодаря двум датсонаерам робот сможет определять, с какой стороны поблизости есть объект, а с какой есть свободное пространство для движения. На рис. 8.6 показано, как это работает.

### 1. Один датчик.

Робот может избежать столкновения со стеной впереди



### 2. Два датчика.

Робот может избежать столкновений со стенами слева и впереди

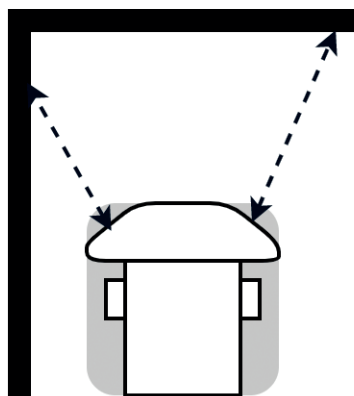


Рис. 8.6. Использование двух датчиков

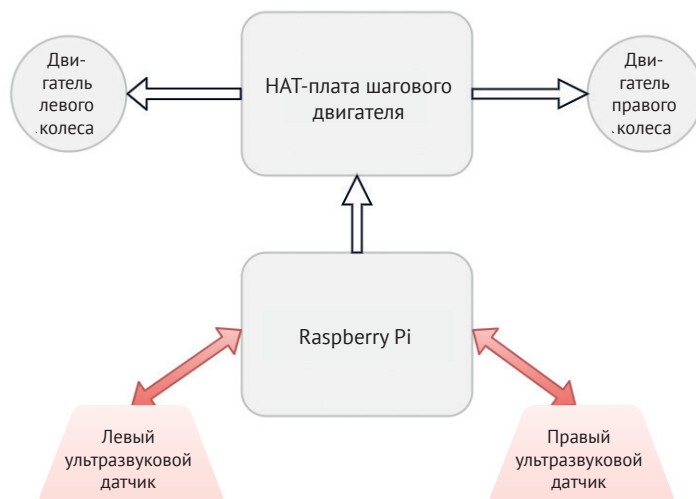
Робот, показанный на рис. 8.6 справа, получает больше данных об окружающем пространстве, поэтому он может принимать более интересные решения.

С учетом всего вышесказанного я рекомендую вам выбрать HC-SR04P/RCWL-1601 или Adafruit 4007 с рабочим напряжением 3,3 В. Их можно использовать сразу несколько, и при этом они недорогие.

Мы рассмотрели некоторые типы дальномеров и обсудили, какие из них подойдут для нашего проекта. Вы узнали больше об уровнях напряжения и почему важно их учитывать. Также мы определили, сколько датчиков можно использовать и где их лучше разместить. В следующем разделе поговорим о том, как их подключить.

## Подключение ультразвуковых дальномеров и обработка получаемых ими данных

Для начала мы установим дальномеры на шасси и подключим их. Затем создадим простой тестовый код, который позже будем использовать в качестве основы для разработки поведенческого скрипта. К концу этого раздела блок-схема робота должна будет выглядеть как на рис. 8.7.



**Рис. 8.7.** Блок-схема робота с ультразвуковыми дальномерами

Схема на рис. 8.7 – это та же схема, что и на рис. 6.33 (глава 6), но в ней появились ультразвуковые дальномеры слева и справа. Эти дальномеры являются активными датчиками и на схеме соединяются с Raspberry Pi двусторонними стрелками. Это означает, что сначала Pi запускает измерение, отправляя запрос на датчики, а затем обрабатывает полученные ими результаты. Давайте установим дальномеры на шасси робота.

### Установка датчиков на шасси

В разделе «Технические требования» я упоминал, что нам потребуется крепежная опора. Вы можете взять специальную опору для HC-SR04 или изготовить ее самостоятельно с помощью 3D-принтера, станка с ЧПУ или другим способом. Однако разумнее использовать уже готовый вариант. На рис. 8.8 показано, какие опоры использую я.

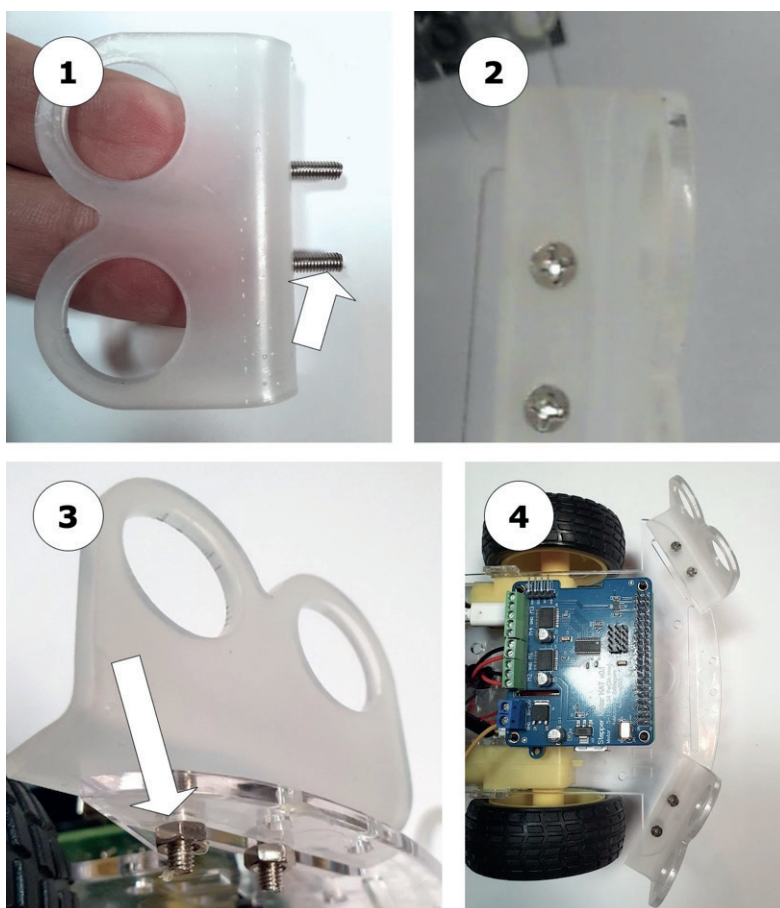




**Рис. 8.8.** Крепежные опоры и винты для ультразвукового датчика HC-SR04

Если ваше шасси похоже на мое, установить опоры не составит труда, поскольку в нем предусмотрены специальные крепежные отверстия (слоты).

На рис. 8.9 показана пошаговая инструкция по установке крепежных опор:



**Рис. 8.9.** Установка крепежных опор для датчика

1. Вставьте два винта в отверстия на опоре.
2. Затем вставьте их в отверстия на передней части шасси.
3. С нижней стороны наденьте на винты гайки и затяните. Повторите те же действия с другой стороны.
4. Робот с двумя опорами должен выглядеть так, как показано на этой части рисунка.

На рис. 8.10 показано, как установить датчики на опоры:

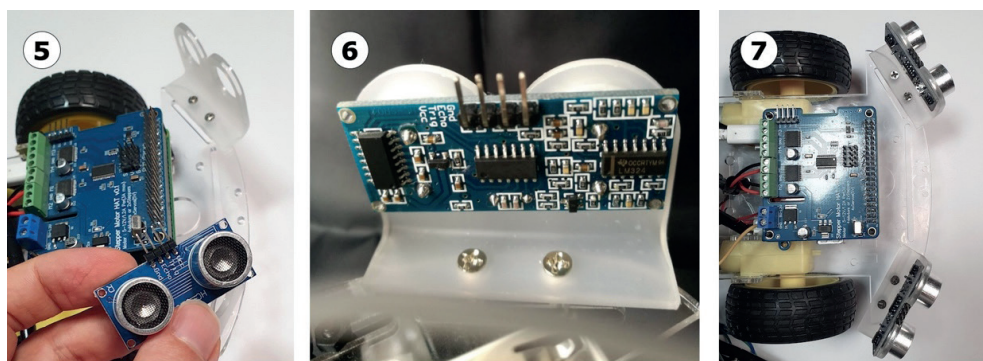


Рис. 8.10. Установка датчиков на опоры

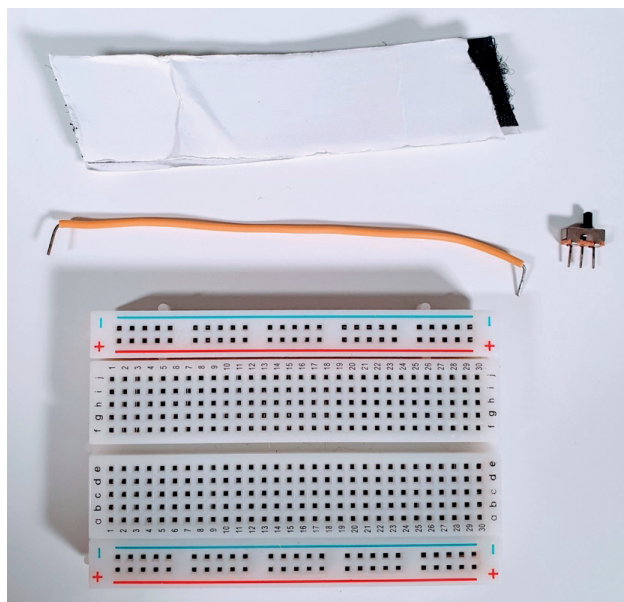
5. Взгляните на датчик и опору. В опоре есть отверстия для элементов излучателя и приемника (круглые, похожи на жестяные банки элементы с металлическими сеточками).
6. Дальномер можно просто вставить в опору, поскольку здесь предусмотрена плотная фрикционная посадка. Электрический разъем датчика должен быть направлен вверх.
7. После установки двух дальномеров робот должен выглядеть так, как показано на этой части рисунка.

Итак, вы установили датчики на шасси. Прежде чем подключить их, мы добавим еще один полезный компонент – выключатель питания.

## Подключение выключателя питания

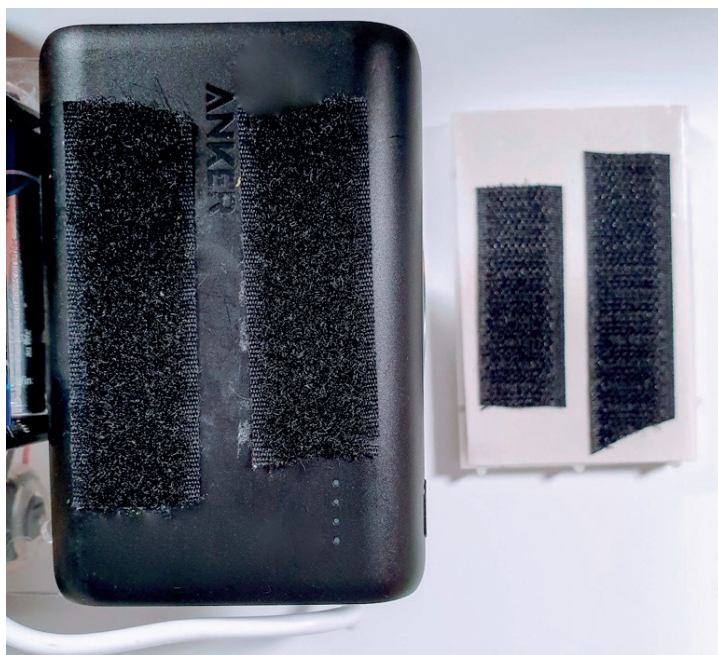
Прежде чем снова запустить робота, давайте добавим выключатель питания двигателя. Это будет намного удобнее, чем постоянно подключать/отключать провод заземления к винтовой клемме. Мы можем сделать это за три простых шага.

1. Убедитесь, что у вас есть все необходимое: макетная плата, «липучка», переключатель (подходящий для макетной платы) и одножильный провод 22 AWG (рис. 8.11).



**Рис. 8.11.** Все необходимое для установки выключателя питания

2. Закрепите макетную плату на аккумуляторе робота с помощью «липучки», как показано на рис. 8.12. «Липучка» будет надежно удерживать макетную плату, но при необходимости ее можно будет снять.



**Рис. 8.12.** Закрепляем макетную плату с помощью «липучки»

3. Теперь мы можем подключить выключатель к макетной плате (рис. 8.13).

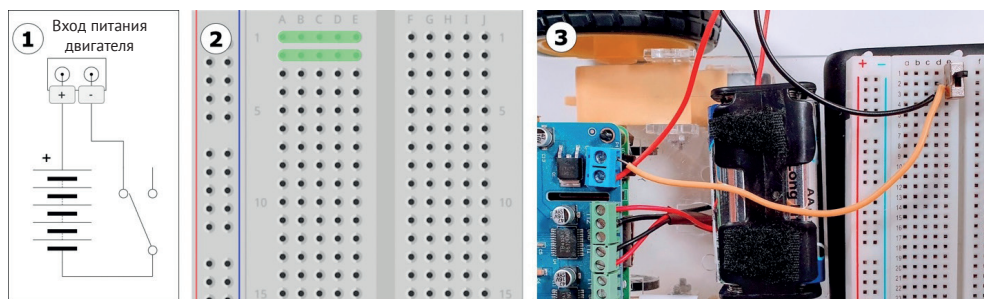


Рис. 8.13. Схема подключения выключателя

На рис. 8.13 изображена принципиальная схема, макетная плата крупным планом и возможный вариант реализации физического подключения. Давайте рассмотрим их подробнее.

1. На принципиальной схеме представлены батарейный блок, выключатель и входные разъемы питания двигателя. Сверху находится *клемма питания двигателя*. От положительного вывода (+) этой клеммы (слева) к батарейному блоку (показан на схеме в виде чередующихся толстых и тонких полос) идет провод. Снизу на батарейном блоке находится отрицательный вывод (–). Из него идет провод к выключателю (справа на схеме). Затем выключатель соединяется проводом с отрицательным выводом *клеммы питания двигателя*. Вы можете подключить компоненты, руководствуясь этой схемой.
2. Прежде чем мы подключим выключатель, стоит обсудить макетную плату. На макетной плате, которую вы видите на этой части рисунка, два ряда отверстий выделены зеленым. Это значит, что каждый из них содержит по 5 монтажных отверстий (точек подключения), соединенных между собой. На плате есть две группы точек, в каждой из которых 30 рядов (пронумерованы от 1 до 30), а в каждом ряду по 5 точек. Группы разделяются пазом в середине платы.
3. Макетная плата – это инструмент для реализации физического подключения, который позволяет соединять провода и устройства. Скорее всего, подключение на макетной плате не будет полностью визуально соответствовать схеме. На этой части рисунка слева вы можете видеть плату двигателя, к которой идет красный провод от батарейного блока. Он обозначает положительную фазу и подключается к положительному выводу (+ или VIN) *клеммы питания двигателя*. В центре находится батарейный блок. От батарейного блока к макетной плате идет черный провод, который примыкает к точке в ряду 3, столбец *d*. Выключатель подключен к точкам в 1-м, 2-м и 3-м рядах в столбце *e*. От точки в ряду 2 отходит предварительно укороченный оранжевый провод 22 AWG, который подключается к винтовой клемме GND. С помощью выключателя мы сможем включать/выключать питание двигателей робота.



Итак, мы оснастили нашего робота выключателем для включения/выключения питания двигателей, поэтому теперь для этого нам не понадобится отвертка. Далее будем использовать эту же макетную плату для подключения датчиков.

## Подключение датчиков

У любого ультразвукового датчика есть четыре контакта:

- контакт управления (trig), предназначенный для запуска процесса измерения расстояния;
- контакт выхода (echo), предназначенный для выдачи отраженного от объекта импульса;
- VCC/контакт напряжения питания +3,3 В;
- GND/контакт «земли».

Прежде чем мы продолжим, убедитесь, что ваш робот выключен. Контакты **trig** и **echo** нужно подключить к порту GPIO на Raspberry Pi.

Воспользуйтесь схемой расположения контактов GPIO, представленной крупным планом на рис. 8.14, чтобы упростить задачу.

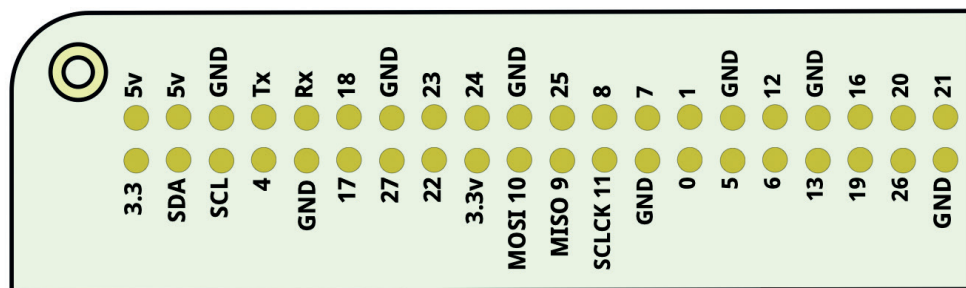
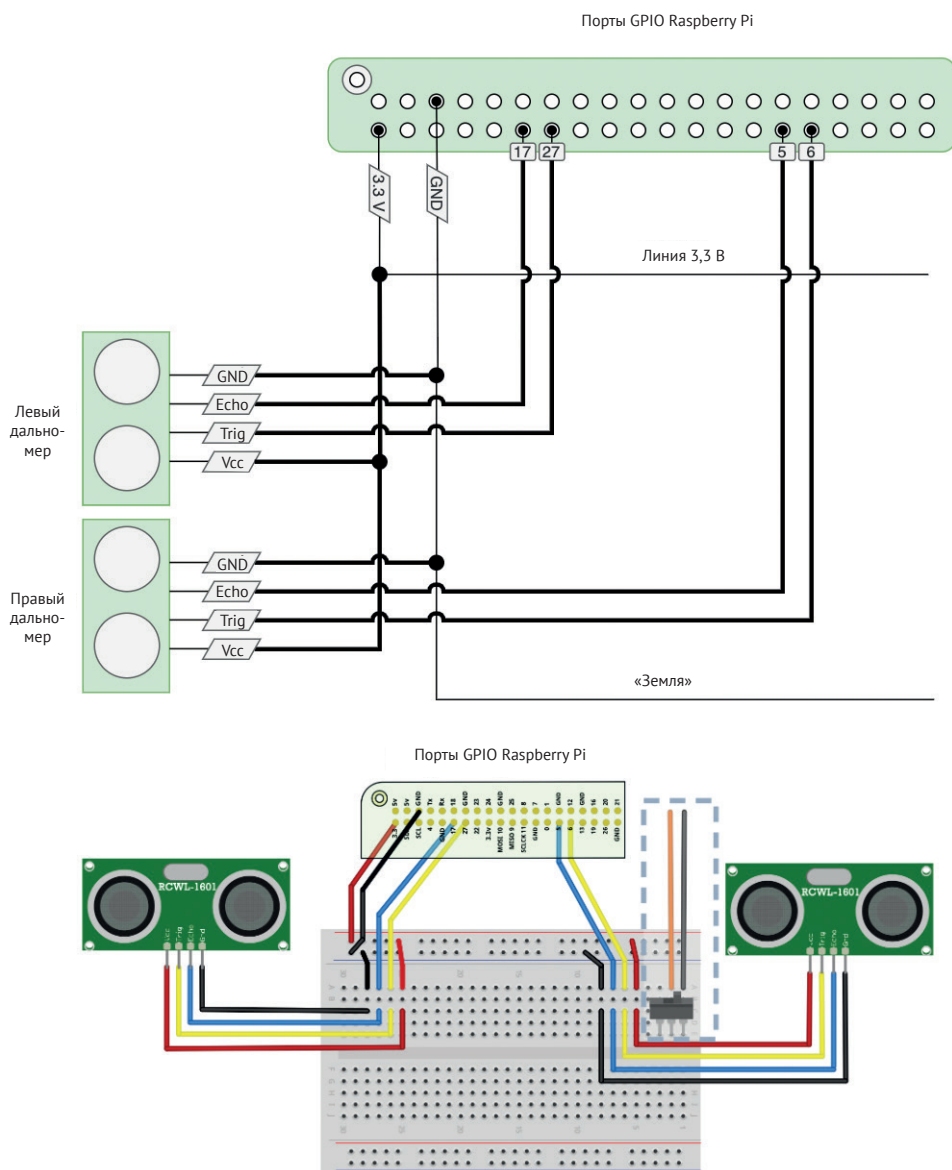


Рис. 8.14. Контакты Raspberry Pi

На рис. 8.14 представлена схема порта GPIO на Raspberry Pi. Он находится в верхней части Pi и имеет 40 контактов, расположенных в два ряда. К этому порту подключаются многие устройства. Как правило, на нем нет номеров/имен контактов, но они подписаны на нашей схеме.

Для того чтобы подключить датчики к GPIO, потребуется макетная плата. На рис. 8.15 показана подробная схема подключения.

Подключать Raspberry Pi и датчики к макетной плате следует с помощью перемычек «гнездо–штекер». Для самой макетной платы можно использовать перемычки из предварительно нарезанного провода (всего их должно быть 4). На рис. 8.15 сверху показана принципиальная электрическая схема, а снизу – вариант подключения проводов на макетной плате.



**Рис. 8.15.** Схема подключения датчиков

Для того чтобы подключить датчики, выполните следующие шаги руководства рис. 8.15.

1. Начнем с подключения питания. Провод должен идти от контакта 3,3V порта GPIO Raspberry Pi (на схемах часто обозначается как 3v3) до красной шины питания в верхней части макетной платы. Мы можем использовать эту шину для других устройств, которым необходимо питание 3,3 В.

2. Провод от одного из контактов GND должен идти к шине, отмеченной на макетной плате черным или синим цветом. Эту шину можно использовать для всех соединений, которым требуется «земля».
3. Возьмите 4 соединительных провода-перемычки, у которых с одной стороны установлена штекерная часть разъема, а с другой – гнездовая.
4. Найдите четыре контакта (VCC, trig, echo и GND) у левого датчика. Для подключения датчика к макетной плате понадобятся четыре провода типа «гнездо–штекер». Подключите с их помощью датчик к макетной плате.
5. На макетной плате с помощью проводов из комплекта подключите «землю» к синей шине, а VCC к красной.
6. Теперь с помощью проводов-перемычек реализуйте сигнальные соединения от контактов trig/echo к контактам порта GPIO Raspberry Pi.

### Важное примечание

Если вы расположили макетную плату слишком далеко, длины проводов для подключения датчика может не хватить. В этом случае последовательно соедините два провода-перемычки и закрепите соединение изоляционной лентой<sup>6</sup>.

Чтобы все выглядело аккуратно, вы можете закрутить провода в виде спирали, но это не обязательно.

Прежде чем продолжить, проверьте все подключения еще раз<sup>7</sup>. Теперь, когда мы добавили в конструкцию робота дальномеры, необходимо подготовить программные компоненты для их тестирования.

## Установка библиотек Python для работы с датчиками

Для взаимодействия через GPIO с датчиками и другими аппаратными компонентами необходима библиотека Python. Давайте воспользуемся библиотекой GPIOZero, предназначенной как раз для работы с подобными устройствами.

```
$ pip3 install RPi.GPIO gpiozero
```

Теперь, когда библиотека установлена, мы можем создать тестовый код.

## Считывание данных ультразвуковых датчиков расстояния

Разработка кода для взаимодействия с датчиками позволяет понять, как они работают. Мы уже знаем, что ультразвуковые дальномеры работают по следующему принципу: они посылают звуковой импульсный сигнал, а затем принимают его отражение от объекта. При этом они измеряют интервал времени между отправкой и получением импульса, на основе чего определяют расстояние до объекта.

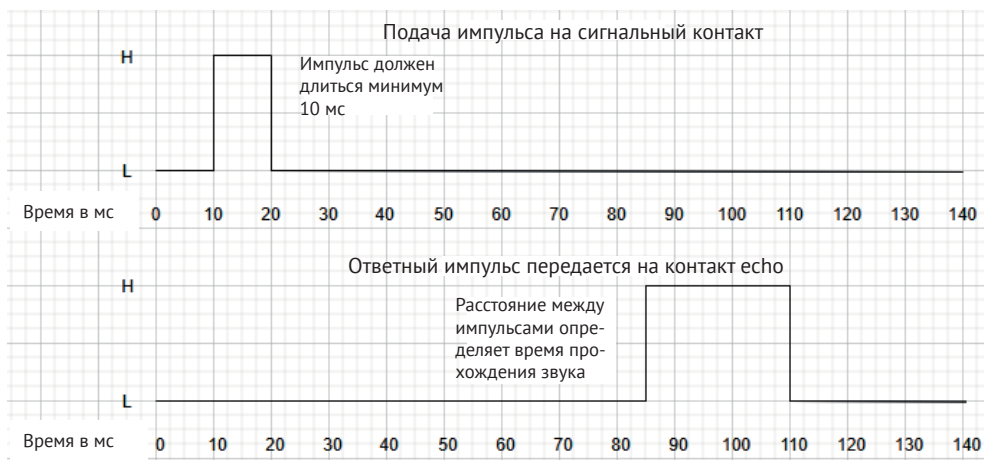
<sup>6</sup> Лучше всего заранее купить наборы проводов разной длины. Они обязательно пригодятся вам в будущем. – *Прим. ред.*

<sup>7</sup> Особенно полярность подключения питания. – *Прим. ред.*



Код на Raspberry Pi отправляет электронный импульс на **сигнальный контакт (trig)**, чтобы запустить измерение расстояния до объекта. После этого датчик посылает звуковой импульс и замеряет время, затраченное на его прохождение до объекта и обратно. На **контакте echo** в ответ формируется импульс в момент приема отраженного сигнала. Интервал времени между импульсами соответствует времени прохождения звукового сигнала.

На рис. 8.16 показано время, за которое все это происходит.



**Рис. 8.16.** Диаграмма импульса запуска измерения и выходного импульса датчика

Библиотека `GPIOZero` может измерить интервал времени между импульсами и преобразовать его в расстояние. Это значение мы можем использовать в коде.

При задержке отражения звука датчик может не получить ответ вовремя. Такая задержка возникает, когда объект находится за пределами рабочего диапазона датчика или когда что-то поглощает звуковой импульс.

Как и в случае с классом для управления двигателями, мы будем использовать комментарии и описательные имена, чтобы лучше разобраться в этой части кода. Я назвал этот файл `test_distance_sensors.py`.

1. Начнем с импорта библиотек `time` и `DistanceSensor`:

```
import time
from gpiozero import DistanceSensor
```

2. Далее настраиваем датчики. Для того чтобы показать, что здесь происходит, я использовал функции `print`. В следующих строках мы создаем объекты библиотеки для каждого датчика, регистрируя контакты, к которым мы их подключили. Убедитесь, что все номера соответствуют реальным подключениям (номерам контактов на плате):

```
print("Prepare GPIO Pins")
sensor_l = DistanceSensor(echo=17, trigger=27, queue_
len=2)
sensor_r = DistanceSensor(echo=5, trigger=6, queue_
len=2)
```

Обратите внимание на дополнительный параметр `queue_len`. Перед выводом на печать библиотека `GPIOZero` может собрать и усреднить показания 30 измерений, что делает ее вывод более точным, но менее быстрым. Поскольку наш робот должен обладать быстрой реакцией, мы будем считывать только по два измерения. Это менее точно, зато намного быстрее.

3. Затем тестовый код зацикливается и работает, пока мы его не остановим:

**while True:**

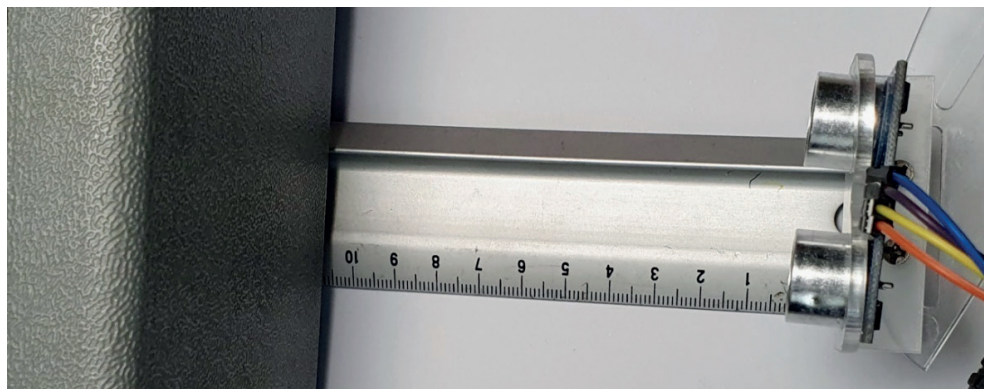
4. Затем мы выводим на печать значения расстояний, измеренных нашими датчиками. Свойство `.distance` похоже на то, которое мы видели ранее в светодиодной системе, — `.count`. Значение этого свойства постоянно обновляется датчиками. Мы умножим значение на 100, поскольку расстояние в `GPIOZero` выражается в метрах:

```
print("Left: {l}, Right: {r}".format(
    l=sensor_l.distance * 100,
    r=sensor_r.distance * 100)
```

5. При помощи небольшой задержки между проходами цикла можно предотвратить переполнение буфера текстового вывода:

**time.sleep(0.1)**

6. Теперь вы можете включить Raspberry Pi и загрузить код.
7. Поместите объект на расстоянии от 4 см до 1 м от датномера, как показано на рис. 8.17.



**Рис. 8.17.** Дальномер и объект

На рис. 8.17 объект находится примерно на расстоянии 10,5 см от датчика. В качестве объекта выступает небольшой ящик для инструментов. Вы можете выбрать любой объект с твердой поверхностью.

8. Запустите код на Pi, введя команду `python3 test_distance_sensors.py`. При перемещении робота поблизости от объекта Pi должен начать выводить значения расстояния:

```
pi@myrobot:~ $ python3 test_distance_sensors.py
Prepare GPIO Pins
Left: 6.565688483970461, Right: 10.483658125707734
Left: 5.200715097982538, Right: 11.58136928065528
```

9. Поскольку код заиклен, нажмите **Ctrl+C** для остановки выполнения программы.
10. Вы увидите числа с большим количеством знаков после запятой, что не совсем удобно. Во-первых, наши датчики вряд ли будут настолько точными, а во-вторых, для принятия решений нашему роботу не требуется точность меньше сантиметра. Мы можем слегка изменить оператор `print` в цикле:

```
print("Left: {:.2f}, Right: {:.2f}".format(
    l=sensor_l.distance * 100,
    r=sensor_r.distance * 100))
```

Оператор форматирования : `.2f` позволяет округлить значения до чисел с двумя знаками после запятой. Отладочные выходные данные помогают увидеть, что происходит в работе, поэтому очень полезно знать, как можно их скорректировать.

11. При запуске обновленного кода вы увидите примерно следующее:

```
pi@myrobot:~ $ python3 test_distance_sensors.py
Prepare GPIO Pins
Left: 6.56, Right: 10.48
Left: 5.20, Right: 11.58
```

Итак, мы убедились, что датчики работают. Также мы изучили, как можно скорректировать выходные данные датчиков для отладки (с ней в робототехнике вы будете встречаться очень часто). Чтобы убедиться, что вы на правильном пути, давайте рассмотрим, как устранить возможные неполадки.

## Устранение неполадок

Если датчики работают не так, как ожидалось, попробуйте предпринять следующие действия:

- провода перегреваются? Пощупайте провода, ведущие к датчику. *Они не должны быть горячими или даже слегка теплыми!* Если вы почувствовали нагрев, немедленно извлеките батарейки, выключите Raspberry Pi и тщательно проверьте все соединения на соответствие схеме, изображенной на рис. 8.15 ;
- при возникновении ошибок синтаксиса сравните ваш код с примерами. Библиотеки Python должны быть установлены с помощью `pip3`, а код запущен командой `python3`<sup>8</sup>;
- если вы по-прежнему видите предупреждения об ошибке или недопустимых значениях, проверьте правильность кода и его структуры;
- если значения всегда равны 0 или вы не получаете данные от датчика, возможно, вы перепутали контакты **trig** и **echo**. Попробуйте по-

<sup>8</sup> Мы используем именно третью версию Python. – Прим. ред.

менять местами их номера в коде и проверьте еще раз. *Не отключайте и не меняйте местами* провода на работающем Pi! Вносите изменения для одного датчика за раз;

- если вы по-прежнему не получаете данные, убедитесь, что вы приобрели датчики с рабочим напряжением 3,3 В. Модель HC-SR04 не будет работать с Raspberry Pi без преобразователя логического уровня;
- если вы видите странные или непостоянные значения, убедитесь, что объект, расстояние до которого вы измеряете, твердый. Мягкую поверхность, например одежду, шторы или вашу руку, дальномеру заметить сложнее, чем стекло, дерево, металл или пластик. Хорошим вариантом будет измерение расстояния до стены;
- еще одной причиной некорректных значений может быть слишком маленькая площадь поверхности. Убедитесь, что поверхность достаточно широкая. Если она меньше, чем квадрат со сторонами 5 см, то расстояние до нее будет сложнее измерить;
- если один датчик работает нормально, а другой нет, возможно он неисправен. Попробуйте поменять датчики местами. Если что-то изменилось, значит проблема в датчике. Если результат тот же, то проблема в проводке или коде.

Теперь мы устранили неполадки и убедились, что дальномеры работают. Мы увидели, что они могут выводить значения, и протестировали измерение расстояния до объекта, чтобы убедиться, что дальномеры могут отправлять данные. Теперь давайте перейдем к написанию сценария обхода препятствий.

## ОБХОД СТЕН – РАЗРАБОТКА СЦЕНАРИЯ ОБХОДА ПРЕПЯТСТВИЙ

Теперь, когда мы протестировали дальномеры, можем интегрировать их в наш класс `Robot` и разработать для них поведенческий скрипт обхода препятствий. Этот поведенческий цикл будет обрабатывать данные дальномеров, а затем принимать поведенческое решение на основе полученной информации.

### Добавление датчиков в класс `Robot`

Прежде чем робот сможет принимать поведенческие решения на основе данных дальномеров, сами датчики необходимо добавить в класс `Robot`, указав номера контактов GPIO для каждого датчика. Таким образом, при изменении номеров контактов или даже интерфейса датчика поведенческий скрипт не придется менять полностью.

1. Чтобы использовать объект `DistanceSensor`, нам нужно импортировать его из `gpiozero`; изменения в коде выделены жирным шрифтом:

```
from RPi_MotorHAT import RPi_MotorHAT
from gpiozero import DistanceSensor
```

2. Мы создаем по одному экземпляру объекта `DistanceSensor` для каждого дальномера в классе `Robot`. Нужно настроить их в конструкторе классов. Мы используем те же номера контактов и длину очереди, что и в нашем тестовом коде:

```
class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Setup the motorhat with the passed in address
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)

        # получение локальных переменных для каждого двигателя
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)

        # настройка датчиков расстояния
        self.left_distance_sensor =
DistanceSensor(echo=17, trigger=27, queue_len=2)
        self.right_distance_sensor =
DistanceSensor(echo=5, trigger=6, queue_len=2)

        # убедимся, что после завершения кода двигатели останавливаются
        atexit.register(self.stop_all)
```

Добавив этот фрагмент кода в объект `Robot`, можем создавать поведенческие сценарии. Когда мы создадим нашего робота, датчики будут измерять расстояния. Давайте разработаем поведенческий скрипт, который будет использовать датчики.

## Разработка сценария обхода препятствий

Эта глава посвящена поведению роботов; как робот может передвигаться и обходить препятствия (точнее, большинство из них)? Технические характеристики датчиков не позволяют точно измерять расстояние до мелких объектов или объектов с мягкой/ворсистой поверхностью, например предметов с тканевой обивкой. На рис. 8.18 показано, что я имею в виду.

Простой робот в нашем примере (рис. 8.18) обнаруживает стену, после чего выполняет поворот в сторону и продолжает движение до тех пор, пока не обнаружит другую стену, после чего опять выполняет поворот. Мы можем реализовать нечто подобное в рамках нашей первой попытки разработки стратегии обхода препятствий.

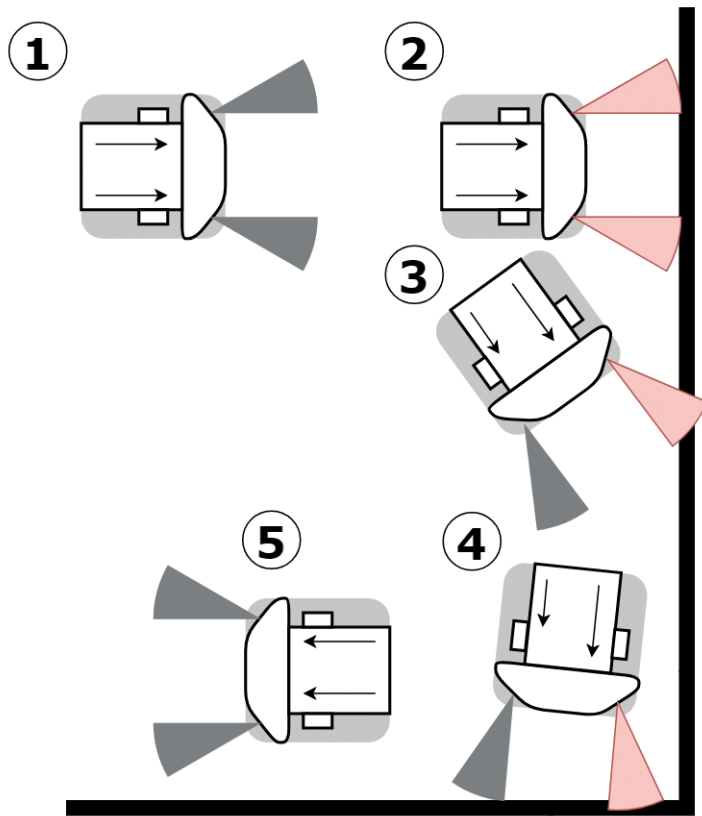


Рис. 8.18. Обход препятствий

## Первая попытка разработки стратегии обхода препятствий

Чтобы разобраться в том, как реализовать обход препятствий, внимательно рассмотрите блок-схему поведенческого сценария на рис. 8.19 (сверху вниз).

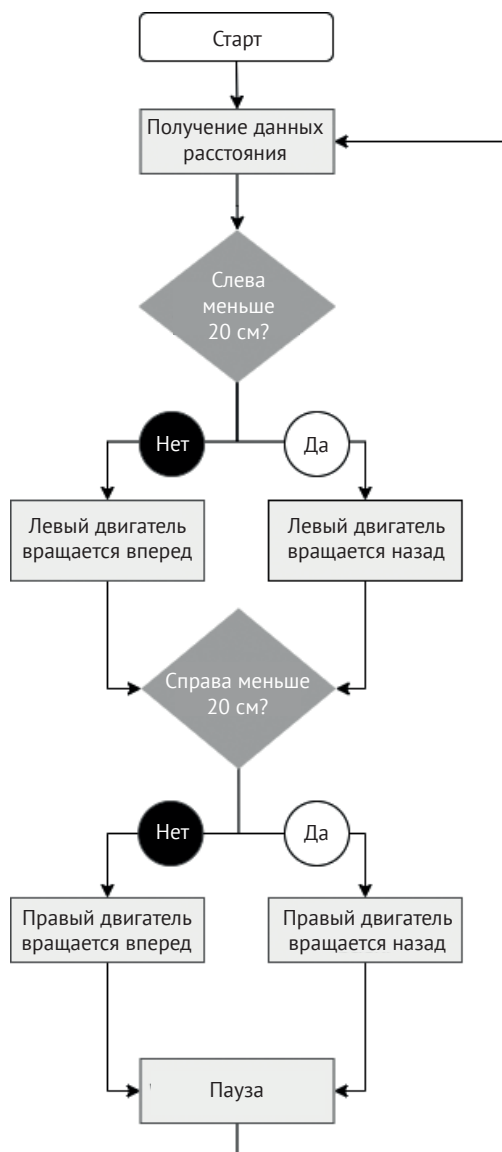


Рис. 8.19. Блок-схема обхода препятствий

Эта схема описывает следующий цикл.

1. Блок **Старт** переходит в блок **Получение данных расстояния**, где робот получает данные о расстоянии от каждого датчика.
2. Сначала мы ориентируемся на результат измерения левого датчика. Расстояние до объекта составляет больше или меньше 20 см (допустимое расстояние для продолжения движения)?



- а) Если меньше, необходимо, чтобы левый двигатель начал вращаться в обратном направлении, благодаря чему робот осуществит поворот и обойдет препятствие.
  - б) Если больше, направление вращения не изменится.
3. Теперь мы проверяем правый датчик. Согласно его данным, расстояние до объекта составляет больше или меньше 20 см? Если больше, то не менять направление вращения.
  4. Затем программа делает паузу, и цикл начинается заново.

Мы помещаем этот цикл в метод `run`. В связи с этим потребуется небольшая доработка. Нам необходимо установить значения поворота и наклона равными 0, чтобы они не мешали датчикам. Нужно добавить в `simple_avoid_behavior.py` следующие фрагменты кода.

1. Начнем с импорта объекта `Robot` и библиотеки `sleep` для работы со временем:

```
from robot import Robot
from time import sleep
...
```

2. Следующий класс – это основа поведенческого скрипта. В этом скрипте хранится объект `Robot`. Также здесь устанавливается частота вращения двигателей, которую можно регулировать, чтобы робот двигался быстрее или медленнее. Чем быстрее движется робот, тем меньше у него времени на принятие решения:

```
...
class ObstacleAvoidingBehavior:
    """Простой обход препятствий"""
    def __init__(self, the_robot):
        self.robot = the_robot
        self.speed = 60
    ...
```

3. Следующий метод выбирает частоту вращения для каждого двигателя в зависимости от данных, полученных датчиком. Если, исходя из этих данных, объект находится близко, двигатель будет вращаться в другую сторону:

```
...
def get_motor_speed(self, distance):
    """Этот метод вычисляет частоту вращения двигателя на основе
    данных о расстоянии, полученных от датчика"""
    if distance < 0.2:
        return -self.speed
    else:
        return self.speed
    ...
```

4. Метод `run` является ключевым, поскольку он контролирует основной цикл. Значения поворота и наклона мы помещаем посередине, чтобы они не мешали опрашивать датчики:

```
...
def run(self):
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
```

5. Теперь запускаем главный цикл:

```
while True:
    # Получение показаний от датчика в метрах
    left_distance = self.robot.left_distance_sensor.distance
    right_distance = self.robot.right_distance_sensor.distance
    ...
```

6. Затем мы выводим наши показания на консоль с помощью функции print:

```
...
print("Left: {l:.2f}, Right: {r:.2f}".
      format(l=left_distance, r=right_distance))
...
```

7. Теперь на основе данных о расстоянии отправляем каждому двигателю команду с помощью метода get\_motor\_speed:

```
...
# Вычисление частоты вращения двигателя на основе данных расстояния
left_speed = self.get_motor_speed(left_distance)
self.robot.set_left(left_speed)
right_speed = self.get_motor_speed(right_distance)
self.robot.set_right(right_speed)
```

8. Поскольку это наш основной цикл, нужно сделать паузу перед тем, как повторить его. Ниже показаны команды ожидания и запуска поведенческого скрипта:

```
...
# Делаем паузу
sleep(0.05)

bot = Robot()
behavior = ObstacleAvoidingBehavior(bot)
behavior.run()
```

Мы создали поведенческий скрипт, и он готов к запуску. Пришло время его проверить. Для этого поместите робота в пространство шириной несколько квадратных метров. Вокруг не должно быть мягких (например, мягкая мебель) или узких (например, ножки стула) объектов. В качестве препятствий я использовал папки и пластиковые ящики для игрушек.

Загрузите код в контроллер робота и протестируйте его. Робот должен двигаться прямо, пока не обнаружит препятствие, после чего он должен разворачиваться в другую сторону. Все должно работать. Однако, даже если вы настроите частоту вращения и другие параметры, робот все равно будет застревать в углах.

Пришло время разработать более сложную стратегию.

## Разработка более сложной стратегии обхода препятствий

Предыдущий простой сценарий может приводить к тому, что робот будет застревать в некоторых местах. Иногда робот будет не в состоянии принять решение относительно каких-либо препятствий или может сталкиваться с ними. Иногда он может не останавливаться вовремя и врезаться в объекты. Давайте разработаем сценарий, который позволит нашему роботу двигаться более плавно.

Итак, какова наша стратегия? Давайте исходить из расположения наших датчиков относительно препятствия. Мы можем вычислить частоту вращения двигателя, который располагается ближе к препятствию, и двигателя, который дальше от препятствия, а также временную задержку. Временная задержка дает роботу время на принятие решения, а временной фактор отвечает, на каком расстоянии от объекта необходимо повернуть. Все это позволяет роботу принимать решения более эффективно. Для того чтобы реализовать вышеописанное, внесем некоторые изменения в наш поведенческий скрипт.

1. Сначала скопируйте код из файла `simple_avoid_behavior.py` в новый файл с именем `escape_behavior.py`.
2. Функция `get_motor_speed` нам не понадобится, поэтому ее можно удалить. Мы заменим ее функцией `get_speeds` с параметром `near_distance` который всегда соотносится с датчиком расстояния, от которого мы получаем меньшее показание расстояния:

```
...
def get_speeds(self, nearest_distance):
    if nearest_distance >= 1.0:
        nearest_speed = self.speed
        furthest_speed = self.speed
        delay = 100
    elif nearest_distance > 0.5:
        nearest_speed = self.speed
        furthest_speed = self.speed * 0.8
        delay = 100
    elif nearest_distance > 0.2:
        nearest_speed = self.speed
        furthest_speed = self.speed * 0.6
        delay = 100
    elif nearest_distance > 0.1:
        nearest_speed = -self.speed * 0.4
        furthest_speed = -self.speed
        delay = 100
    else: #столкновение
        nearest_speed = -self.speed
        furthest_speed = -self.speed
        delay = 250
    return nearest_speed, furthest_speed, delay
...
```

Все эти значения должны быть тщательно настроены. Идея заключается в том, что по мере приближения к препятствию мы снижаем частоту вращения двигателя, а если робот подходит слишком близко к объекту, двигатель начинает вращаться в обратную сторону. Основываясь на значении временной задержки и зная, какой двигатель находится ближе, а какой дальше, мы можем управлять нашим роботом эффективнее.

3. Большую часть кода оставим без изменений. Например, ниже показана функция `run`, которую вы уже видели:

```
...
def run(self):
    # Движение вперед
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
    while True:
        # Получение показаний от датчика в метрах
        left_distance = self.robot.left_distance_sensor.distance
        right_distance = self.robot.right_distance_sensor.distance
        # Отображение
        self.display_state(left_distance, right_distance)
    ...
```

4. Теперь функция использует метод `get_speeds` для определения, с какой стороны расстояние ближе, а с какой дальше. Обратите внимание, что из двух значений нас интересует минимальное (`min`). Мы получаем значения частоты вращения обоих двигателей и задержки, а затем используем `print`:

```
...
# Вычисляем значения частоты вращения двигателей исходя из расстояния
nearest_speed, furthest_speed, delay = self.
get_speeds(min(left_distance, right_distance))
print(f"Distances: l {left_distance:.2f},
r {right_distance:.2f}. Speeds: n: {nearest_speed}, f:
{furthest_speed}. Delay: {delay}")
...
```

Здесь мы используем *f-строку* (сокращение от `.format`), с которой встречались ранее. Размещение буквенного префикса `f` перед строкой позволяет нам поместить в фигурные скобки в строке локальные переменные. Для того чтобы указать количество знаков после запятой в десятичных дробях, мы по-прежнему можем использовать `.2f`.

5. Далее проверяем, какая сторона робота ближе к препятствию – левая или правая, на основе чего передаем значения скорости двигателям:

```
...
# Отправляем это двигателям
if left_distance < right_distance:
    self.robot.set_left(nearest_speed)
    self.robot.set_right(furthest_speed)
else:
    self.robot.set_right(nearest_speed)
    self.robot.set_left(furthest_speed)
...
```

6. Вместо фиксированного времени мы устанавливаем паузу в соответствии с переменной задержки. Значение задержки выражается в миллисекундах, но нам нужно перевести его в секунды:

```
...
# Делаем паузу в соответствии с переменной задержки
sleep(delay * 0.001)
...
```

7. Весь последующий код оставляем без изменений. Полный код из этого файла вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter8>.

Когда вы запустите этот код, робот будет обходить препятствия более эффективно. Возможно, вам придется более точно подобрать параметры времени и другие значения, а также настроить условия реверсивного вращения двигателя и разворота задним ходом. Если робот не отъезжает на нужное расстояние, установите значение времени выше, а если он выполняет поворот слишком далеко от объекта, установите значение ниже.

Стоит отметить, что разработанный нами скрипт имеет недостатки. С его помощью робот не может построить карту и не получает данные от реверсивных датчиков, поэтому, уходя от столкновения с объектом впереди, он может врезаться в объект позади. Для решения этой проблемы роботу потребуются другие датчики. На данном этапе наш робот не оснащен датчиками, которые позволяют определить, насколько далеко от объекта робот выполнил поворот или отъехал, поэтому он не может строить карты.

## Выводы

В этой главе мы оснастили робота датчиками расстояния. Это важный шаг, который делает робота более автономным и позволяет ему реагировать на окружающую среду, не требуя при этом участия человека. Вы узнали, как робот может определять расстояние до объекта, а также познакомились с другими типами датчиков. Мы создали код, который позволяет взаимодействовать с датчиками, и протестировали его. Затем разработали поведенческие скрипты, позволяющие роботу избегать столкновений со стенами. Первый скрипт был довольно простой, а второй позволял роботу принимать более сложные решения и двигаться более плавно.

Обладая этим опытом, вы можете поразмышлять над тем, как другие датчики могут быть связаны с кодом и какой простой скрипт для взаимодействия с ними можно разработать. Можете выводить показания с датчиков, что позволяет создавать и отлаживать поведенческие скрипты, посредством которых робот сможет сам выполнять простую навигацию.

В следующей главе мы подробно рассмотрим движение по заранее заданным траекториям и прямым линиям с помощью энкодера. Мы увидим, что так робот будет двигаться более точно. Благодаря наличию энкодера мы сможем сравнить фактически пройденное расстояние с ожидаемым, ввиду чего наш робот сможет выполнять более точные повороты.

## Задание

1. Некоторые роботы обходятся всего одним датчиком. Подумайте, как добиться того, чтобы робот мог обходить препятствия с помощью всего одного датчика.
2. У нас есть механизм поворота и наклона, который позже мы будем использовать для камеры. Подумайте, как можно связать его с датчиком и разработать скрипт для взаимодействия с ним.

3. Наш поведенческий сценарий допускает, что робот может сталкиваться с предметами позади него. Как можно это исправить? По желанию можете разработать план и реализовать его.

## Дополнительные материалы

Дополнительную информацию вы можете получить из следующих источников:

- дальномер RCWL-1601 очень похож на модель HC-SR04. В техническом паспорте HC-SR04 вы можете найти полезную информацию о рабочем диапазоне этой модели. Технический паспорт найдете по адресу [https://www.mouser.com/ds/2/813/HC\\_SR04-1022824.pdf](https://www.mouser.com/ds/2/813/HC_SR04-1022824.pdf);
- на сайте ModMyPi вы можете найти руководство по альтернативному способу подключения датчиков HC-SR04 и трансляции логических уровней их контактов ввода/вывода: <https://www.modmypi.com/blog/hc-sr04ultrasonic-range-sensor-on-the-raspberry-pi> ; <https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi>;
- в разделе Raspberry Pi Tutorials по адресу <https://tutorials-raspberrypi.com/raspberry-pi-ultrasonic-sensor-hc-sr04/> вы можете найти схему макетной платы и скрипт на Python, использующий библиотеку RPi.GPIO вместо gpiozero;
- мы задействовали множество контактов на Raspberry Pi. Для того чтобы понять, какие контакты я использую, посетите сайт <https://pinout.xyz/>;
- мы кратко упомянули отладочные данные и их форматирование. У W3schools есть интерактивное руководство по форматированию строк Python. Найти его можно по адресу [https://www.w3schools.com/python/ref\\_string\\_format.asp](https://www.w3schools.com/python/ref_string_format.asp);
- существует множество исследовательских статей, посвященных более интересным и сложным поведенческим сценариям, которые позволяют роботу обходить препятствия. Я рекомендую ознакомиться со статьей «Simple, real-time obstacle avoidance algorithm for mobile robots» (<https://pdfs.semanticscholar.org/519e/790c8477cfb1d1a176e220f010d5ec5b1481.pdf>), чтобы лучше разобраться в таких сценариях.

# Глава 9

## Код на Python для работы со светодиодами

Светодиодные индикаторы используются для получения отладочных данных и реализации обратной связи с роботом. Мы можем создать код, который будет информировать пользователя о состоянии робота посредством светодиодов. RGB-светодиоды работают по принципу смешения основных цветов (красный, зеленый и синий), благодаря чему могут светиться разными цветами. С их помощью мы можем сделать робота ярче и красочнее. До этого мы мало говорили о том, как сделать нашего робота интереснее с точки зрения внешнего вида, поэтому сейчас сосредоточимся на этой теме.

Используя включение светодиодов в разной последовательности, можно реализовать передачу информации в реальном времени. Вы сможете сами решать, какие из них будут включены/выключены, регулировать их яркость и задавать определенный цвет для индикации той или иной информации. Такой тип обратной связи считывать проще, чем поток текстовой информации, что будет полезно при подключении к роботу различных датчиков. Также это означает, что мы сможем получать информацию о состоянии робота, не используя терминал SSH.

В этой главе мы рассмотрим следующие темы:

- что такое линейка RGB-светодиодов;
- сравнение технологий светодиодной индикации;
- подключение светодиодной линейки к Raspberry Pi;
- разработку кода для управления дисплеем робота;
- использование светодиодов для получения отладочных данных сценариев обхода препятствий;
- реализацию радужного свечения на светодиодной линейке.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам понадобится следующее:

- компьютер с доступом к интернету и Wi-Fi;
- наш робот, Raspberry Pi и код, который мы создали в предыдущей главе;
- светодиодная линейка LED SHIM от Pimoroni.



Код из этой главы вы можете найти на GitHub по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter9>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/39vglXm>.

## Что такое линейка RGB-светодиодов?

Использование светодиодов для отображения данных – простой и удобный способ информирования пользователя, не требующий подключения полноценного дисплея. Например, светодиодный индикатор может показывать включен робот или выключен, а также передавать данные с простых датчиков. Светодиодные RGB-индикаторы могут менять цвет свечения, обозначая разные состояния робота. Аббревиатура RGB расшифровывается как Red-Green-Blue (красный-зеленый-синий). Светодиоды меняют цвет свечения за счет изменения интенсивности основных цветовых каналов. В разделе «RGB-значения» мы поговорим об этом более подробно.

Чем больше светодиодов, тем больше у вас источников информации. Вы можете использовать линейки (несколько светодиодов, конструктивно объединенных в одну полосу), а также панели/матрицы, кольца и другие интересные формы дисплеев.

## Сравнение технологий светодиодной индикации

На сегодняшний день существует множество конкурирующих между собой светоизлучающих элементов и светящихся лент. Например, традиционные лампы накаливания не подойдут для использования в качестве светового индикатора для робота, поскольку они потребляют слишком много энергии и занимают достаточно много места. Компактные люминесцентные лампы или люминесцентные светильники, часто используемые для освещения кухонь, нуждаются в сложных системах питания, которые также занимают много места. Электролюминесцентными проводами (также их называют ЭЛ-провода) можно подсвечивать объекты по контуру. Это выглядит эффектно, но управлять такими системами довольно сложно. **Светоизлучающие диоды (LED – Light Emitting Diode)**, или просто светодиоды, отличаются низким энергопотреблением и небольшим размером, при этом они просты в управлении. Светодиодные индикаторы являются наиболее подходящим вариантом для нашего робота. К тому же, они недорогие.

Больше всего нас интересуют адресуемые RGB-светодиоды, и именно эту технологию мы будем использовать в этой главе. Они называются адресуемыми потому, что для каждого отдельного светодиода можно настроить цвет и яркость, что позволяет раскрасить светодиодную линейку в разные цвета по всей длине. Для упрощения работы мы будем использовать линейку, состоящую из светодиодов со встроенным контроллером.

На рис. 9.1 показаны некоторые варианты конструкций на основе RGB-светодиодов, с которыми я когда-то экспериментировал.

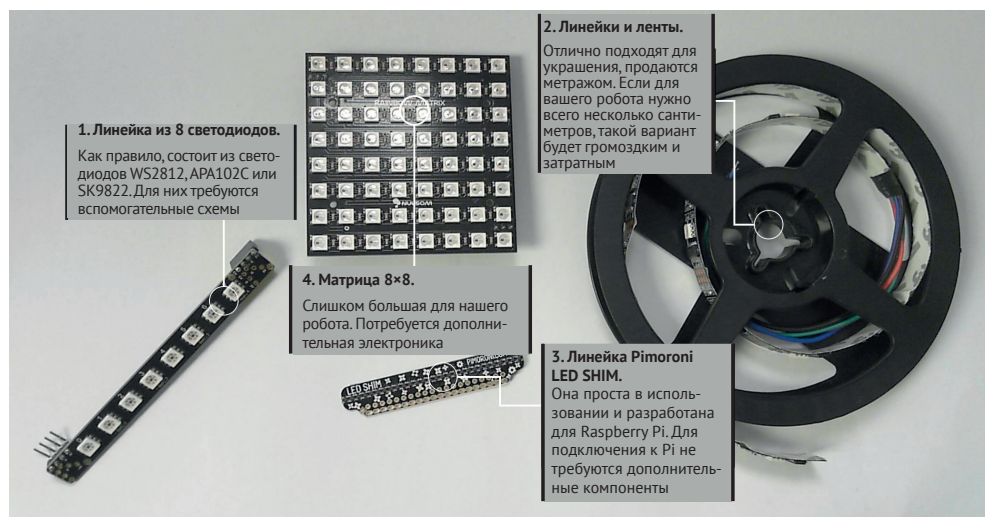


Рис. 9.1. Варианты конструкций на основе RGB-светодиодов

Все представленные варианты оснащены микроконтроллерами, которые управляют потоками данных. Некоторые из них, такие как Neopixel, WS2812, SK9822, APA102C, DotStar и 2801, работают по последовательному принципу: микроконтроллер подает сигнал на первый светодиод, который «забирает» необходимые красный, зеленый и синий компоненты, а оставшуюся цифровую информацию передает следующему светодиоду. Разработчики могут размещать в своих проектах светодиоды в разных конфигурациях (линейки, кольца, квадратные матрицы), чтобы воспользоваться всеми преимуществами этого способа передачи данных. Светодиодные линейки могут быть жесткими (в виде планок) или гибкими (в катушках). Для дисплейного элемента нашего робота понадобится восемь или более светодиодов.

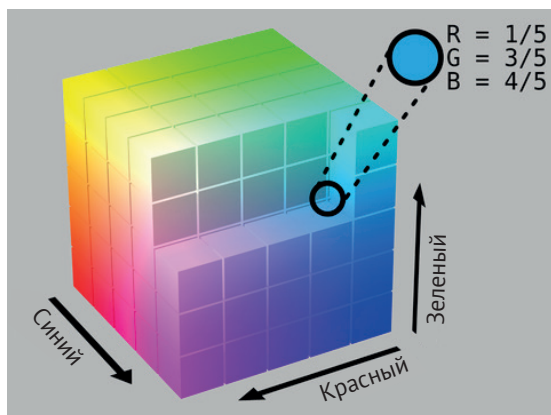
Существуют варианты, которые работают по другим технологиям, например светодиодные линейки LED SHIM от Pimoroni (линейка состоит из светодиодов со встроенным контроллером) или светодиодные RGB-матрицы со сдвиговыми регистрами. LED SHIM от Pimoroni – одна из самых простых в использовании (для Raspberry Pi) светодиодных линеек. Ее встроенный микроконтроллер подключается к Pi (или другому контроллеру) через шину I2C. В этой линейке 24 светодиода, чего для нас будет более чем достаточно. Кроме того, она широко доступна и не требует отдельной системы питания.

Контроллер двигателя нашего робота взаимодействует с Pi через шину I2C. К этой шине можно подключить другие устройства, которые имеют отдельный адрес, в том числе и LED SHIM. Управляющие сигналы в шине I2C содержат следующие данные: адрес устройства, регистр в I2C для записи и его значение.

Простая светодиодная линейка LED SHIM от Pimoroni отлично подходит для нашего робота, поэтому я рекомендую выбрать именно ее. Данную линейку можно приобрести в Mouser Electronics, который есть в большинстве стран, а также в магазинах Pimoroni, Adafruit и SparkFun.

## RGB-значения

Из красного, зеленого и синего цветов можно получить любые цветовые комбинации, которые выражаются в RGB-значениях. По этому принципу работает большинство (если не все) цветных дисплеев, которыми вы пользуетесь в повседневной жизни. Он используется в экранах телевизоров, мобильных телефонов и компьютеров. RGB-светодиоды работают по принципу смешения цветов, что позволяет менять цвет их свечения. Для смешения цветов в коде задается интенсивность каждого цвета в виде трехзначного числа, как показано на рис. 9.2.



**Рис. 9.2.** Цветовое пространство RGB [SharkD / CC BY-SA (<https://creativecommons.org/licenses/by-sa/3.0/>)]

На рис. 9.2 цветовое пространство RGB представлено в виде куба. Стрелками показаны векторы значений трех основных цветов – красного, зеленого и синего. Цвета смешиваются во всех точках куба, поэтому открытые поверхности куба в разных точках имеют разные оттенки и интенсивность.

В нижнем ближнем правом углу – синий цвет, в верхнем ближнем правом – бирюзовый (получается при смешении синего и зеленого), в нижнем ближнем левом – фиолетовый (красный и синий), в нижнем дальнем левом – красный (без зеленого и синего), а в верхнем дальнем левом – желтый (высокое значение красного и зеленый, без синего).

С увеличением значения каждого цвета мы получаем другие оттенки. В верхнем ближнем левом углу будет белый цвет, где значения трех основных цветов – максимальные. В нижнем дальнем правом углу будет черный, где значения трех основных цветов – минимальные. В вырезанной части куба показано, как меняется интенсивность цветов.

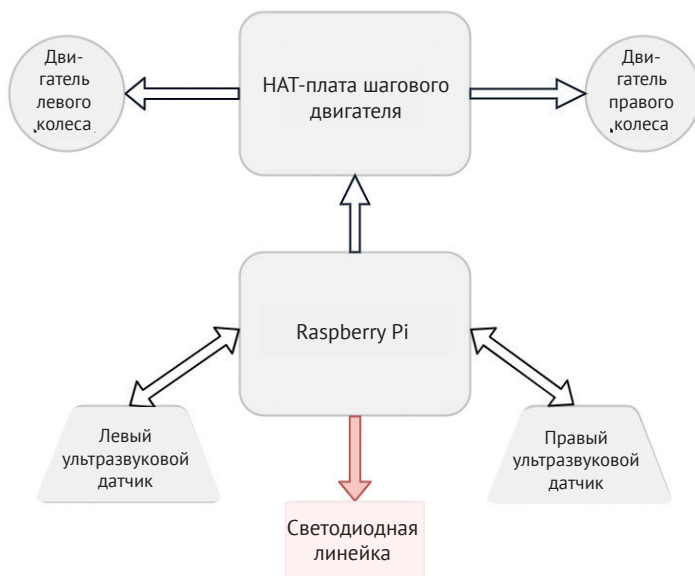
В нашем коде мы будем использовать значения от 0 (минимальная интенсивность) до 255 (максимальная интенсивность), а значения между ними – для других уровней интенсивности. Смешение происходит посредством суммирования цветов, поэтому, если задать всем цветам максимальное значение и суммировать их, получится белый.

В теории мы можем получить множество самых разных оттенков, но в реальности разница интенсивности свечения между значениями 250 и 255 у большинства RGB-светодиодов.

Мы рассмотрели некоторые технологии светодиодной индикации и узнали о том, как смешиваются цвета. Мы решили, что будем использовать светодиодную линейку LED SHIM от Pimoroni. В следующем разделе подключим ее к роботу. Приобретите данную линейку и возвращайтесь к книге.

## Подключение светодиодной линейки к RASPBERRY PI

Прежде чем создать код для отображения цветовых последовательностей на LED SHIM, необходимо подключить линейку к Raspberry Pi. К концу этого раздела блок-схема робота будет выглядеть так, как показано на рис. 9.3.



**Рис. 9.3.** Блок-схема робота со светодиодной линейкой

На блок-схеме появилась светодиодная линейка, подключенная к Raspberry Pi. Стрелка указывает, что поток информации идет от Pi на линейку и на схеме выделена красным, поскольку линейка является новым компонентом системы. Давайте посмотрим, как она работает.

## Установка светодиодной линейки на робота

Светодиодная линейка LED SHIM от Pimoroni легко устанавливается на Raspberry Pi. Мы подключим ее к линейному разъему контроллера двигателя, чтобы световые индикаторы находились сверху. На рис. 9.4 показано, как установить линейку.

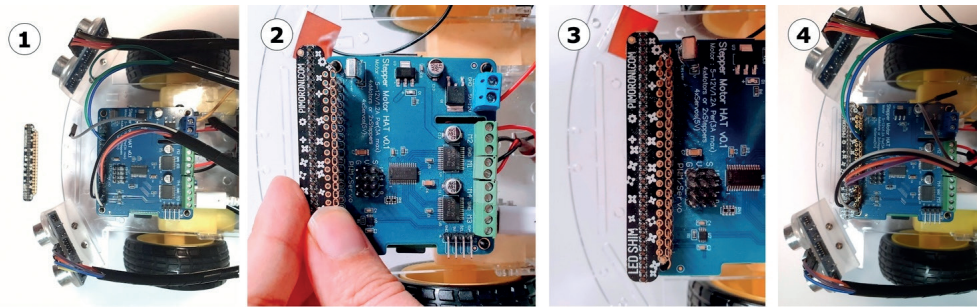


Рис. 9.4. Установка светодиодной линейки

1. Линейка небольшая. Совместите отверстия на ней с контактами линейного разъема в верхней части HAT-платы двигателя. Обратите внимание, что на данном этапе необходимо временно убрать подключенные к Pi провода.
2. Широкая часть должна выходить за пределы HAT-платы. Осторожно надавите на линейку, чтобы контакты вошли в отверстия. Постепенно прикладывайте усилие по всей длине линейки, пока все контакты не войдут в отверстия – с некоторым трудом, но это нужно для надежного контакта.
3. Когда все контакты войдут в отверстия, равномерно надавите на линейку, чтобы контакты выступили над отверстиями.
4. Теперь необходимо подключить провода, которые мы убрали. Для этого вернитесь к рис. 8.15 в главе 8.

Итак, вы установили линейку LED SHIM, и теперь мы можем перейти к созданию кода.

## РАЗРАБОТКА КОДА ДЛЯ УПРАВЛЕНИЯ ДИСПЛЕЕМ РОБОТА

Как вы уже знаете, помимо нашей LED SHIM от Pimoroni, существуют и другие типы светодиодных RGB систем. В дальнейшем вы можете заменить SHIM на другую систему, поэтому нам необходимо создать универсальный программный интерфейс (уровень) для управления светодиодами. Как и интерфейс, который мы создали для управления двигателями, он призван сделать поведенческие скрипты независимыми от аппаратных функций робота.

### Создание интерфейса для управления светодиодами

Каким должен быть интерфейс управления светодиодами? Во-первых, мы должны создать структуру данных `robot.leds`. Для начала необходимо «очистить» светодиодный дисплей (выключить все светодиоды), затем задать цвет свечения для каждого из них и настроить списки цветов для групп светодиодов / светодиодов одного диапазона.

В коде должно быть в явном виде указано количество имеющихся светодиодов, чтобы в дальнейшем, если оно изменится, можно было целенаправленно исправить настройки анимации и отображения.

Для того чтобы задать цвет свечения, мы будем использовать три значения – *r*, *g* и *b* для красного, зеленого и синего компонентов соответственно. В Python существует структура данных под названием **кортеж (tuple)**, которая может представлять собой группы, состоящие из небольшого количества элементов. В нашем случае в качестве элементов будут выступать наши компоненты (цвета). Параметру *color* будет присвоено значение, представляющее собой кортеж (*r*, *g*, *b*).

Светодиоды в нашей линейке являются адресуемыми по порядковым номерам, и нумерация в коде начинается с нуля.

Наша структура данных начинается с *robot.leds*.*leds* – элемента уже существующего класса *robot*. Этот объект включает следующие элементы:

- *set\_one(led\_number, color)*: задает определенный цвет для светодиода с номером *led\_number*;
- *set\_range(led\_range, color)*: задает параметр *color* для итерируемого объекта Python *led\_range*, который включает в себя несколько светодиодов. Итерируемый объект может представлять собой список номеров светодиодов [0, 3] или диапазон, созданный функцией *range*. Например, *range(2,8)* создает список [2, 3, 4, 5, 6, 7];
- *set\_all(color)*: задает цвет для всех светодиодов;
- *clear()*: задает всем светодиодам черный цвет, и они гаснут;
- *show()*: все вышеперечисленные методы позволяют настраивать комбинации светодиодов, но никакие изменения не будут видны, пока вы не вызовете этот метод. Вызов этого метода приводит к передаче светодиодам всех настроек в виде одного потока данных;
- *count*: показывает количество светодиодов в линейке.

С учетом всего вышесказанного создадим код для работы с LED SHIM.

1. Сначала необходимо установить библиотеку LED SHIM. На Raspberry Pi введите следующую команду:

```
pi@myrobot:~ $ pip3 install ledshim
```

2. Наш код должен начинаться с импорта этой библиотеки и настройки устройства. Создайте файл *leds\_led\_shim.py* (название выбрано в соответствии с типом устройства) и поместите в него следующий фрагмент кода:

```
import ledshim

class Leds:
    @property
    def count(self):
        return ledshim.width
```

Для настройки устройства достаточно ввести функцию *import ledshim*.

Мы настроили свойство *count*, указывающее количество светодиодов в нашем классе. Это свойство может считываться как переменная, но при этом оно неизменяемо и доступно только для чтения, что обеспечивает защиту от случайных изменений.

3. Теперь создаем методы для взаимодействия со светодиодной линейкой.



Настроить один светодиод достаточно просто:

```
def set_one(self, led_number, color):
    ledshim.set_pixel(led_number, *color)
```

Наш интерфейс использует кортеж (r, g, b), в то время как библиотека LED SHIM ожидает, что цвет будет задаваться отдельными параметрами. В Python есть небольшая хитрость, позволяющая представить кортеж как множество параметров. Для этого нужно добавить звездочку (\*) перед именем переменной. Например, как \*color во второй строке представленного выше фрагмента кода.

Если пользователь попытается задать значение, выходящее за пределы допустимого диапазона, код выдаст ошибку KeyError.

4. Настроить группу светодиодов можно с помощью простой функции-обертки:

```
def set_range(self, led_range, color):
    ledshim.set_multiple_pixels(led_range, color)
```

5. Также нам нужен метод, позволяющий настроить все светодиоды. Этот код аналогичен коду настройки одного светодиода:

```
def set_all(self, color):
    ledshim.set_all(*color)
```

6. Далее добавим метод, позволяющий задавать всем светодиодам черный цвет (выключать их):

```
def clear(self):
    ledshim.clear()
```

7. Теперь нам необходим метод show, который позволит управлять светодиодами линейки и задавать им цвет свечения в соответствии с нашей настройкой. С помощью библиотеки LED SHIM сделать это довольно просто:

```
def show(self):
    ledshim.show()
```

Мы установили библиотеку LED SHIM и создали интерфейс для управления светодиодами. При замене светодиодного устройства можем внести небольшие изменения в его код и сделать его совместимым с новым устройством. Далее мы поместим этот интерфейс в объект Robot.

## Добавляем светодиоды в объект Robot

Далее обновим файл robot.py, чтобы взаимодействовать со светодиодной системой. Для этого выполним следующие шаги.

1. Для начала необходимо добавить импорт файла leds\_led\_shim (изменения в коде выделены жирным шрифтом):

```
from Raspi_MotorHAT import Raspi_MotorHAT
from gpiozero import DistanceSensor
import atexit
import leds_led_shim
```



2. Затем мы добавляем экземпляр объекта линейки SHIM в метод конструктора (`init`) для объекта `Robot` (выделено жирным):

```
class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Настройка HAT-платы в соответствии с переданным адресом
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)

        # Получение локальных переменных для каждого двигателя
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)

        # Настройка датчиков расстояния
        self.left_distance_sensor =
DistanceSensor(echo=17, trigger=27, queue_len=2)
        self.right_distance_sensor =
DistanceSensor(echo=5, trigger=6, queue_len=2)

        # Настройка светодиодов
        self.leds = leds_led_shim.Leds()
```

3. Нам необходимо приостанавливать работу не только двигателей, но и других устройств (например, сбрасывать значения светодиодов). Для этого заменим `stop_motors` в вызове `atexit` новым методом `stop_all`.

```
# убедимся, что после завершения кода все устройства прекращают работу
atexit.register(self.stop_all)
```

4. Далее мы создаем метод `stop_all`, который приостанавливает работу двигателей и сбрасывает значения светодиодов:

```
def stop_all(self):
    self.stop_motors()

    #Сбрасываем значения светодиодов
    self.leds.clear()
    self.leds.show()
```

### Важное примечание

Полный код можно найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter9>.

Итак, мы добавили поддержку управления светодиодами в класс `Robot`. В нем мы указали, что программа должна гасить светодиоды при завершении работы кода. Далее протестируем светодиоды в действии.

## Тест одного светодиода

Мы установили новый аппаратный компонент (светодиодную линейку) и соответствующую библиотеку, а затем создали код для доступа к этому уровню. Прежде чем пойти дальше, необходимо протестировать светодиоды и убедиться, что все работает. На этом этапе мы можем выявить и устранить многие неполадки.

Для начала протестируем один светодиод. До этого мы не упоминали о том, что наш робот способен запускать Python REPL<sup>9</sup> – программу-оболочку, позво-

<sup>9</sup> Программу REPL также называют интерактивным интерпретатором Python. – Прим. пер.

ляющую сразу после запуска Python вводить и исполнять код на Python. Мы будем использовать ее для проверки светодиодов.

1. Скопируйте код из файлов `leds_led_shim.py` и `robot.py` в Raspberry Pi.
2. Подключитесь к роботу через терминал SSH и введите команду `python3`. Ответ от Raspberry Pi будет выглядеть следующим образом:

```
pi@myrobot:~ $ python3
Python 3.7.3 (default, Apr 3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3. Подготовим к использованию библиотеку `robot`. Введите строки, выделенные жирным шрифтом:

```
>>> import robot
>>> r = robot.Robot()
```

4. Теперь попробуем включить светодиод, задав ему красный цвет свечения:

```
>>> r.leds.set_one(0, (255, 0, 0))
```

5. Хм, ничего не произошло. Не забывайте, для того чтобы настройка сработала, нужно вызвать метод `leds.show`:

```
>>> r.leds.show()
```

Теперь один светодиод должен загореться красным.

6. Давайте попробуем задать другому светодиоду фиолетовый цвет свечения, смешав красный и синий:

```
>>> r.leds.set_one(5, (255, 0, 255))
>>> r.leds.show()
```

Этот код задействует светодиод под номером 5, поэтому теперь вы можете видеть, что активны два светодиода.

### Важное примечание

Не забывайте использовать `leds.show`, чтобы передать значения цветов на светодиодную линейку.

7. Для завершения сеанса переместите курсор на пустую строку и нажмите **Ctrl+D**. Код `atexit` автоматически отключит все светодиоды.

Итак, мы убедились, что отдельные светодиоды работают и мы можем задавать им разные цвета свечения. Это означает, что на данном этапе проблем в коде нет. Если возникли какие-либо неполадки, обратитесь к следующему подразделу. Если все работает, перейдите к подразделу «Тест всех светодиодов».

### Устранение неполадок

Если при проверке светодиода вы столкнулись с неполадками, воспользуйтесь пошаговыми советами по их устранению, представленными ниже.

В случае возникновения ошибок при запуске кода выполните следующие действия:

- убедитесь, что у вас включена шина I2C (как ее включить, рассказывается в главе 7);
- введите команду `sudo i2cdetect -y 1` (как в главе 7). В ее выводе вы должны увидеть светодиодное устройство с адресом 74;
- убедитесь, что вы установили пакет Python `ledshim` с помощью `pip3`;
- внимательно проверьте код на наличие ошибок. Если вы копировали его с GitHub, сообщите о проблеме!

Если не работают светодиоды, предпримите следующие действия:

- попробуйте запустить образец кода, предназначенный специально для SHIM. Его вы можете найти по адресу <https://github.com/pimoroni/led-shim/tree/master/examples>;
- убедитесь, что светодиодная линейка установлена правильно (в соответствии с рис. 9.4);
- убедитесь, что все контакты линейного разъема HAT-платы двигателя вошли в отверстия на линейке;
- проверьте, использовали ли вы `leds.show()`.

Представленные выше советы призваны решить наиболее распространенные проблемы, возникающие при работе со светодиодной системой. На этом этапе у вас должен быть рабочий светодиод, после чего можно перейти к следующему подразделу.

## Тест всех светодиодов

Сейчас мы протестируем метод `set_all`. В этом подразделе мы заставим светодиоды на нашей линейке поочередно вспыхивать красным и синим светом. Для начала создайте файл с названием `leds_test.py`.

1. Чтобы привести в действие наши светодиоды, необходимо импортировать библиотеки `Robot` и `time`:

```
from robot import Robot
from time import sleep
```

2. Теперь давайте объявим объект `bot` и зададим значения для двух цветов:

```
bot = Robot()
red = (255, 0, 0)
blue = (0, 0, 255)
```

3. Следующий фрагмент кода – это основной цикл. В этом цикле чередуются вспышки двух цветов и функции `sleep`:

```
while True:
    print("red")
    bot.leds.set_all(red)
    bot.leds.show()
    sleep(0.5)
    print("blue")
    bot.leds.set_all(blue)
    bot.leds.show()
    sleep(0.5)
```

Методы `print` показывают, когда система отправляет данные светодиодам. Мы задаем для всех светодиодов значение `red` посредством метода `set_all`, а затем вызываем метод `show`, чтобы отправить данные на микроконтроллер светодиодной ленты. Функция `sleep` приостанавливает исполнение кода на 0,5 с, а затем всем светодиодам задается значение `blue`.

#### Важное примечание

Полный код можно найти по адресу [https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter9/leds\\_test.py](https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter9/leds_test.py).

4. После загрузки файлов в Raspberry Pi необходимо просмотреть результат – вы должны увидеть красные и синие вспышки световых индикаторов. Для этого введите следующую команду:

```
pi@myrobot:~ $ python3 leds_test.py
```

5. Чтобы сообщить терминалу, что необходимо прервать исполнение программы, нажмите **Ctrl+C**.

Мы убедились, что все светодиоды работают. Также мы увидели, что они могут переключаться между разными цветами свечения с небольшими паузами. Можем пойти дальше и создать более интересные варианты цвета и анимации, но сначала поговорим подробнее о смешивании цветов.

## РЕАЛИЗАЦИЯ РАДУЖНОГО СВЕЧЕНИЯ НА СВЕТОДИОДНОЙ ЛИНЕЙКЕ

Теперь немного развлечемся и расширим скрипты обхода препятствий, разработанные в предыдущей главе. Мы добавим светодиодную анимацию в виде радужной гистограммы, которая будет появляться со стороны, расположенной ближе к препятствию в соответствии с данными, полученными от датчиков. Но перед тем, как связать движение робота и анимацию светодиодов, давайте узнаем, как создается радужное свечение.

### Цветовые модели

Модель RGB отлично подходит для управления аппаратными компонентами системы. Однако она не совсем удобна для кодировки промежуточных оттенков и градиентных переходов. Цвета в RGB-модели могут не соответствовать восприятию человеческого глаза. По этой причине были разработаны другие цветовые модели.

Мы будем использовать еще одну цветовую модель – **HSV (Hue, Saturation, Value – Цветовой тон, насыщенность, значение цвета)**. В этой главе мы будем использовать HSV для реализации радужного свечения, а в следующей главе она пригодится нам для задач компьютерного зрения и поможет обнаруживать объекты.

## Цветовой тон

Представьте, что вы берете цвета спектра и помещаете их в круг. Красный цвет переходит в оранжевый, оранжевый – в желтый, желтый – в зеленый, зеленый – в синий, синий – в фиолетовый, а фиолетовый – снова в красный. Значение **цветового тона** представляет собой точку на этом круге. При этом оно не влияет на яркость и интенсивность цвета. На рис. 9.5 показано, как располагаются точки тона на цветовом круге.



Рис. 9.5. Точки тона на цветовом круге

Согласно рис. 9.5 красный тон находится на отметке 0°. Другие точки обозначают другие тона. По мере того как один тон переходит к другому, цвета смешиваются. Возможно вы уже видели нечто подобное в программах для рисования. При переходе одних тонов в другие по кругу получается радуга.

## Насыщенность

Красный цвет может быть разным: сероватым/тусклым или ярким/интенсивным. **Насыщенность** – это интенсивность чистого цвета. В центре нашего круга находятся только оттенки серого. По мере увеличения насыщенности начинает проявляться цвет – сначала в пастельных тонах, потом в более ярких «постерных», постепенно переходя к чистым наиболее ярким тонам, которые проявляются ближе к границе нашего круга.

## Значение цвета

**Значение цвета** – это его яркость. Минимальное значение (0) соответствует черному цвету, промежуточное значение – очень темному оттенку цвета, а максимальное – очень яркому. Обратите внимание, что при максимальном значении красный цвет будет не светло-розовым, а ярко-красным. Чтобы получить белый цвет, нужно уменьшить насыщенность. В других системах, например в HSL (Hue, Saturation, Lightness – цветовой тон, насыщенность, светлота), есть цветовая координата, определяющая *светлоту цвета*, с помощью которой можно получить белый цвет.

## Преобразование цвета из HSV в RGB

Существуют сложные формулы для преобразования цветов из одной системы в другую. Однако мы можем сделать это с помощью Python.

Для преобразования будем использовать библиотеку `colorsys.hsv_to_rgb`. Эта библиотека определяет три компонента HSV как дробные значения от 0 до 1 включительно.

Если цветовой тон определяется как 0 – значит, это начало круга, если 0,5 – 180° (полукруг), а 1 – весь круг до 360°, полный круг.

Значение насыщенности 0 определяется как полностью ненасыщенный серый, а 1 – самый интенсивный оттенок.

Если значение цвета равно 0, то он определяется как полностью черный, а 1 – самый яркий, полностью светлый оттенок.

Для того чтобы получить яркий голубой цвет (cyan), нам нужны следующие значения: 0,6 для цветового тона, 1,0 для насыщенности и 1,0 для значения цвета:

```
cyan = colorsys.hsv_to_rgb(0.6, 1.0, 1.0)
```

Однако этого недостаточно. Результатом вызова `colorsys` является кортеж, набор из трех значений для компонентов R, G и B.

Значения этих компонентов также находятся в диапазоне от 0 до 1,0. Большинство систем RGB ожидает значения от 0 до 255. Так, нам необходимо преобразовать эти значения, умножив их:

```
cyan_rgb = [int(c * 255) for c in cyan]
```

Здесь мы умножаем каждый компонент на 255. В Python можем создать цикл, который будет последовательно обрабатывать все значения и помещать их в список. Для этого поместим оператор цикла `for` в квадратные скобки.

Теперь, когда вы знаете, как преобразовать значения из HSV в RGB, давайте воспользуемся этой информацией, чтобы реализовать радужное свечение.

## Радужное свечение светодиодов

Сейчас вы знаете больше о цветовых системах поэтому можете заставить светодиодную линейку переливаться всеми цветами радуги.

На рис. 9.6 показано, как выглядит светодиодная линейка с радужным свечением.



Рис. 9.6. Светодиодная линейка с радужным свечением

Давайте реализуем это! Создайте новый файл с именем `led_rainbow.py`:

```
import colorsys

def show_rainbow(leds, led_range):
    led_range = list(led_range)
    hue_step = 1.0 / len(led_range)
    for index, led_address in enumerate(led_range):
        hue = hue_step * index
        rgb = colorsys.hsv_to_rgb(hue, 1.0, 0.6)
        rgb = [int(c*255) for c in rgb]
        leds.set_one(led_address, rgb)
```

Рассмотрим каждую строку данного фрагмента кода подробнее:

- сначала выполняется импорт `colorsys`;
- мы определяем функцию `show_rainbow` с двумя параметрами: ссылкой на нашу светодиодную систему (которая называется `robot.leds`) и диапазоном значений светодиодов;
- поскольку мы хотим знать длину нашей светодиодной линейки, необходимо убедиться, что значения будут представлены в виде списка. Для этого мы задаем соответствующую структуру данных в первой строке нашей функции;
- для того чтобы мы могли получить радужное свечение, значение цветового тона должно охватывать полный круг. В `colorsys` это значения от 0 до 1. Нам необходимо увеличивать значение цветового тона для каждого светодиода в диапазоне на определенный шаг – дробное число. Чтобы получить это число, нужно разделить 1,0 на количество светодиодов в диапазоне;
- затем мы создаем цикл, где будет обрабатываться каждый светодиод. С помощью функции `enumerate` мы получаем индекс для каждого элемента, пока цикл проходит по адресам (`led_address`). Этот код не делает



никаких предположений о диапазоне, поэтому он может использовать произвольный список светодиодов;

- далее мы умножаем `hue_step` на `index` и получаем значение `hue`, которое можем использовать – дробное число от 1,0. Следующая строка преобразует его в значение RGB с фиксированной насыщенностью и яркостью;
- значения в выводе `coloursys` находятся в диапазоне от 0 до 1, поэтому код должен умножить дробное значение на 255 и преобразовать его в целое число: `rgb = [int(c*255) for c in rgb]`;
- затем код использует метод `leds.set_one` с полученным значением RGB и адресом светодиода.

Давайте проведем тест в файле `test_rainbow.py`:

```
from time import sleep
from robot import Robot
from led_rainbow import show_rainbow

bot = Robot()

while True:
    print("on")
    show_rainbow(bot.leds, range(bot.leds.count))
    bot.leds.show()
    sleep(0.5)
    print("off")
    bot.leds.clear()
    bot.leds.show()
    sleep(0.5)
```

Мы выполняем тест, подобный тому, который мы проводили, когда задавали нашим светодиодам программу попеременно вспыхивать красным/синим цветами. Однако здесь мы используем функцию `show_rainbow`, импортированную из `led_rainbow`. Эта функция работает с диапазоном всех имеющихся светодиодов.

После паузы в программе (0,5 с) на это же время сбрасываются значения светодиодов. Мы работаем с циклом, что позволяет создать эффект включения/выключения радужного свечения. Запустите программу с помощью команды `python3 test_rainbow.py`, а после того, как убедитесь, что она работает, нажмите **Ctrl+C** для ее завершения.

Теперь вы знаете, как создать простую анимацию для RGB-светодиодов. Далее мы можем перейти на следующий уровень и реализовать совместную работу датчиков и светодиодов.

## ИСПОЛЬЗОВАНИЕ СВЕТОДИОДОВ ДЛЯ ПОЛУЧЕНИЯ ОТЛАДОЧНЫХ ДАННЫХ В СЦЕНАРИЯХ ОБХОДА ПРЕПЯТСТВИЙ

Радужное свечение светодиодов и переключение цветов делает нашего робота интереснее с точки зрения внешнего вида, но светодиоды можно применять и в практических целях. В главе 8 мы оснастили нашего робота датчиками, которые позволяют ему обходить препятствия. Все данные, получаемые датчиками, отображаются в числовом формате в PyTTY. Однако мы можем добиться большего и использовать светодиоды для индикации этих данных.

В этом разделе мы свяжем работу светодиодов и значения в поведенческих скриптах. Сначала поработаем с простым свечением, а затем попробуем реализовать радужное свечение.

## Реализация обычного свечения

Прежде чем вернуться к радужному свечению, поработаем с обычным. Наша цель состоит в том, чтобы разделить светодиодную линейку на два сегмента – левый и правый. С той стороны, с которой, исходя из данных датчиков, будет ближе находиться препятствие, будет вспыхивать больше светодиодов. Сегменты будут сходиться в середине линейки, поэтому если горит один светодиод с краю – значит, препятствие находится далеко, а если горит несколько (или большинство) светодиодов с одной стороны, значит, препятствие находится близко к этой стороне.

В нашем поведенческом скрипте необходимо сделать следующее:

- настроить переменные для нашего светодиодного отображающего элемента и сопоставление с результатами измерений, поступившими от датчиков;
- добавить способ преобразования расстояния в количество активных светодиодов;
- добавить метод для отображения состояния наших датчиков на светодиодах с использованием предыдущих элементов;
- вызвать метод `display_state` для основного цикла поведенческого скрипта.

Давайте посмотрим, как реализовать все перечисленные выше пункты. Откройте файл `escape_behavior.py`, созданный в главе 8, и выполните следующие шаги.

1. Прежде чем использовать светодиоды, необходимо разделить их на два сегмента. В метод `__init__` в скрипте `ObstacleAvoidingBehavior` добавьте следующее:

```
# Вычисления для светодиодов
self.led_half = int(self.robot.leds.leds_count/2)
```

2. Затем нам необходимо задать цвет свечения для светодиодов во время сбора данных датчиками. Я выбрал красный, но вы можете попробовать любой другой:

```
self.sense_colour = 255, 0, 0
```

3. Разобравшись с настройками переменных, добавим метод преобразования расстояния в количество активных светодиодов. Я добавил его после метода `__init__`:

```
def distance_to_led_bar(self, distance):
```

4. Расстояния выражены в метрах, где значение 1,0 соответствует 1 м. Если вычесть расстояние из 1,0, мы инвертируем значение. Функция `max` вернет наибольшее из двух значений. Здесь она гарантирует, что значения будут положительными:

```
# Инверсия измеренного расстояния, чтобы
# при уменьшении значения загоралось больше светодиодов
inverted = max(0, 1.0 - distance)
```

5. Теперь мы умножаем полученное значение (десятичное значение в диапазоне от 0 до 1) на значение `self.led_half` и получаем количество используемых светодиодов. Мы округляем его и превращаем в целое число с помощью `int(round())`, так как количество светодиодов может быть выражено только целым числом. Например, если после умножения мы получаем значение 3,8, то его необходимо округлить до 4,0. Значение преобразуется в целое число и означает, что будут активны 4 светодиода. Мы прибавляем к этому значению 1, чтобы один светодиод был активен всегда, а затем возвращаем полученное число:

```
led_bar = int(round(inverted * self.led_half))
return led_bar
```

6. Следующий метод несколько сложнее. С его помощью мы разделим линейку на два сегмента. Начнем с объявления метода и сброса значений светодиодов:

```
def display_state(self, left_distance, right_distance):
    # Сначала сбрасываем значения светодиодов
    self.robot.leds.clear()
```

7. Для левого сегмента мы преобразуем расстояние, полученное левым датчиком, в количество активных светодиодов, а затем создаем диапазон от 0 до этого числа. Метод `set_range` настраивает `sense_color` для группы светодиодов. Обратите внимание, что у вас сегменты могут располагаться иначе. В этом случае поменяйте `left_distance` и `right_distance` в методе `display`:

```
# Левая сторона
led_bar = self.distance_to_led_bar(left_distance)
self.robot.leds.set_range(range(led_bar), self.sense_colour)
```

8. С правым сегментом все немного сложнее. После преобразования данных расстояния в количество светодиодов нам нужно создать диапазон для светодиодов. Переменная `led_bar` отражает количество активных светодиодов. Чтобы активировать правый сегмент, необходимо вычесть количество активных светодиодов из общего числа. Так мы определим первый светодиод. Затем необходимо создать диапазон, начинающийся с общей длины. Мы должны вычесть 1 из длины, иначе первый светодиод загорится слишком рано:

```
# Правая сторона
led_bar = self.distance_to_led_bar(right_distance)
# Здесь немного сложнее - нужно отсчитывать светодиоды в обратном порядке.
start = (self.robot.leds.count - 1) - led_bar
self.robot.
leds.set_range(range(start, self.robot.leds.count - 1), self.sense_colour)
```

9. Далее нам необходимо продемонстрировать работу отображающего элемента:

```
# Отображаем состояние светодиодов на дисплее
self.robot.leds.show()
```

10. Затем мы отображаем показания датчиков на светодиодах, вызывая `display_state` внутри метода `run` из поведенческого скрипта. Ниже показан соответствующий фрагмент кода, а новая строка выделена жирным шрифтом:

```
# Получаем показания датчиков в метрах
left_distance = self.robot.left_distance_sensor.distance
right_distance = self.robot.right_distance_sensor.distance
# Отображаем созданный дисплей
self.display_state(left_distance, right_distance)
# Получаем скорости вращения двигателей исходя из расстояния
nearest_speed, furthest_speed, delay = self.get_speeds(min(left_distance,
right_distance))
```

Сохраните код, загрузите его в Raspberry Pi и запустите. Вы должны увидеть, как светодиоды загораются на линейке в зависимости от расстояния до препятствия. Кроме того, что за этим приятно наблюдать, такой подход позволяет легче понимать поведение робота. Давайте сделаем нашу систему еще интереснее, добавив радужное свечение.

## Реализация радужного свечения

Мы можем использовать нашу светодиодную радугу, чтобы сделать демонстрацию измерения расстояния еще более интересной.

На рис. 9.7 показано, как выглядят сегменты светодиодной линейки с радужным свечением, предназначенные для двух датчиков расстояния. Это отличная наглядная демонстрация анимации светодиодов.



Рис. 9.7. Сегменты с радужным свечением

У нас уже есть библиотека для реализации радужного свечения, и мы можем использовать ее повторно. Посмотрим, как это сделать.

1. Откройте файл `escape_behaviour.py` с кодом, который мы создали в предыдущем разделе.
2. Импортируйте `led_rainbow` для дальнейшего использования:

```
from robot import Robot
from time import sleep
from led_rainbow import show_rainbow
```

3. В нашем коде отображается сегмент для левой стороны. Вместо этого задайте здесь радужное свечение. Нужно убедиться, что у нас есть хотя бы один элемент для отображения:

```
# Левая сторона
led_bar = self.distance_to_led_bar(left_distance)
show_rainbow(self.robot.leds, range(led_bar))
```

4. Опять же, с правой стороной все обстоит немного сложнее, так как нам необходимо, чтобы анимация радуги отображалась в противоположную сторону. Следовательно, нам необходимо отсчитать диапазон в обратном направлении. Функция Python `range`, наряду с параметрами `start` и `end`, принимает параметр шага. Задав шаг `-1`, мы можем произвести обратный отсчет в диапазоне:

```
start = (self.robot.leds.count - 1) - led_bar
right_range = range(self.robot.leds.count - 1, start, -1)
show_rainbow(self.robot.leds, right_range)
```

5. Загрузите код в Raspberry Pi и запустите его. Теперь наши гистограммы расстояния имеют радужное свечение.

Мы перешли от настройки одного светодиода к нескольким. Поработав с системами кодирования цвета, мы смогли реализовать радужное свечение и использовать его для индикации поведения.

## Выводы

В этой главе вы узнали, как использовать RGB-светодиоды и взаимодействовать с ними. Также мы обсудили как выбрать светодиодную RGB-линейку, которая подойдет для Raspberry Pi. Вы создали код, позволяющий отображать поведение робота с помощью светодиодных индикаторов. Также вы увидели, как работает цветовая система HSV, которую можно использовать для реализации радужного свечения.

Можете использовать методы из этой главы, чтобы реализовать отображение состояния робота с помощью светодиодов, и написать код, чтобы связать их свечение с поведением.

В следующей главе мы рассмотрим серводвигатели и построим механизм поворота и наклона для подвижных датчиков.

## Задание

1. Попробуйте вычислить другие цветовые компоненты RGB или найдите готовое значение, а затем использовать `set_one`, `set_all` или `set_range`, чтобы задать это значение светодиодам.

2. Создайте и используйте функции `show_left_rainbow` и `show_right_rainbow`, чтобы робот включал радужное свечение на той стороне, в которую поворачивает при выполнении кода `behavior_path`.
3. Создав цикл времени и продвигая индекс или меняя диапазон, можно «оживить» радуги и заставить их *проходить* по светодиодной полосе. Попробуйте это реализовать.
4. Можно ли использовать другие компоненты цвета HSV для создания светодиодных лент, меняющих яркость?

## Дополнительные материалы

Больше информации вы можете найти в следующих источниках:

- «*Make Electronics: Learning by Discovery*», Чарльз Платт (Charles Platt), *Make Community, LLC*: я рассказал лишь немного об электронных устройствах с переключателями и макетными платами. Познакомьтесь с книгой «*Make Electronics*», которая позволит лучше разобраться в этой теме;
- если хотите узнать о более продвинутых электронных устройствах, прочтите книгу «*Practical Electronics for Inventors, Fourth Edition*», Пауль Шерц (Paul Scherz), Саймон Монк (Simon Monk), McGraw-Hill Education TAB. Здесь вы можете найти информацию о строительных блоках для электроники, которые можно использовать для сопряжения контроллера робота с любыми устройствами или новыми датчиками;
- для библиотеки `colorsys`, как и для большинства основных библиотек Python, есть отличные справочные материалы. С ними вы можете ознакомиться по адресу <https://docs.python.org/3/library/colorsys.html>;
- у Pimoroni есть и другие демонстрационные примеры с LED SHIM. Их вы можете найти по адресу <https://github.com/pimoroni/led-shim/tree/master/examples>. Было бы интересно попробовать адаптировать их к нашему интерфейсу управления светодиодами.

# Глава 10

## Код на Python для управления сервоприводами

Сервоприводы позволяют выполнять повторяемые точные действия. Они имеют принципиальное значение для перемещения подвижных датчиков и «конечностей» робота, поскольку позволяют приводить их в определенное положение. Сервоприводы применяются в задачах прогнозирования поведения робота, с их помощью можно задавать роботу автоматическую программу выполнения повторяющихся действий, а также посредством кода приводить «конечности» в действие в соответствии с показаниями датчиков. Управлять сервоприводами можно с помощью Raspberry Pi или плат расширения. В этой главе мы будем использовать сервоприводы для создания механизма поворота и наклона, предназначенного для позиционирования датчиков.

Эта глава призвана раскрыть следующие темы:

- что такое сервоприводы;
- позиционирование сервоприводов с помощью Raspberry Pi;
- создание механизма поворота и наклона;
- разработку кода для механизма поворота и наклона.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам понадобится следующее:

- робот с Raspberry Pi, которого мы собрали в предыдущих главах;
- набор небольших крестовых отверток от Phillips;
- маленькие плоскогубцы или гаечные ключи;
- набор нейлоновых болтов и стоек – 2,5 мм;
- набор для сборки миниатюрного двухосевого механизма поворота и наклона с сервоприводами;
- два сервопривода SG90/9g или MG90s с необходимыми крепежными деталями и качалками.
- В набор для сборки механизма поворота и наклона может входить следующее:
  - кусачки или бокорезы;
  - защитные очки.



Код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter10>.

Посмотреть видеоролик Code in Action можно по адресу <https://bit.ly/2LKk92g>.

## Что такое сервоприводы?

**Сервоприводы**, или **приводы со следящим механизмом**, применяются для позиционирования дополнительных компонентов робота, таких как манипуляторы, захватные устройства, ноги (у шагающих роботов) и опоры подвижных датчиков. В отличие от двигателей колес (ДПК), где определяющим фактором является частота вращения, сервоприводы обеспечивают другой тип движения, при котором определяющим фактором является положение. Сервоприводы применяются в механизмах, где требуются точные движения и повороты на определенный угол. Управлять такими движениями и их последовательностью можно посредством кода.

Сервоприводы бывают разных размеров, от небольших (20–30 мм), как на рис. 10.1, до достаточно крупных, которые могут работать с тяжелой техникой. На рис. 10.2 показаны миниатюрные сервоприводы, которые я использовал в своих проектах.

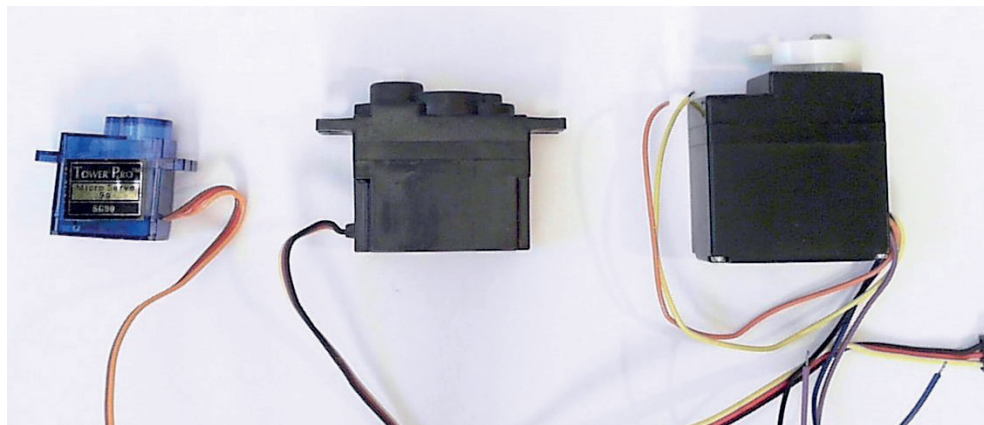


Рис. 10.1. Небольшие сервоприводы

Теперь, когда вы узнали, где применяются сервоприводы, мы можем перейти к рассмотрению принципов их работы.

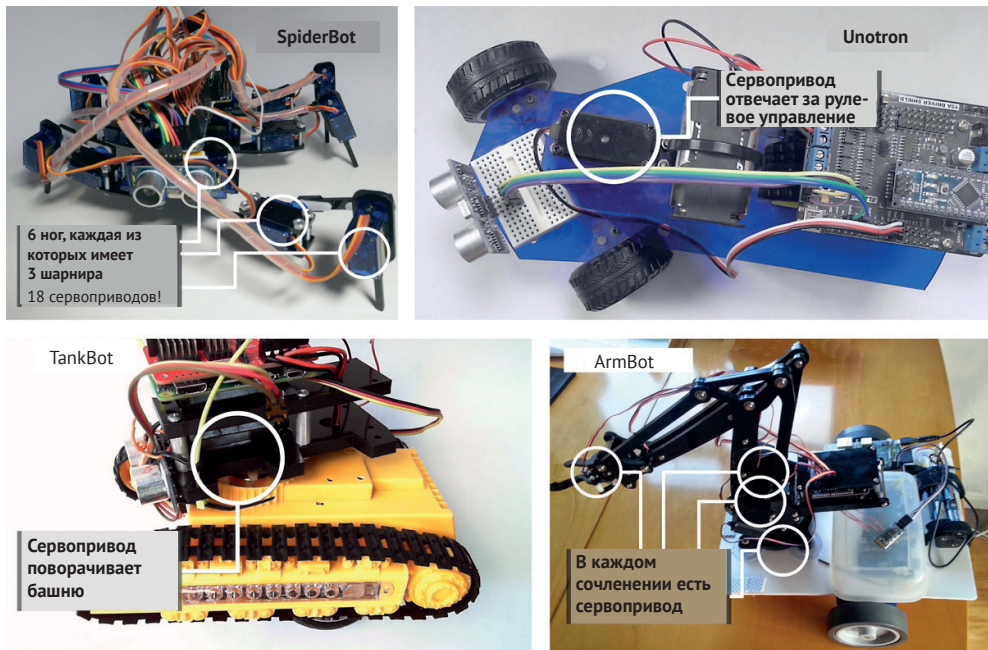


Рис. 10.2. Сервоприводы в роботах

## Сервопривод: взгляд изнутри

Несмотря на компактную форму, внутри сервопривода находится контроллер, ДПТ, редуктор (обычно с ограничителем) и встроенный датчик положения вала. Сервоприводы имеют встроенную систему управления с обратной связью. Сервопривод получает управляющие импульсы (данные о нужном положении) от главного контроллера. Контроллер сервопривода в свою очередь получает данные о текущем положении вала от встроенного датчика. Затем этот контроллер сравнивает текущее положение привода с полученным командным значением и определяет разницу между ними. Основываясь на этом значении, сервопривод производит некоторое действие так, чтобы разница стала минимальной. При работе привода меняются показания датчика и, соответственно, значение разницы. Так формируется петля обратной связи, показанная на рис. 10.3.

Некоторые типы сервоприводов, например как в ArmBot (рис. 10.2), имеют дополнительный выход, позволяющий считывать показания датчика положения в коде.

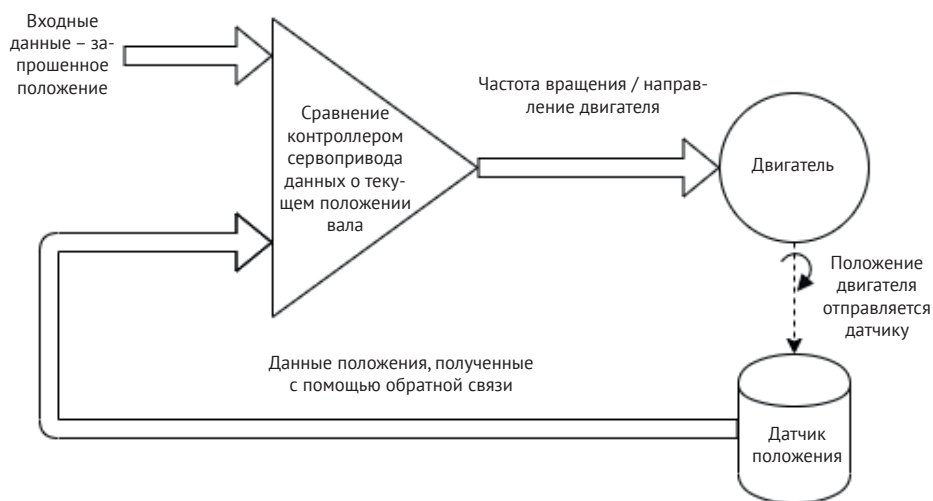


Рис. 10.3. Петля обратной связи сервопривода

## Отправка управляющих сигналов сервоприводу

Сервоприводы управляются сигналами **ШИМ (PWM)**. Эту технологию мы уже рассматривали в главе 2. В нашем роботе сигналы ШИМ управляют вращением двигателей колес. ШИМ представляет собой прямоугольный сигнал, имеющий только два состояния: *включенное* и *выключенное*, которые соответствуют низкой и высокой длительности сигнала (рис. 10.4). Мы можем представить эту последовательность как поток импульсов, где длительность каждого импульса кодирует для сервоконтроллера информацию о положении. Эти импульсы образуют цикл с определенным периодом или частотой. Частота выражается в циклах в секунду или в герцах. Короткий импульс представляет низкое значение командного сигнала, а длинный импульс – высокое. Контроллер поддерживает постоянную частоту следования импульсов и таким образом формируется рабочий цикл с определенным коэффициентом заполнения (соотношением длительности включенного и выключенного состояний), который меняется в зависимости от команды. Таким образом, длительность импульсов управляет работой сервоприводов.



**Рис. 10.4.** ШИМ для сервоприводов

На графиках на рис. 10.4 по осям  $x$  показано время. На осях  $y$  есть значения  $L$  – логический ноль и  $H$  – логическая единица. На верхнем графике мы видим короткие импульсы. На двух других представлены более длинные импульсы, однако эти графики отличаются по одному важному аспекту. На графике в середине длительность выключенного состояния не меняется, но меняется период.

На нижнем графике период такой же, как и на верхнем, но при этом импульсный интервал больше, а длительность выключенного состояния меньше. Сервоприводам важна длительность импульса, а не их частота, поэтому правильным является нижний график.

В контроллере двигателя нашего робота есть микросхема, отвечающая за ШИМ-модуляцию с фиксированным периодом. Несмотря на то что эта микросхема предназначена для управления светодиодами, с ее помощью можно управлять и другими устройствами, которым требуется ШИМ. Мы можем регулировать длительность выключенного и включенного состояний с фиксированным периодом, что выглядит как нижний график с большей длительностью импульсов (рис. 10.4).

Затронув тему ШИМ, мы переходим к следующему разделу, где реализуем генерацию управляющих сигналов для позиционирования сервопривода на нашем роботе. Давайте подготовим сервопривод, подключим его и посмотрим, как он работает.

## ПОЗИЦИОНИРОВАНИЕ СЕРВОПРИВОДОВ С ПОМОЩЬЮ RASPBERRY PI

Перед тем как мы перейдем к позиционированию сервопривода, необходимо установить **качалку** и убедиться, что она подвижна, а затем подключить привод к плате контроллера двигателя. Качалка сервопривода представляет собой небольшую насадку на вал с одним или несколькими рычагами. Она соединяет вал сервопривода с механизмом, на который он должен воздействовать. На рис. 10.5 показано, как прикрепить качалку к сервоприводу.

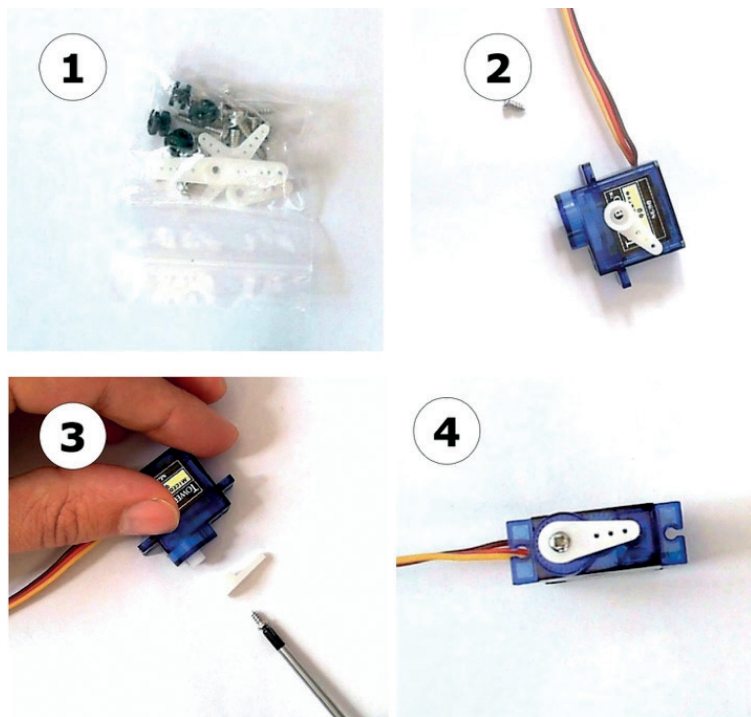


Рис. 10.5. Установка качалки на вал сервопривода

Выполним следующие шаги.

1. Как правило, сервоприводы поставляются с набором дополнительных деталей, содержащим несколько различных типов качалок и винты, с помощью которых эти качалки крепятся к сервоприводу и механизмам, с которыми привод должен взаимодействовать.
2. Для того чтобы прикрепить качалку, используйте короткие маленькие винты, так как длинные могут повредить сервопривод.
3. Прикрепите качалку с одной лопастью к сервоприводу с помощью винта. Вороник с удлиненными концами надевается на выходной вал сервопривода.
4. Сервопривод должен выглядеть так, как показано на этой части рисунка. Не затягивайте винт слишком сильно, так как в дальнейшем вам может понадобиться снова ослабить его.



Далее мы подключим сервопривод к плате контроллера и посмотрим, работает ли он.

На рис. 10.6 показано как подключить сервопривод к плате полнофункционального НАТ-контроллера двигателя на работе. Перед тем как это сделать, выключите робота.

1. Контурная стрелка на этой части рисунка указывает на коннектор сервопривода. Он имеет три контакта (отмечены сплошными стрелками), от которых отходят провода: коричневый – «**земля**» (G), красный – **питающее напряжение** (V), и желтый/оранжевый/белый – **сигнальный** (S).
2. На этой части рисунка мы видим, что на НАТ-плате двигателя есть блок разъемов 3\*4 с пометкой RWM/Servo (отмечен контурной стрелкой). Четыре столбца каналов сервопривода пронумерованы (0, 1, 14 и 15). Каждый канал имеет три контакта, на которых есть метки **GVS** (**земля** (ground), **напряжение** (voltage) и **сигнальный** (signal)). Соедините желтый провод, идущий от коннектора сервопривода, с шиной S, коричневый провод с шиной G, а красный провод будет расположен посередине 0. Подключите сервопривод к каналу 0.

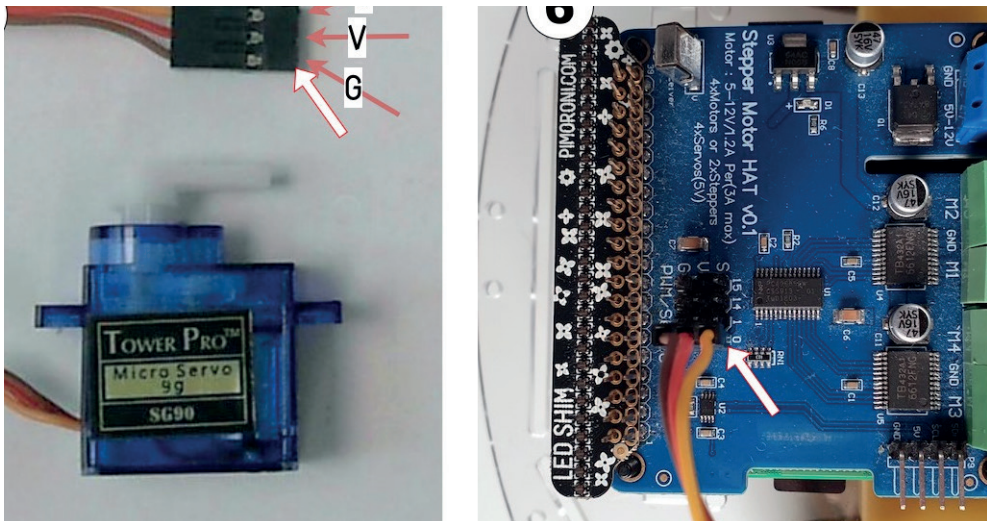


Рис. 10.6. Подключение сервопривода к плате контроллера

Сервоприводы аналогичным образом подключаются и к другим контроллерам (например, PiConZero), но для подключения к некоторым платам может потребоваться пайка. Теперь, когда мы подключили сервопривод, необходимо создать тестовый код.

## Создание тестового кода

Некоторые библиотеки плат могут преобразовывать градус угла поворота напрямую в управляющие импульсы для сервопривода. Мы создадим код с библиотекой, которая позволит выполнять такие вычисления. Результатами

некоторых вычислений могут быть константы, которые не меняются в зависимости от используемых сервоконтроллера и сервопривода. Мы можем сохранять результаты как константы и использовать их повторно.

Сохранить результат таких вычислений можно в переменной. Такой подход дает несколько преимуществ:

- компьютер может быстро и точно вычислять эти константы. Могут возникать ошибки округления, но их будет меньше, чем если бы этим занимался человек;
- нам сразу понятно, откуда взялись те или иные числа и как их вычислять. Понять, как получилось *волшебное число*<sup>10</sup>, которое вычислили на калькуляторе и затем вставили в код, намного сложнее;
- если вы измените коэффициент угла поворота, вычисления автоматически обновятся;
- найденные ошибки легче исправить.

### Совет

Сопровождайте результаты вычислений в коде описательными именами переменных и комментариями. Это поможет вам разобраться в коде и понять смысл вычислений. После завершения любого кода вам наверняка придется перечитывать его еще много раз, поэтому данный принцип применим к любому имени переменной или функции.

Давайте создадим тестовый код, который будет заставлять сервопривод вращать вал в соответствии с введенным пользователем значением (в градусах). Создайте файл `servo_type_position.py`, а затем поместите в него следующий код.

1. Библиотека `Raspi_MotorHAT`, которую мы используем в нашем роботе, имеет модуль `PWM`. Мы импортируем данный модуль и создаем объект для управления сервоприводом. Для прекращения отправки управляющих сигналов на двигатель по-прежнему используем `atexit`:

```
from Raspi_MotorHAT.Raspi_PWM_Servo_Driver import PWM
import atexit
```

2. При настройке ШИМ-устройства необходимо указать его адрес. У нас это то же устройство, которое мы использовали для двигателей. Оно по-прежнему использует шину I2C и имеет тот же адрес:

```
pwm = PWM(0x6f)
```

3. Частота импульсов сервопривода составляет 50 Гц, но мы можем использовать 100 Гц, для того чтобы наши двигатели колес тоже работали. При низкой частоте двигатели будут останавливаться. Данное значение будет временным базисом частоты ШИМ, который мы будем использовать в вычислениях:

<sup>10</sup> Константа, обозначающая ресурс или данные. Может вызвать недоумение, встретившись в коде без комментария. – *Прим. пер.*



```
#Устанавливаем временной базис
pwm_frequency = 100
pwm.setPWMFreq(pwm_frequency)
```

4. Мы примем среднее положение вала за 0°. Большинству сервоприводов для поворота вала на –90° требуется импульс длительностью 1 мс, а для того, чтобы вернуться в центр – 1,5 мс:

```
# Устанавливаем длительность импульса для перехода в среднее положение
# в миллисекундах
servo_mid_point_ms = 1.5
```

Этот фрагмент кода содержит пример использования описательных имен переменных. Переменная с именем `servo_mid_point_ms` более понятна, чем `m` или `p`.

5. Для поворота на 90° требуется импульс длительностью 2 мс; это на 0,5 мс дольше, чем импульс для возврата в среднее положение, и дает нам калибровочное значение для 90°:

```
# Вводим разницу длительности импульса для поворота на 90 градусов
# в миллисекундах
deflect_90_in_ms = 0.5
```

6. Длительность импульса в нашей микросхеме зависит также и от частоты. Микросхема вычисляет длительность импульса как количество тактов<sup>11</sup> за цикл – 4096 тактов (12 бит). При более высокой частоте необходимо большее количество тактов в импульсе для поддержания определенной длительности в миллисекундах. Мы можем рассчитать количество тактов в миллисекунду:

```
# Частота равняется 1, разделенной на период, но, так как длина импульса
# измеряется в миллисекундах, мы можем взять 1000
period_in_ms = 1000 / pwm_frequency
# Микросхема определяет 4096 тактов в каждом периоде
pulse_steps = 4096
# Количество тактов на каждую миллисекунду
steps_per_ms = pulse_steps / period_in_ms
```

7. Теперь, когда мы рассчитали количество тактов в миллисекунду и знаем, какая длительность (также в мс) должна быть у импульса для поворота вала сервопривода на 90°, мы можем вычислить количество тактов на градус. Также это вычисление позволит нам переопределить длительность импульса для перехода в среднее положение в тактах:

```
# Такты на каждый градус
steps_per_degree = (deflect_90_in_ms * steps_per_ms) / 90
# Импульс для перехода в среднее положение в тактах
servo_mid_point_steps = servo_mid_point_ms * steps_per_ms
```

8. Используя эти константы в функции `convert_degrees_to_pwm`, мы можем преобразовать количество тактов, необходимое для поворота на любой угол, в градусы:

<sup>11</sup> Речь идет о встроенном тактовом генераторе микросхемы ШИМ. – Прим. ред.

```
def convert_degrees_to_steps(position):
    return int(servo_mid_point_steps + (position * steps_
per_degree))
```

9. Прежде чем привести вал сервопривода в движение, мы должны убедиться, что система остановлена. Для этого задаем ШИМ значение 4096. Может показаться, что это значение должно задавать большую длительность импульса, но на самом деле оно включает дополнительный бит на плате контроллера, который выключает контакт сервопривода. Выключение контакта «отпускает» сервопривод. Если не сделать этого, сервопривод будет пытаться найти/удержать последнее заданное ему положение до тех пор, пока мы не отключим питание. Аналогично остановке двигателей, для остановки сервопривода мы можем использовать `atexit`:

```
atexit.register(pwm.setPWM, 0, 0, 4096)
```

10. Теперь мы можем запросить ввод данных пользователем в цикле. Функция Python `input` запрашивает пользователя ввести какое-либо значение и сохраняет его в переменной. Для того чтобы использовать эту переменную, необходимо преобразовать ее в целое число:

```
while True:
    position = int(input("Type your position in degrees
(90 to -90, 0 is middle): "))
```

11. Используя предыдущие вычисления, можем преобразовать значение положения в окончательное количество тактов:

```
end_step = convert_degrees_to_steps(position)
```

12. Затем мы должны установить значение нашего импульса в тактах с помощью `pwm.setPWM`. Для этого нам требуется номер канала сервопривода, значение начального шага (примем его за 0) и значение тактов, которое мы вычислили ранее:

```
pwm.setPWM(0, 0, end_step)
```

Теперь вы можете включить робота и загрузить в него код. После запуска кода вам нужно будет ввести число. Введите 0 и нажмите **Ввод**. После этого вы услышите звук вращения вала сервопривода.

Можете попробовать задать другие значения. Не выходите за пределы диапазона от  $-90^\circ$  до  $90^\circ$ , поскольку это может повредить сервопривод. Позже мы создадим код для предотвращения подобных повреждений. Если все работает правильно, вы должны видеть, как вал сервопривода вращается в соответствии с заданными значениями.

## Устранение неполадок

Если сервопривод работает не так, как нужно, попробуйте предпринять следующие действия:

- убедитесь, что шлейф сервопривода подключен к правильным портам и провода расположены правильно. У большинства сервоприводов к контакту **S** должен идти желтый провод;

- если вал дрожит или не может занять правильное положение, это может означать, что батареи разряжены – замените их новыми;
- если при включении ДПТ-колес (двигателей, которые мы обсуждали в прошлых главах) сервопривод перестает работать, это также может быть связано с низким уровнем заряда батареи. Используйте металл-гидридные аккумуляторы.

Прежде чем мы продолжим, необходимо разобраться как НАТ-плата двигателя управляет сервоприводами и ДПТ. Рассмотрим это более детально.

## Управление ДПТ и сервоприводами

В НАТ-плату, которую я рекомендую использовать в этой книге (вы можете выбрать другую), встроена микросхема PCA9685, которая часто используется при создании роботов, подобных нашему. Она представляет собой многоканальный ШИМ-контроллер. Взгляните на рис.10.7, чтобы разобраться, как она работает.

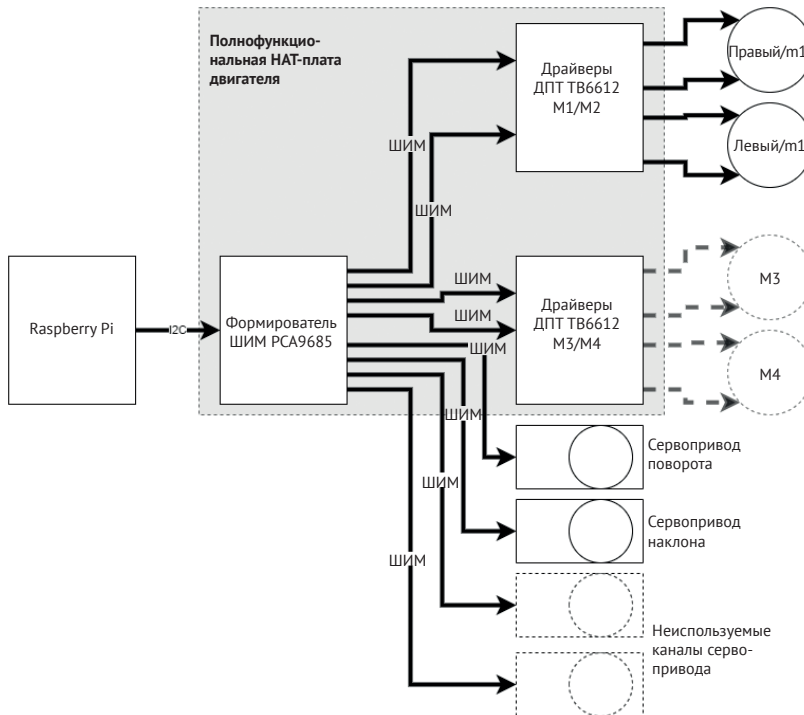


Рис. 10.7. Блок-схема платы двигателя

На рис. 10.7 показана блок-схема платы полнофункционального НАТ-контроллера двигателя. Это не принципиальная электрическая схема, а функциональная блок-схема, но на ней показаны все компоненты и соединения. Слева находится Raspberry Pi, от которого отходит линия, соединяющая его с микросхемой PCA9685. Эта линия представляет собой соединение I2C, которое идет к НАТ-плате (на схеме выделена серым квадратом).

Формирователь ШИМ имеет множество выходов. Восемь из них предназначены для управления драйверами двигателей TB6612. Драйверы имеют выходы питания, которые подходят для ДПТ (или шаговых двигателей). Они являются частью НАТ-платы, поэтому на блок-схеме находятся в сером квадрате. К этим выходам питания мы подключили правый (m1) и левый (m2) двигатели. Также у нас есть место для подключения других двигателей (m3/m4).

Каналы сервопривода представляют собой четыре выхода ШИМ, к которым можно подключаться напрямую. К одному выходу мы подключим сервопривод для поворота, а к другому – для наклона.

Ранее я говорил об управлении микросхемой (формирователем) ШИМ с частотой 100 Гц вместо 50 Гц. Это связано с тем, что, если мы объединим сервоприводы и ДПТ, ко всем выходам ШИМ на микросхеме будет применяться один временной базис, даже при разных рабочих циклах (соотношениях длительности включенного и выключенного состояний).

Теперь, когда мы протестировали основную часть, можем перейти к калибровке сервопривода. Мы найдем положение 0 и убедимся, что поворот на 90° выполняется правильно.

## Калибровка сервоприводов

По качалке сервопривода мы определяем вращение вала серводвигателя. Положение 0 должно быть близким к среднему положению.

1. Сначала необходимо перевести качалку из положения 0 в среднее положение. Отверткой ослабьте винт, выкрутите его, переставьте качалку в среднее положение, а затем снова вкрутите винт. Не затягивайте винт слишком сильно, поскольку в дальнейшем мы будем выкручивать его снова.

### Важное примечание

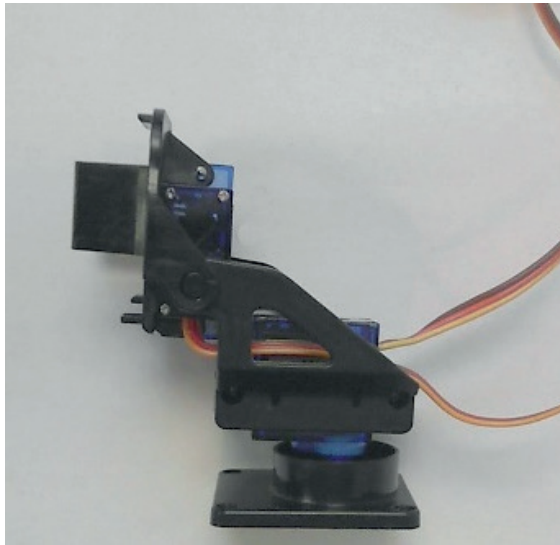
Если вал не может повернуться, в том числе при попытке задать ему положение, выходящее за пределы диапазона, двигатель сервопривода начинает потреблять больше тока для того, чтобы достичь нужного положения. В таком случае сервопривод сильно нагревается, и остановившийся двигатель может выйти из строя.

2. Теперь попробуйте ввести значения 90 и -90. Качалка вашего сервопривода может не достигать этих углов, поскольку заводские настройки сервоприводов могут незначительно отличаться. Далее необходимо увеличить значение `deflect_90_in_ms`, чтобы отрегулировать диапазон двигателя. Постепенно увеличивайте значение на 0,1, поскольку более резкое увеличение может привести к повреждению сервопривода.
3. После того как вы откалибруете сервопривод, перед следующим шагом необходимо перевести его в положение 0. Вы можете сделать это, подключив второй сервопривод к каналу 1, а затем поменяв первый параметр каждого вызова метода `pwm.setPWM` с 0 на 1.

Итак, мы создали тестовый код, а затем протестировали сервоприводы и два канала. Теперь можно использовать сервоприводы для реализации механизма поворота и наклона, который мы будем использовать для позиционирования датчиков.

## СОЗДАНИЕ МЕХАНИЗМА ПОВОРОТА И НАКЛОНА

В этом разделе мы соберем механизм поворота и наклона с сервоприводами и добавим его к нашему роботу. Этот механизм будет выполнять роль «головы» робота, на которую можно будет устанавливать датчики. В **механизме поворота и наклона** (рис. 10.8) сервоприводы позиционируют датчик (или другое устройство) по двум осям.



**Рис. 10.8.** Механизм поворота и наклона, собранный из обычного набора

Одна часть механизма (*поворот*) обеспечивает поворот влево и вправо, а другая (*наклон*) – наклон вниз и вверх.

Наборы, подобные показанному на рис. 10.8, можно приобрести на Aliexpress. Возможно, сервоприводы придется докупить отдельно. Существуют разные виды механизмов поворота и наклона. Убедитесь, что ваш набор предусматривает подключение двух сервоприводов. Для этого обратитесь к документации производителя. Нам необходимо собрать набор, установить получившийся механизм на нашего робота и подключить к контроллеру.

На рис. 10.9 показано, как выглядит блок-схема робота с сервоприводами.

Блок-схема на рис. 10.9 – это расширенная схема из предыдущей главы, к которой добавили сервоприводы, используемые в механизме поворота и наклона. Они подключаются к НАТ-плате двигателя.

Теперь, когда вы увидели, какое место сервоприводы занимают в блок-схеме робота, настало время собрать механизм поворота и наклона.

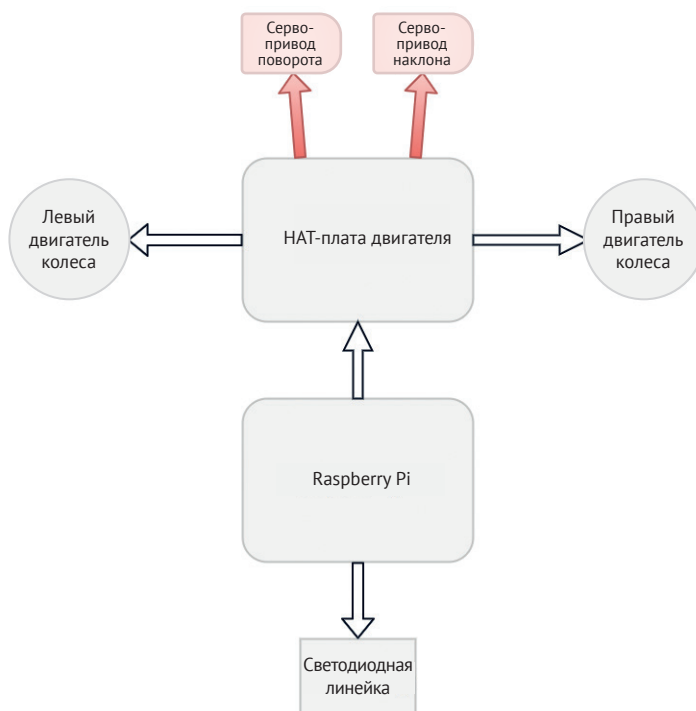


Рис. 10.9. Блок-схема робота с сервоприводами

## Сборка механизма

Вам понадобится набор для сборки механизма поворота и наклона, отвертка и резак. На рис. 10.10 показано, что входит в набор.

Обратите внимание на пластиковые компоненты, у которых на рис. 10.10 есть подписи. Этими названиями я буду оперировать при сборке. Рядом с этими компонентами вы можете видеть винты, которые также входят в набор. Обычно наборы поставляются с самонарезающими винтами M2. Убедитесь, что они есть в вашем наборе.

### Важное примечание

Пластиковые компоненты достаточно хрупкие, поэтому с ними нельзя работать без защитных очков. Мелкие и острые кусочки пластика могут разлететься по комнате и причинить вред окружающим людям. На этом этапе я настоятельно рекомендую надеть защитные очки!

Теперь, когда вы подготовились, можно перейти к сборке основания. Инструкция по сборке основания показана на рис. 10.11.

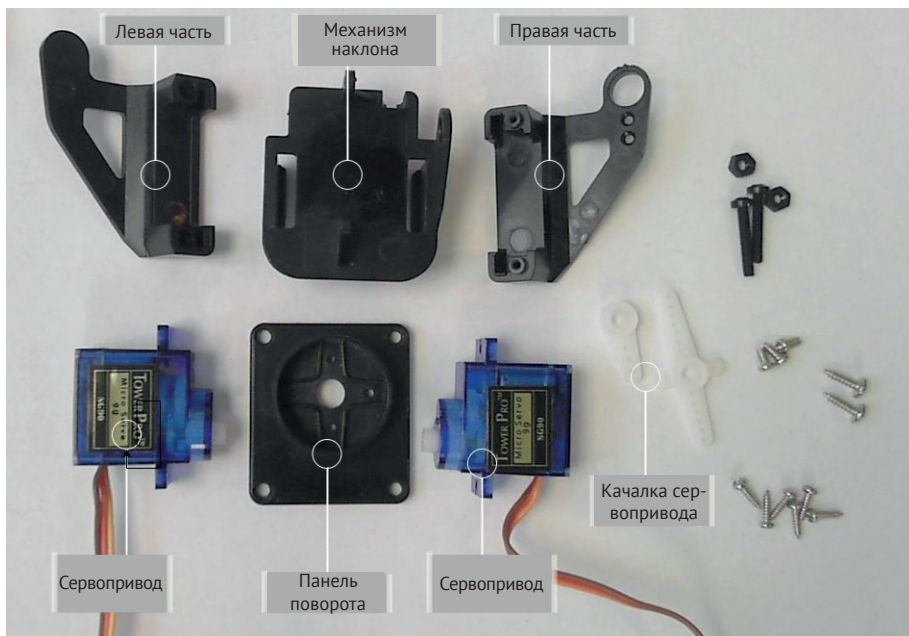


Рис. 10.10. Состав набора механизма поворота и наклона

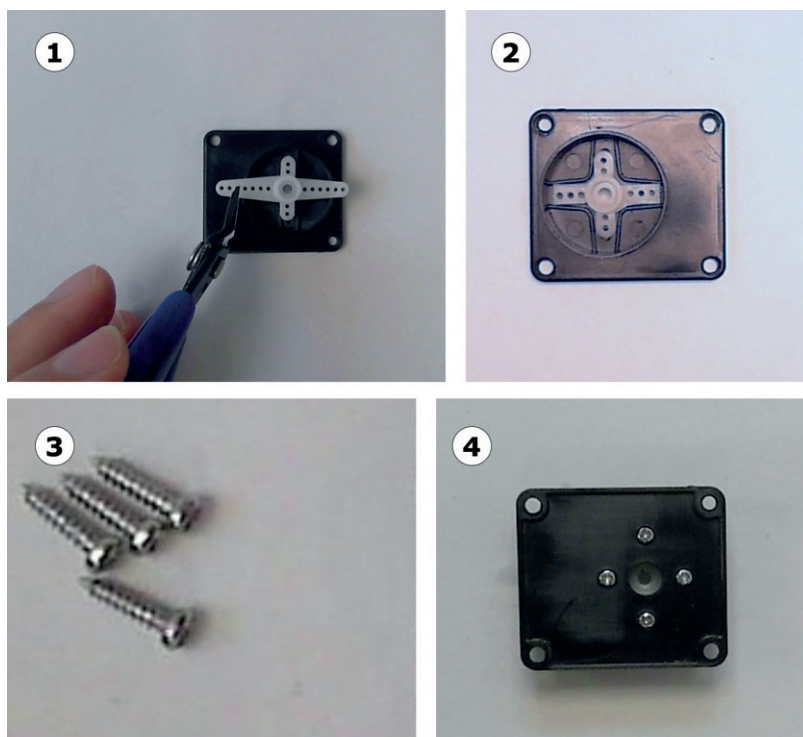


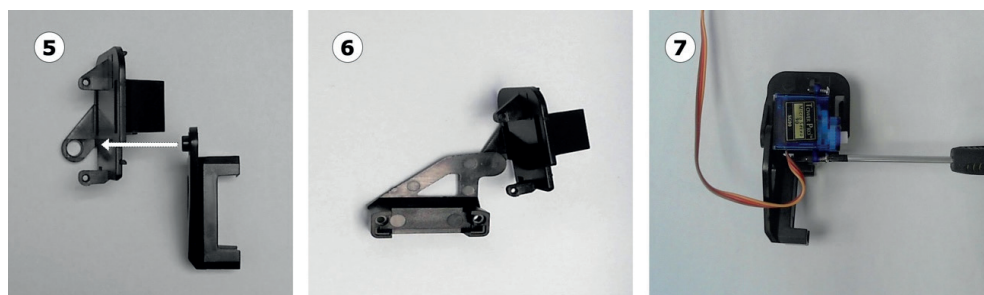
Рис. 10.11. Сборка основания



1. Обрежьте крестообразную качалку по размеру выемки в основании. Резаком укоротите рычаги и сделайте их немного тоньше.
2. Качалка должна находиться в основании таким образом, чтобы лопасти были в углублении, а фланец качалки был обращен в противоположную сторону от основания.
3. Возьмите четыре самонарезающих винта M2.
4. Закрепите качалку винтами. Обратите внимание, что некоторые качалки могут закрепляться на двух винтах (по горизонтали или вертикали). Двух будет достаточно, но четыре винта обеспечат большую надежность.

Наше основание готово. Теперь соберем левую часть.

Для этого обратитесь к рис. 10.12 и выполните следующие шаги.

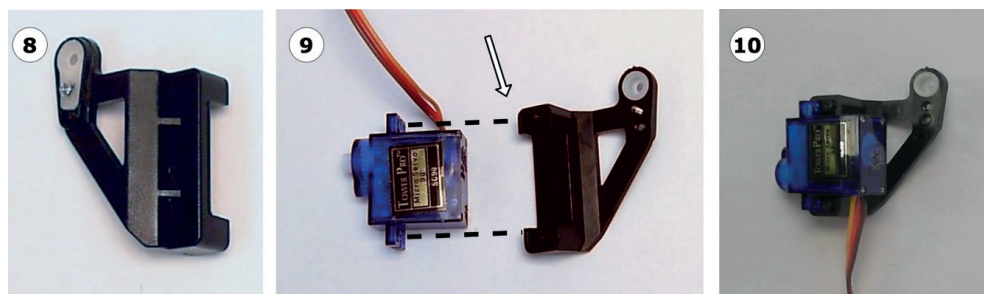


**Рис. 10.12.** Сборка левой части механизма и планки наклона

1. Совместите выступ с отверстием на механизме наклона, как показано на рис. 10.12.
2. Вставьте выступ в отверстие. Это крепление должно надежно держаться.
3. Возьмите один сервопривод и два винта с широкими шляпками. Сервопривод устанавливается на две опоры на планке механизма наклона и после привинчивания левая часть остается зафиксированной. Перед тем как вкрутить винт, убедитесь, что вал сервопривода совмещен с выступом и отверстием.

Отлично! Теперь перейдем к правой части.

Для этого обратитесь к рис. 10.13 и выполните следующие шаги.

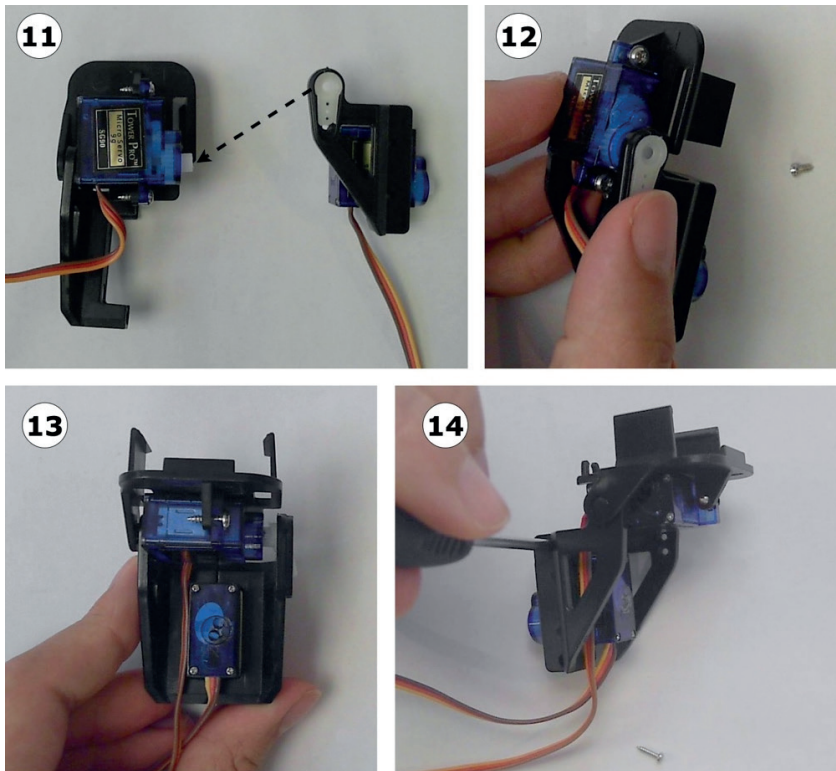


**Рис. 10.13.** Сборка правой части

1. Для правой части вам понадобится еще одна качалка, на этот раз только с фланцем и одним прямым рычагом. Как показано на рис. 10.13, качалку сервопривода необходимо обрезать, чтобы она входила в выемку на правой части. Прикрутите качалку винтом M2. Она должна находиться в передней части механизма.
2. Переверните собранную часть и вставьте другой сервопривод (сервопривод поворота) в слоты, как показано в этой части рисунка.
3. Его вал должен быть обращен вниз, как показано в этой части рисунка. Соответственно, сервопривод также будет обращен вниз.

Наш следующий шаг – соединить левую и правую части.

Для этого обратитесь к рис. 10.14 и выполните следующие шаги.

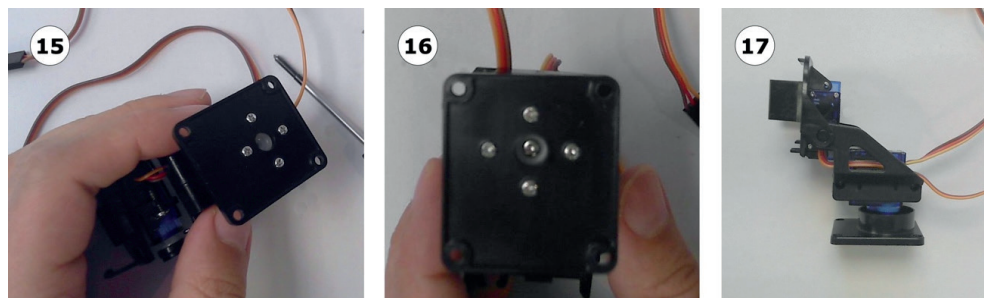


**Рис. 10.14.** Соединение левой и правой частей

1. На рис. 10.14 показано, как соединить левую и правую части. При соединении деталей фланец качалки правого сервопривода должен охватывать сервопривод наклона, который вы прикрутили к планке наклона.
2. Сервопривод поворота из левой части должен войти в соответствующую выемку.
3. С помощью короткого винта закрепите фланец качалки правого сервопривода на сервоприводе наклона, удерживая основание механизма наклона в вертикальном положении.
4. Соедините части с помощью маленьких тонких винтов.

Все почти готово. Последний этап – установка механизма на основание, которое мы собрали ранее.

Для этого обратитесь к рис. 10.15 и выполните следующие шаги.



**Рис. 10.15.** Установка механизма на основание

1. Наденьте фланец качалки сервопривода, закрепленного болтами на основании, на ось сервопривода поворота. Сопоставьте длинную ось основания с нижней частью механизма так, чтобы они образовывали прямую линию.
2. Используйте один из коротких винтов, чтобы соединить воротник и сервопривод.
3. В этой части рисунка показан собранный механизм поворота и наклона.

Вы увидели, как устроен механизм поворота и наклона с сервоприводами. Сборка подобных конструкций помогает понять, как работают механизмы и сервоприводы. Теперь, когда мы собрали механизм, необходимо установить его на робота.

## Установка механизма поворота и наклона на робота

Механизм должен стать частью робота. Нам нужно установить его на шасси и подключить к контроллеру, чтобы последний мог отправлять сигналы.

Следуйте рис. 10.16 и инструкциям ниже.

1. Для установки вам понадобятся два длинных болта и две гайки.
2. Вставьте болты в короткий конец основания механизма поворота и наклона так, чтобы они были направлены вниз.
3. В шасси робота, которое я рекомендовал, есть слот на передней части, пригодившийся для установки датчика линии. Этот слот идеально подходит для установки механизма поворота и наклона с помощью винтов. На другом шасси может потребоваться разметить и просверлить отверстия для винтов.
4. Накрутите гайки и затяните их с нижней стороны робота.
5. Подключите сервоприводы. Сервопривод наклона (обеспечивающий движение вверх и вниз) нужно подключить к сервоканалу 0, а сервопривод поворота (обеспечивающий движение влево и вправо) – к сервоканалу 1.

Поздравляю! Установка механизма завершена, и робот готов. Теперь мы можем создать код для управления «головой» нашего робота. Итак, приступим.

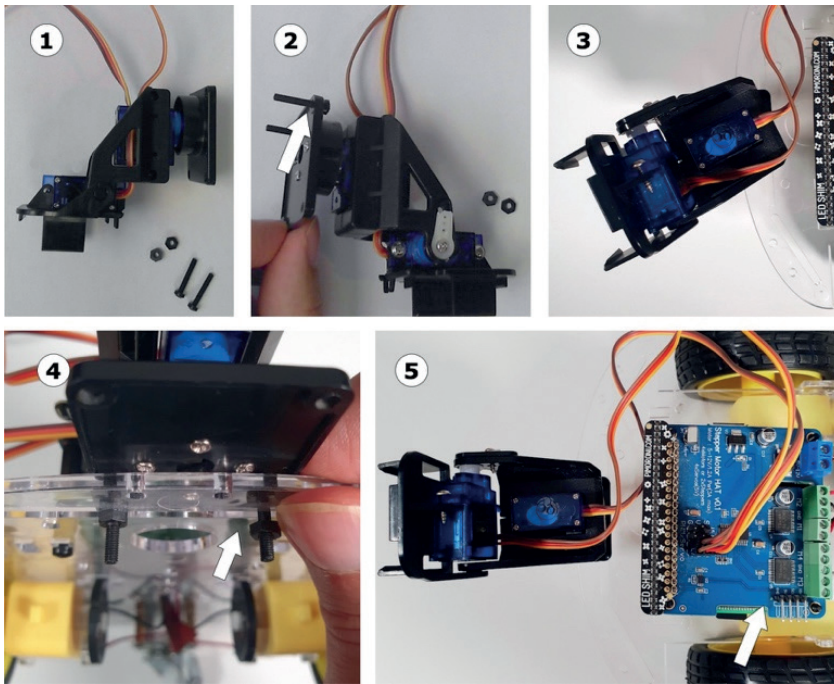


Рис. 10.16. Установка механизма поворота и наклона на робота

## РАЗРАБОТКА КОДА ДЛЯ МЕХАНИЗМА ПОВОРОТА И НАКЛОНА

Наш код для механизма поворота и наклона состоит из нескольких уровней. Мы создадим класс `Servos` и поместим в него результаты предыдущих вычислений. Добавим экземпляр класса `Servos` в наш класс `Robot`, чтобы через него управлять сервоприводами поворота и наклона.

### Создание объекта `Servos`

В этот класс мы поместим данные преобразования угла в импульс для управления сервоприводом, а также некоторые подробности описания нашего сервоконтроллера, такие как номера каналов. Мы создаем класс `Servos` в файле с именем `servos.py`.

1. Файл `servos.py` начинается с импорта, за которым следует конструктор (функция `__init__`):

```
from RPi_MotorHAT.Raspi_PWM_Servo_Driver import PWM

class Servos:
    def __init__(self, addr=0x6f, deflect_90_in_ms=0.6):
```

Здесь мы видим адрес ШИМ-устройства. Также здесь находится параметр отклонения/калибровки, который называется `deflect_90_in_ms`. Его необходимо заменить значением, полученным при калибровке сервоприводов.

2. Затем необходимо добавить комментарий, чтобы при использовании класса `Servos` мы понимали, о чем идет речь. В некоторых редакторах код текст комментария будет отображаться в диалоговой справке для нашего класса.

```
"""addr: Адрес i2c микросхемы ШИМ.
deflect_90_in_ms: устанавливаем значение, полученное при калибровке
сервопривода.
это значение соответствует повороту на 90 градусов
(длительность соответствующего импульса в миллисекундах)."""
```

Строки в тройных кавычках, объявленные в начале конструктора, представляют собой соглашение, которое в Python называется **строка документации (docstring)**. Любая строка, объявленная в верхней части функции, метода, класса или файла, понимается как особый тип комментария, который во многих редакторах отображается как справка для библиотеки. Это полезно на любом уровне библиотеки. Соглашение об использовании строки документации дополнит все пояснительные комментарии, которые мы перенесем из тестового кода.

3. Следующий фрагмент с методом `__init__` должен показаться вам знакомым. Он устанавливает все результаты вычислений, произведенных в шагах 3–7 в разделе «Создание тестового кода». Мы храним объект ШИМ в `self._pwm`. Мы сохраняем несколько переменных в `self` для дальнейшего использования. При этом остальные переменные содержат результаты промежуточных вычислений:

```
self._pwm = PWM(addr)
# Устанавливаем временной базис
pwm_frequency = 100
self._pwm.setPWMFreq(pwm_frequency)
# Устанавливаем длительность импульса для перехода в среднее положение
# в миллисекундах.
servo_mid_point_ms = 1.5
# Частота равняется 1, разделенной на период, но, так как длина импульса
# измеряется в миллисекундах, мы можем взять 1000.
period_in_ms = 1000 / pwm_frequency
# Микросхема отводит 4096 тактов на каждый период
pulse_steps = 4096
# Количество тактов на каждую миллисекунду.
steps_per_ms = pulse_steps / period_in_ms
# Такты на каждый градус.
self.steps_per_degree = (deflect_90_in_ms *
steps_per_ms) / 90
# Импульс для перехода в среднее положение в тактах.
self.servo_mid_point_steps = servo_mid_point_ms *
steps_per_ms
```

4. В последней части метода `__init__` мы создаем переменную `self._channels`. Она позволяет нам использовать номера каналов 0, 1, 2 и 3 и сопоставить их с номерами на плате:

```
# Карта распределения каналов
self._channels = [0, 1, 14, 15]
```



5. Далее нам необходима функция безопасности, которая выключит сервоприводы. Для этого необходимо прекратить отправку импульсов и «отпустить» сервоприводы, что не позволит превысить нагрузку и защитит двигатели от повреждения. Эта функция использует тот же трюк, который представлен в *шаге 9* раздела «Создание тестового кода». Она устанавливает время запуска на 0 и 4096 для флага выключения, ввиду чего импульс не генерируется:

```
def stop_all(self):
    # 0 означает отсутствие импульса, 4096 устанавливает бит, который
    # выключает выход (бит OFF).
    off_bit = 4096
    self._pwm.setPWM(self.channels[0], 0, off_bit)
    self._pwm.setPWM(self.channels[1], 0, off_bit)
    self._pwm.setPWM(self.channels[2], 0, off_bit)
    self._pwm.setPWM(self.channels[3], 0, off_bit)
```

6. Затем идет функция преобразования, которую мы уже видели в *шаге 8* раздела «Создание тестового кода». Теперь она локализована в классе. Мы будем использовать эту функцию только для внутреннего преобразования. В соответствии с соглашениями Python перед ней ставится нижнее подчеркивание (префикс):

```
def _convert_degrees_to_steps(self, position):
    return int(self.servo_mid_point_steps + (position *
        self.steps_per_degree))
```

7. Также нам нужен метод для позиционирования сервопривода на заданный угол. Для этого потребуются номер канала и угол. В этом методе я использовал еще одну строку документации, для того чтобы описать его работу и ограничения значений:

```
def set_servo_angle(self, channel, angle):
    """position: Положение от центра в градусах. От -90 до 90"""
```

8. Следующая пара строк проверяет вводимые значения. Они ограничивают значение угла, что предотвращает поворот вала на значение, выходящее за пределы допустимого диапазона. Если значение выходит за пределы диапазона, происходит вызов исключения Python. Исключение передает проблему каждой вызывающей системе до тех пор, пока одна из них не решит ее. Как правило, решение проблемы заключается в завершении кода:

```
#Проверка
if angle > 90 or angle < -90:
    raise ValueError("Угол за пределами допустимого диапазона")
```

9. В последних двух строках этого метода задается положение:

```
# Задаем положение
off_step = self._convert_degrees_to_steps(angle)
self._pwm.setPWM(self.channels[channel], 0, off_step)
```

Вы можете найти полный код на <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter10/servos.py>.

Теперь класс готов тому, чтобы добавить его в Robot. Этим мы займемся в следующем разделе.

## Добавляем сервоприводы в класс Robot

Прежде чем мы сможем использовать класс Servos в поведенческих скриптах, его нужно добавить в наш файл robot.py и задать сервоприводам конкретные задачи. Таким образом, поведенческие скрипты будут подходить и для других роботов с иначе настроенными механизмами поворота и наклона (для этого достаточно будет заменить класс Robot на соответствующий).

1. Нам нужно сделать необходимую доработку класса Robot. В файле robot.py добавим импорт нового класса после импорта leds:

```
import leds_led_shim
from servos import Servos
...
```

2. Затем объект servos необходимо настроить в конструкторе для Robot. Для этого нужно передать адрес:

```
class Robot:
    def __init__(self, motorhat_addr=0x6f):
        # Настройка HAT-платы в соответствии с переданным адресом
        self._mh = Raspi_MotorHAT(addr=motorhat_addr)

        # получение локальных переменных для каждого двигателя
        self.left_motor = self._mh.getMotor(1)
        self.right_motor = self._mh.getMotor(2)

        # Настроить светодиоды
        self.leds = leds_led_shim.Leds()

        # Настроить сервоприводы для механизма поворота и наклона
        self.servos = Servos(addr=motorhat_addr)

        # обязательно остановим двигатели после завершения кода
        atexit.register(self.stop_all)
    ...
```

3. Мы должны остановить двигатели после завершения кода. Для этого используем метод stop\_all:

```
def stop_all(self):
    self.stop_motors()
    # Сбрасываем значения светодиодов
    self.leds.clear()
    self.leds.show()
    # Сбрасываем сервоприводы
    self.servos.stop_all()
    ...
```

4. Завершающий этап в классе robot – сопоставление заданных значений для механизма поворота и наклона с реальными сервоприводами:

```
def set_pan(self, angle):
    self.servos.set_servo_angle(1, angle)
def set_tilt(self, angle):
    self.servos.set_servo_angle(0, angle)
```



Теперь у нашего объекта `Robot` есть методы для взаимодействия с сервоприводами механизма поворота и наклона, который установлен на шасси робота. Итак, мы создали уровень, который может быть использован в поведенческих скриптах, и реализовали определенные элементы управления. В следующем разделе разработаем поведенческий скрипт, который заставит «голову» нашего робота выполнять движение по окружности.

## Разработка скрипта для движения «головы» робота по окружности

Сейчас мы разработаем скрипт, который заставит механизм поворота и наклона очерчивать небольшую окружность (диаметром около 30°). Данный скрипт покажет, как взаимодействует механизм и код. С помощью кода мы реализуем повторяющееся движение по окружности на основе временного базиса – текущего времени. Мы используем временной базис, чтобы очертить окружность.

1. Создайте новый файл. Я предлагаю назвать его `circle_pan_tilt_behavior.py`:
2. Начнем с импорта объекта `Robot`, библиотеки `math` и некоторых параметров времени:

```
from time import sleep
import math

from robot import Robot
```

Библиотека `math` потребуется нам для вычисления синуса и косинуса.

3. Наш поведенческий скрипт содержит локальные данные, поэтому мы переместим их в отдельный класс. Конструктор (метод `__init__`) принимает объект `Robot`:

```
class CirclePanTiltBehavior:
    def __init__(self, the_robot):
        self.robot = the_robot
```

4. В сущности, такой поведенческий сценарий представляет собой анимацию, поэтому у него есть определенные параметры времени и количество *кадров* (позиций) для каждого круга. Мы используем переменную `frames_per_circle`, чтобы задать количество шагов за круг:

```
self.current_time = 0
self.frames_per_circle = 50
```

5. Математические функции используют радианы. Полный круг равен  $2\pi$  радианам. Мы делим это на переменную `frames_per_circle` и получаем множитель `radians_per_frame`. Путем умножения этого множителя на текущий кадр можем получить радианную меру угла для окружности:

```
self.radians_per_frame = (2 * math.pi) / self.
frames_per_circle
```

Мы работаем сразу радианами, а не с градусами, поскольку, переводя градусы в радианы и производя деление на количество кадров для круга, в итоге мы получили бы постоянный множитель и вернулись к `radians_per_frame`.

6. Окружность имеет радиус, который показывает, насколько сервоприводы отклоняются от среднего положения:

```
self.radius = 30
```

7. Следующий метод в нашем поведенческом скрипте – это метод `run`. С его помощью мы помещаем поведенческий скрипт в цикл `while True`. Таким образом, код будет выполняться до тех пор, пока пользователь не остановит его:

```
def run(self):
    while True:
```

8. После запуска нашего поведенческого скрипта мы получаем `current_time` и преобразовываем его в количество кадров, используя операцию по модулю (остаток от деления) для `frames_per_circle`. Модуль устанавливает ограничение от нуля до количества кадров:

```
frame_number = self.current_time % self.
frames_per_circle
```

9. Затем преобразовываем значение переменной `frame_number` в радианы (позиции на окружности), умножая его на значение `radians_per_frame`. В результате умножения мы получаем значение `frame_in_radians`:

```
frame_in_radians = frame_number * self.
radians_per_frame
```

10. Для очерчивания окружности нам необходимо применить формулу, которая приравнивает одну из осей к значению косинуса угла, умноженному на радиус, а другую к синусу угла, умноженному на радиус. Производим расчет и передаем данные на сервоприводы:

```
self.robot.set_pan(self.radius * math.
cos(frame_in_radians))
self.robot.set_tilt(self.radius * math.
sin(frame_in_radians))
```

11. Для того чтобы у двигателей было время занять необходимую позицию, мы задаем небольшую паузу (`sleep()`). Затем добавляем к текущему времени единицу:

```
sleep(0.05)
self.current_time += 1
```

12. Полностью метод `run` (шаги 7–11) выглядит следующим образом:

```
def run(self):
    while True:
        frame_number = self.current_time % self.
frames_per_circle
        frame_in_radians = frame_number * self.
radians_per_frame
        self.robot.set_pan(self.radius * math.
cos(frame_in_radians))
        self.robot.set_tilt(self.radius * math.
sin(frame_in_radians))
        sleep(0.05)
        self.current_time += 1
```

13. Наконец, мы запускаем поведенческий скрипт:

```
bot = Robot()
behavior = CirclePanTiltBehavior(bot)
behavior.run()
```

Итак, мы создали класс `Servos`, посредством которого теперь можем управлять механизмом поворота и наклона. Мы создали код, заставляющий сервоприводы работать в режиме анимации. В следующем разделе применим похожую стратегию и реализуем реальные повороты и наклоны.

## Запуск сценария

Для запуска необходимо отправить файлы `servos.py`, `robot.py` и `circle_pan_tilt_behavior.py` на Raspberry Pi через SFTP. После этого введите на Pi `python3 circle_pan_tilt_behaviour.py`, и вы увидите, что все работает.

Мы создали демонстрационный код, который показывает, что устройство работает. В дальнейшем, оснастив устройство камерой, его можно будет использовать для отслеживания лиц. Важную роль в созданном нами коде играют кадры, на основе которых мы реализовали анимацию движения. Такой подход обеспечивает выполнение роботом плавных заданных движений путем управления последовательностью небольших перемещений.

## Устранение неполадок

Если вы столкнулись с неполадками, воспользуйтесь советами, представленными ниже:

- протестируйте сервоприводы в соответствии с разделом «Создание тестового кода». Проверьте, чтобы при запуске кода вал сервопривода вращался (для этого не придется разбирать механизм поворота и наклона);
- если при запуске этого кода возникают ошибки, убедитесь, что вы ввели `python3`, и проверьте код на наличие опечаток;
- если код не импортирует файлы, убедитесь, что вы создали копии всех файлов и установили/настроили все библиотеки в предыдущей главе;
- если приводы отклоняются до упора и заклиниваются, возможно, вы пропустили этап калибровки. Для того чтобы это исправить, открутите болты, снимите качалки и приведите валы каждого привода в положение 0 посредством тестового кода из раздела «Создание тестового кода». Затем оставьте их в среднем положении, установите качалки и закрутите болты;
- если сервоприводы не двигаются, проверьте, правильно ли они подключены. Черный провод должен идти к контакту G, красный – к V, а желтый сигнальный – к S. Убедитесь, что сервоприводы подключены к каналам 0 и 1 на плате двигателей (как этого ожидает код);
- убедитесь, что в проводах нет обрывов, а изоляция не повреждена. Однажды я заказал несколько сервоприводов, и оказалось, что они из бракованной партии, поэтому мне пришлось вернуть их обратно. Провода должны быть без повреждений;
- убедитесь, что коннекторы вставлены надежно. В противном случае сигнал не будет поступать на сервоприводы;

- повторяюсь: при разряженных батареях сервопривод не будет работать правильно (он не сможет достичь заданного положения и будет дрожать).

Эти советы призваны устранить часто возникающие проблемы при реализации движения по окружности. Теперь «голова» робота должна медленно двигаться по окружности с небольшим диаметром. В следующей главе мы будем использовать эту систему для распознавания лиц.

## РЕАЛИЗАЦИЯ СКАНИРУЮЩЕГО СОНАРА

Сочетание датчика расстояния из главы 8 с механизмом поворота и наклона позволит нам реализовать интересный эксперимент. Если мы прикрепим датчик к «голове» робота и будем медленно перемещать его в определенном направлении (например, в направлении поворота), то реализуем охват датчиком определенной области. Мы разработаем для этого код на Python и создадим небольшую карту объектов, находящихся перед роботом.

Подобные системы используются в продвинутых роботах (например, от Boston Dynamics) и в беспилотных автомобилях. Лидары (LIDAR) и радары (RADAR) выполняют описанную выше функцию намного быстрее, поскольку используют лазерные или радиочастотные импульсы, испускаемые быстро вращающейся активной головкой. Несмотря на то что лидары начинают появляться на любительском рынке, они по-прежнему стоят довольно дорого.

Мы представим результат работы нашей системы в виде полярной диаграммы. Она представляет собой круговую систему координат, где по оси  $x$  показаны точки положения на окружности (в радианах – кратных числах  $\pi$ ), а по оси  $y$  показывается насколько далеко от центра эти точки находятся. Чем больше значение – тем дальше от центра. Такое представление отлично подходит для нашей системы, поскольку сервопривод позиционируется в соответствии со значениями углов, и мы получаем данные о расстоянии. Для того чтобы построить полярную диаграмму, нам необходимо будет перевести результаты из градусов в радианы.

## Установка датчика

На этом этапе нам необходимо удлинить провода датчика и установить его на механизм поворота и наклона. Для начала выключите робота.

Для того чтобы переместить датчик, выполните следующие шаги, руководствуясь рис. 10.17.

1. Сначала нужно извлечь один из датчиков из крепления на передней части робота. Я извлек левый.
2. На механизме поворота и наклона найдите небольшие отверстия.
3. Возьмите два отрезка одножильного провода или нейлоновые кабельные стяжки. Они должны быть длиной около 18 см.
4. Проденьте провода через отверстия на обеих сторонах механизма, как показано на рис. 10.18.

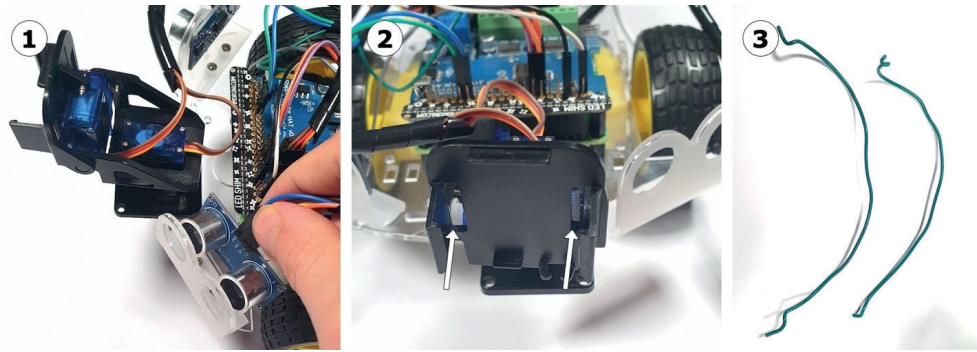


Рис. 10.17. Извлечение датчика и подготовка двух проводов

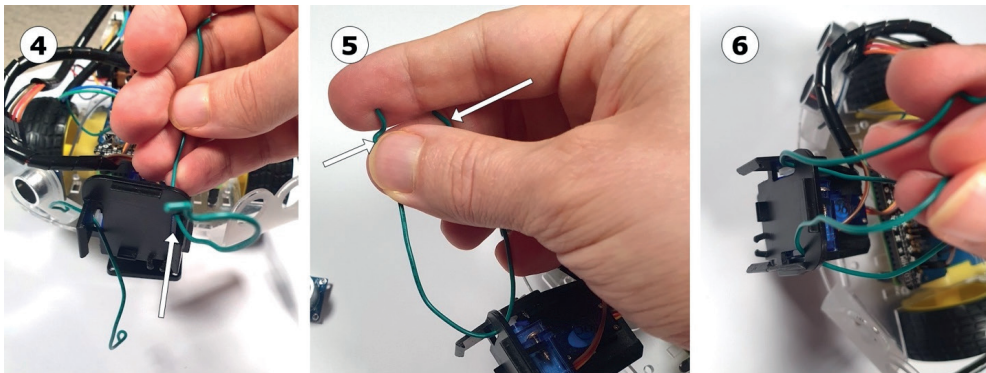


Рис. 10.18. Шаги 4–6. Продеваем провода через отверстия

5. Слегка загните концы, почти соединив их, чтобы закрепить провод в отверстии.
6. То же самое проделайте с другой стороны.
7. Установите датчик на нужное место и обогните левый провод вокруг передней части датчика.
8. Теперь протяните провод под большим круглым элементом датчика (ультразвуковым приемопередатчиком), как показано белой стрелкой на рис. 10.19.

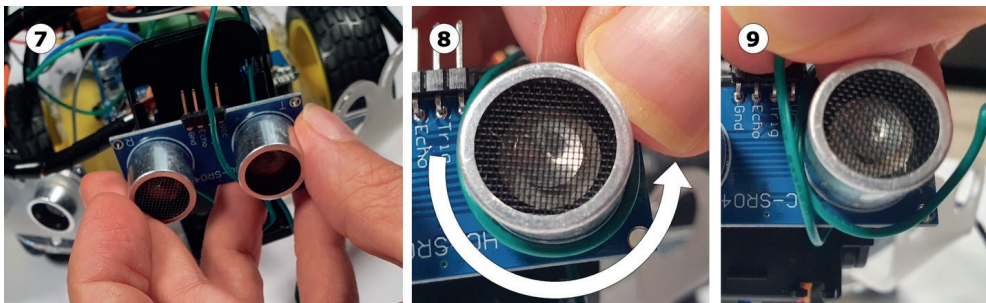
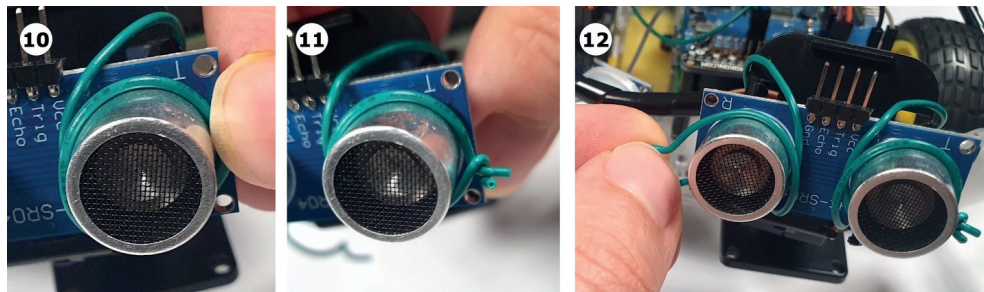


Рис. 10.19. Протягиваем провод под элементом датчика



9. Делая круг, протяните провод над датчиком.
10. Оберните этот провод вокруг левого приемопередатчика и соедините два конца провода.
11. Скрутите верхний и нижний концы вместе, как показано на рис. 10.20.



**Рис. 10.20.** Скручиваем концы провода и повторяем шаги с правой стороны

12. Повторите шаги 8–11 для правой стороны.
13. Теперь датчик временно закреплен на механизме поворота и наклона как показано на рис. 10.21. Можем подключить его.



**Рис. 10.21.** Датчик временно закреплен на механизме

Вам может потребоваться удлинить соединительные провода, как показано на рис. 10.22. Скорее всего, у вас остались соединительные провода с разъемами «гнездо–штекер» из прошлых глав, которые вы как раз можете использовать для этой цели. Будьте внимательны и убедитесь, что соединяете провода правильно. По возможности используйте провода одинакового цвета.

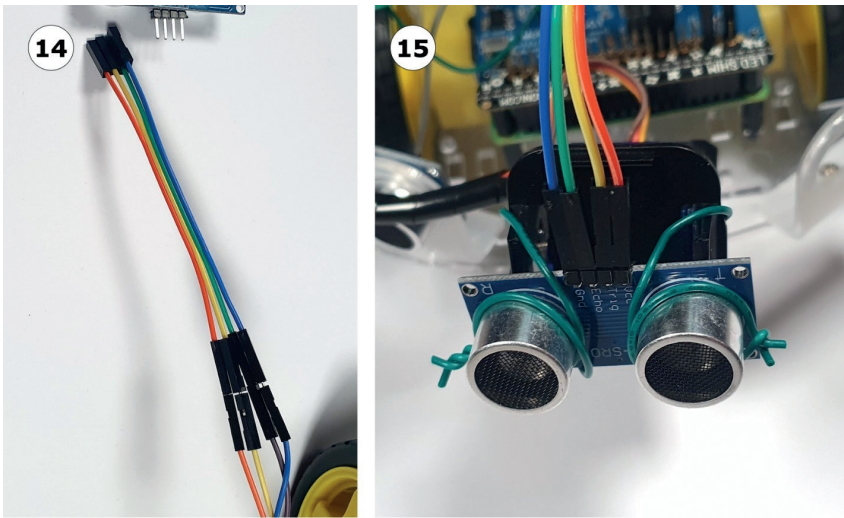


Рис. 10.22. Удлиняем и соединяем провода

14. Теперь подключите провода к датчику. Для справки обратитесь к главе 8.

Прежде чем продолжить, я предлагаю запустить код `test_distance_sensor.py` из главы 8 и проверить, работает ли датчик.

Теперь, когда вы установили датчик, он будет приводиться в движение сервоприводами вместе со всем механизмом. Мы будем управлять им посредством кода, но сначала убедимся, что на Raspberry Pi есть нужные инструменты.

## Установка библиотек

В нашем коде для представления данных мы будем использовать инструмент Python `matplotlib`. Он создает полярную диаграмму – график, исходящий из точки в радиальном направлении (возможно, вы видели подобное в фильмах). Для этого необходимо установить библиотеку `Matplotlib` (и ее зависимости) на Raspberry Pi.

Чтобы получить пакеты для `Matplotlib` в клиенте SSH (PuTTY), введите следующее:

```
$ sudo apt update
$ sudo apt install libatlas3-base libgfortran5
```

Затем установите саму библиотеку:

```
$ pip3 install matplotlib
```

Установка `Matplotlib` и необходимых вспомогательных пакетов может занять некоторое время. После установки мы можем приступить к созданию кода.

## Поведенческий скрипт

Чтобы построить нашу диаграмму и получить данные, код должен переместить датчик сначала в одну сторону, а затем пошагово в другую (например, по 5°),



фиксируя данные каждой точки. В этом примере представлено совместное использование датчика и сервопривода. Также мы поговорим о других способах представления показаний датчика.

Создайте файл с именем `sonar_scan.py` и выполните следующие действия.

1. Начнем с импорта. Нам необходимо импортировать `time`, чтобы управлять временем работы приводов и датчиков, `math`, чтобы преобразовывать градусы в радианы, и `matplotlib` для отображения скрипта и реализации взаимодействия робота с объектом `robot`:

```
import time
import math
import matplotlib.pyplot as plt
from robot import Robot
```

2. Затем необходимо задать настраиваемые параметры. Здесь вы можете свободно экспериментировать с разными скоростью и радиусом поворота:

```
start_scan = 0
lower_bound = -90
upper_bound = 90
scan_step = 5
```

3. Далее мы инициализируем объект `Robot` и приводим механизм наклона в строго горизонтальное положение:

```
the_robot = Robot()
the_robot.set_tilt(0)
```

4. Нужно подготовить место для хранения данных сканирования. Мы будем использовать словарь, сопоставляющий положения «головы» робота в градусах и соответствующие результаты измерений:

```
scan_data = {}
```

5. Цикл сканирования начинается от нижней границы и увеличивается на шаг сканирования до верхней границы. Таким образом, мы получаем широкий диапазон измерений, полученных в каждом положении поворота «головы» (`facing`):

```
for facing in range(lower_bound, upper_bound, scan_step):
```

6. Для получения измерений необходимо задать датчику определенное положение и дождаться, пока сервопривод переместит его и датчик получит показания. Мы меняем знак результата измерения в примере, поскольку сервопривод перемещает датчик в направлении, противоположном полярной диаграмме:

```
the_robot.set_pan(-facing)
time.sleep(0.1)
```

7. Затем сохраняем измеренное датчиком расстояние (в сантиметрах) в данных сканирования для каждого измерения, используя угол поворота в качестве ключа:

```
scan_data[facing] = the_robot.left_distance_sensor.
distance * 100
```

8. Следующий цикл преобразовывает результаты измерения в радианы. Сервопривод работает со значениями в градусах, но особенность библиотеки Matplotlib заключается в том, что значения полярной оси должны быть представлены в радианах:

```
axis = [math.radians(facing) for facing in scan_data.  
keys()]
```

9. Теперь строим полярную диаграмму. Мы сообщаем Matplotlib ось, данные и то, что нам необходима зеленая линия с g-:

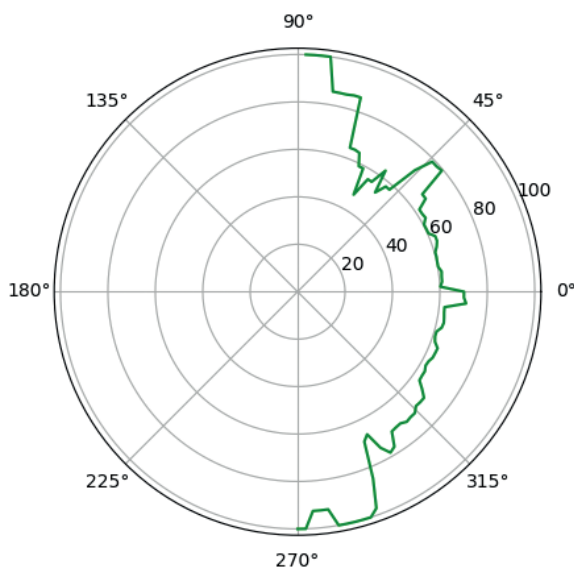
```
plt.polar(axis, list(scan_data.values()), 'g-')
```

10. Последняя строка записывает нашу диаграмму в файл изображения png, поэтому мы можем использовать scp, чтобы загрузить его с Raspberry Pi и просмотреть диаграмму:

```
plt.savefig("scan.png")
```

Загрузите код в Raspberry Pi. Поместите робота в пространство, где на расстоянии менее метра от него находится несколько препятствий. Введите `python3 sonar_scan.py` и запустите код. Вы увидите, как сервопривод перемещает датчик в соответствии с установленными границами.

После запуска кода изображение в файле `scan.png` должно выглядеть примерно так, как показано на рис. 10.23.



**Рис. 10.23.** Моя диаграмма результатов сканирования сонара

На рис. 10.23 показаны результаты сканирования сонара в виде полярной диаграммы. На ней показаны измерения в градусах. Зеленая линия показывает контуры предметов, обнаруженных датчиком. В моем примере нижняя граница составляет  $-90^\circ$ , верхняя  $90^\circ$ , а шаг  $-2^\circ$  (для более высокого разрешения).

Чем больше разрешение (например, если шаг меньше  $2^\circ$ ), тем медленнее будет происходить процесс сканирования. Вы можете отрегулировать длительность паузы, но низкие значения могут привести к тому, что сервопривод не успеет занять нужное положение, а датчик перестанет производить считывание данных.

### Устранение неполадок

Если у вас возникли проблемы, воспользуйтесь следующими советами:

- во-первых, убедитесь, что датчик расстояния работает нормально, для этого протестируйте его и выполните шаги по устранению неполадок (при необходимости), как в главе 8. Проверьте подключение проводов и запустите тестовый код;
- убедитесь, что сервоприводы работают так, как показано в разделе «Разработка кода для механизма поворота и наклона». При возникновении проблем воспользуйтесь предложенными там советами по устранению неполадок;
- в случае возникновения ошибок при запуске кода проверьте, установили ли вы необходимые библиотеки;
- проверьте код на наличие опечаток;
- помните, что выходной файл диаграммы будет сохранен на Raspberry Pi, и для просмотра его нужно скопировать на ваш компьютер;
- выведите значения на печать перед тем, как они попадут в метод `plt.polar`. Для этого добавьте следующие строки:

```
print(axis)
print(scan_data.values())
```

Теперь у вас должна быть возможность выполнить сканирование и построить диаграмму, подобную представленной выше. Я предлагаю поэкспериментировать с настраиваемыми значениями, чтобы посмотреть, как выглядит диаграмма в разном разрешении. Также вы можете попробовать разместить перед датчиком разные комбинации объектов.

Подведем итог тому, что мы узнали в этой главе.

## Выводы

В этой главе вы узнали о сервоприводах, о принципе их работы и о том, как управлять ими через контроллер двигателя. На их основе вы создали механизм поворота и наклона и добавили код для работы с ним в объект `Robot`. Затем мы реализовали сценарий движения всех частей механизма по окружности.

Созданный код вы можете использовать для управления другими системами на основе сервоприводов, например манипуляторами роботов. Техники анимации позволят вам реализовать плавные и круговые движения. Я использовал похожую систему в `SpiderBot`, ногами которого управляют 18 сервоприводов.

Вы увидели, что сочетание датчика и сервопривода позволяет создавать подобие карты объектов, которые находятся перед роботом. Также вы узнали, что подобные системы реализуются и в более крупных и дорогих роботах, но в них используются лидары.

В следующей главе мы рассмотрим получение роботом данных об окружающей среде (и их отображение) с помощью энкодера. Такие датчики измеряют угол поворота колес робота, на основе чего определяют, как он движется.

## Задание

1. Подумайте, какие еще системы для нашего робота можно создать на основе сервоприводов. Для реализации манипулятора потребуется четыре сервопривода, но для простого механизма захвата хватит двух (для подключения которых в нашем роботе как раз осталось два канала).
2. Посмотрите, какие наборы для сборки механизма захвата существуют. Некоторые наборы предполагают наличие клещей или системы управления подъемом/опусканием
3. Какой код необходим для управления механизмом захвата? Что бы вы добавили в объект Robot?
4. Какой демонстрационный поведенческий сценарий вы бы создали для этого механизма?
5. Механизм захвата может двигаться очень резко, если просто задавать ему определенные положения. Каким образом можно реализовать более медленное и плавное движение?

## Дополнительные материалы

- Основой платы управления сервоприводом является устройство PCA9685. Больше информации о работе с этой микросхемой вы можете найти в технической спецификации PCA9685 (<https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf>). Я настоятельно рекомендую вам ознакомиться с этим материалом.
- Также я рекомендую прочесть техническую спецификацию сервопривода SG90 ([http://www.ee.ic.ac.uk/pcheung/teaching/DE1\\_EE/stores/sg90\\_datasheet.pdf](http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf)), где представлена информация о принципах его работы.
- В руководстве к механизму поворота и наклона от AdaFruit (<https://learn.adafruit.com/mini-pan-tilt-kit-assembly>) вы можете найти инструкции по сборке. Порядок шагов отличается от представленного в этой книге, однако это может помочь взглянуть на сборку с другой стороны в случае, если возникнут какие-либо трудности.

# Глава 11

## Код на Python для работы с энкодерами

Во многих робототехнических проектах необходима точная информация о вращении валов двигателей и колес. Мы научили нашего робота двигаться по заданной траектории еще в главе 7, но маловероятно, что сейчас он может точно следовать по курсу. При разработке поведенческих сценариев важно иметь возможность задавать роботу определенную дистанцию движения и отслеживать пройденное расстояние. В этой главе мы узнаем, как работают датчики, предназначенные для таких задач, а также разработаем программу, которая заставит робота двигаться по прямой на определенное расстояние. Затем рассмотрим, как реализовать точный поворот. Обратите внимание, что в этой главе мы используем математику, но волноваться не стоит, поскольку она довольно проста.

В этой главе мы рассмотрим следующие темы:

- измерение пройденного расстояния с помощью энкодеров;
- установку энкодеров на робота;
- измерение пройденного расстояния с помощью кода на Python;
- движение по прямой;
- перемещение на определенное расстояние;
- выполнение точного поворота.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам потребуется следующее:

- робот с Raspberry Pi и код, созданный в предыдущей главе (<https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter10>);
- два энкодера с щелевыми оптопарами. Вам подойдет датчик скорости от Arduino, датчик скорости LM393 или модуль фотопрерывателя. Обратите внимание, что датчик должен быть рассчитан на напряжение питания 3,3 В. Посмотреть, как выглядят подходящие датчики, можно в разделе «Энкодеры, которые мы будем использовать»;
- длинные соединительные провода с разъемами «штекер–гнездо»;
- линейка для измерения диаметра колес робота (по возможности используйте штангенциркуль).

Найти код на из этой главы на GitHub вы можете по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter11>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/2XDFae0>.

## ИЗМЕРЕНИЕ ПРОЙДЕННОГО РАССТОЯНИЯ С ПОМОЩЬЮ ЭНКОДЕРОВ

**Энкодеры** – это датчики, выходные значения которых зависят от перемещения компонентов робота. Они могут определять угловое положение вала или частоту вращения колесной оси. Такие датчики могут регистрировать вращение или прямолинейное движение.

Оценка пройденного расстояния с течением времени называется **одометрией**, а датчики, предназначенные для этой цели, принято называть **тахометрами**. Обратите внимание, что датчики, упомянутые в разделе «*Технические требования*», в магазинах могут называться **тахометрами для Arduino**.

### Где применяются энкодеры

При создании роботов, подобных нашему, часто используются электронные датчики. В легковых и грузовых транспортных средствах в качестве спидометров и тахометров могут применяться как электронные, так и механические датчики.

В принтерах и сканерах энкодеры используются вместе с ДПТ в качестве альтернативы шаговым двигателям. Определение угла поворота по дуге является критически важным фактором для сервоприводов, которые мы рассматривали в главе 10. В высокотехнологичных аудио- или электрических системах диагностики/измерений энкодеры используются в регуляторах. Они представляют собой автономные модули, похожие на ручки регулировки громкости, которые пользователь может прокручивать без ограничений.

Получив общее представление об энкодерах, мы можем перейти к рассмотрению их типов.

### Типы энкодеров

На рис. 11.1 показаны кодовые датчики разных типов. Устройства 1–3 используют разные механизмы для измерения параметров движения. Под цифрой 4 показан диск энкодера и энкодерная лента.

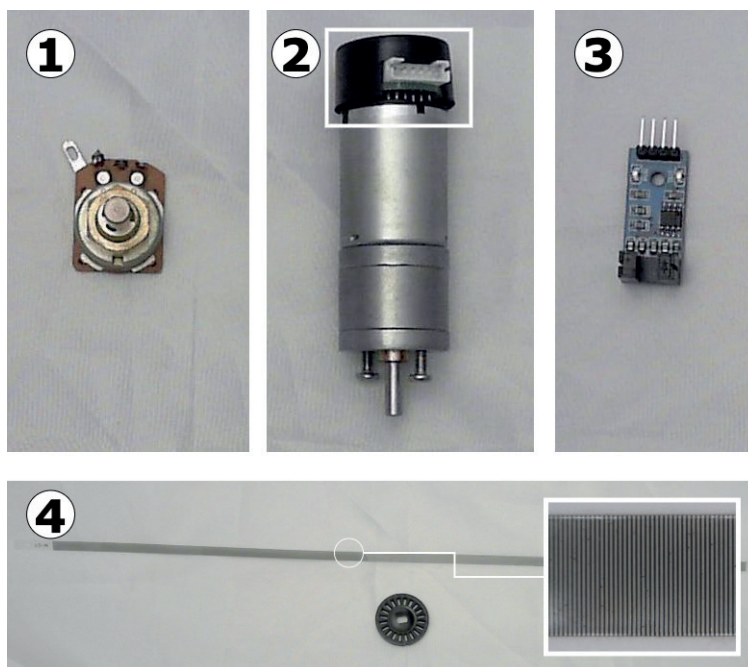


Рис. 11.1. Кодовые датчики

Такие датчики делятся на несколько категорий.

1. На этой части рисунка показан **переменный резистор**. Это аналоговое устройство, которое измеряет угол поворота. При этом оно не рассчитано на неограниченное вращение. В переменном резисторе есть контакты, подключенные к подложке с металлическим или резистивным слоем, который со временем изнашивается. Переменный резистор – это не совсем энкодер, но все же он является крайне полезным устройством. Для его подключения к Raspberry Pi потребуется аналогово-цифровой преобразователь, поэтому он не подходит для нашего проекта. Переменные резисторы также применяются в сервоприводах.
2. На этой части рисунка показан двигатель с магнитными энкодерами (выделены белым прямоугольником), которые также называют **датчиками Холла**. Они работают по следующему принципу: магниты, расположенные на диске или ленте, перемещаются рядом с датчиком, что повышает или понижает напряжение на его выходе.
3. Здесь показан стандартный оптический датчик. Принцип его работы состоит в следующем: датчик фиксирует прерывание инфракрасных лучей, которые проходят через пазы на диске энкодера. Такие устройства используются в трекболах для компьютеров, принтеров, а также в робототехнике. Оптические датчики производят последовательность импульсов. Они реагируют на прерывание инфракрасных лучей, поэтому их называют **фотопрерывателями**, **оптическими датчиками** или **оптическими прерывателями**. Мы будем использовать именно этот тип датчиков.



4. Здесь показан диск энкодера и энкодерная лента с полосками, которые используются в оптических датчиках. Лента подходит для измерения параметров прямолинейного движения, а диск – вращательного. Как на диске, так и на ленте есть прозрачные и непрозрачные сегменты. Реже встречаются варианты с датчиками света и чередующимися светлыми и темными штрихами вместо пазов.

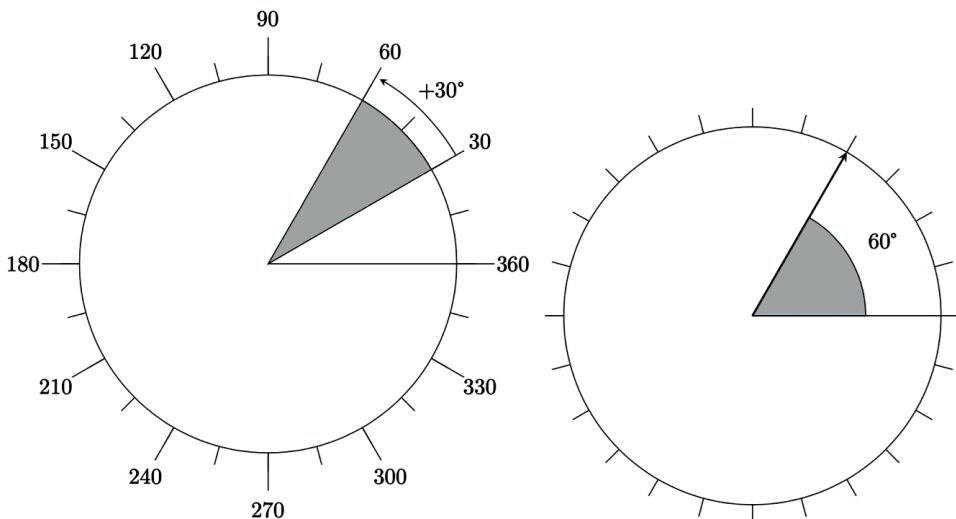
Рассмотрев некоторые типы энкодеров, я предлагаю более подробно изучить кодирование этими устройствами скорости и положения.

## Кодирование абсолютного или относительного положения

**Инкрементальные энкодеры** (относительные энкодеры) кодируют относительное изменение положения – например, если мы сделали определенное количество шагов по часовой или против часовой стрелки или вперед/назад по оси координат. Такие энкодеры позволяют измерять только положение относительно последнего измерения, подсчитывая количество пройденных пазов. Инкрементальные энкодеры недороги и просты в использовании, но у них есть и свои недостатки. Чтобы перейти на следующую позицию, датчики запоминают предыдущую, ввиду чего могут накапливаться ошибки.

Еще один тип – это **абсолютные энкодеры**. Они могут кодировать точное положение относительно оси. Абсолютным энкодерам не требуется информация о предыдущем положении, но требуется калибровка, которая позволит определить, соответствуют ли данные энкодера реальному положению.

На рис. 11.2 показано сравнение измерений абсолютного и инкрементального энкодеров.



**Рис. 11.2.** Сравнение измерений абсолютного и инкрементального энкодеров

Здесь показана разница между измерениями энкодеров двух упомянутых типов. На окружности слева мы видим, что перемещение (на 30°) происходит от предыдущего зафиксированного положения 30°. Это работает при ус-

ловии, что исходное зафиксированное положение является точным. Каждый раз, когда меняется исходное положение, меняется и значение перемещения. На окружности справа мы видим перемещение от 0 до 60°. Если датчик сообщает о конкретном положении объекта, это называется абсолютным положением. Если датчик сообщает о том, на сколько переместился объект, это называется относительным положением.

Грубо говоря, переменный резистор тоже можно рассматривать как абсолютный энкодер, когда его применяют для определения положения сервоприводов. Кодирование абсолютного положения выполняется с помощью оптических или магнитных маркеров на диске или ленте энкодера, что обеспечивает высокую точность определения абсолютного положения. Такие датчики могут быть громоздкими и дорогими, а для их подключения требуется несколько выходов. Обратите внимание, что диск или ленту абсолютного энкодера иногда могут называть шкалой.

## Кодирование направления и частоты вращения

Как правило, измерение относительного положения происходит посредством подсчета пазов на диске, которые прошли через датчик. Так мы узнаем частоту вращения и пройденное расстояние. Если поместить на диск два датчика на небольшом расстоянии друг от друга, мы сможем кодировать направление вращения. На рис. 11.3 показано, как это работает.

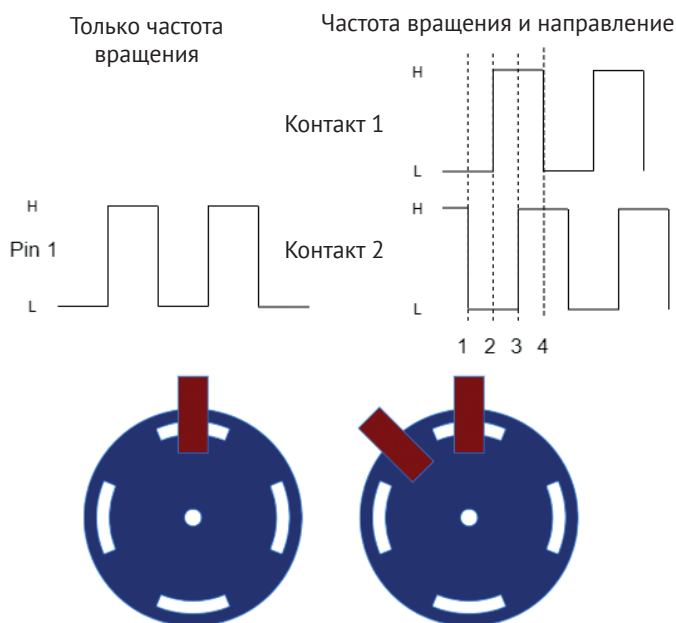


Рис. 11.3. Кодирование направления и частоты вращения с помощью нескольких датчиков

Система, показанная на левой части рисунка, кодирует частоту вращения. Когда пазы проходят рядом с датчиком, они генерируют электронные импульсы. Каждый импульс имеет **нарастающий фронт** (когда линия на

диаграмме идет вверх) и **спадающий** (линия идет вниз). Путем подсчета количества импульсов мы можем определить количество фронтов. Если двигатель вращается в одном направлении, то для него можно использовать систему с одним датчиком, подобную этой. Именно такой вариант мы реализуем в нашем роботе.

В системе, показанной на рисунке справа, появляется второй датчик. На диаграмме показано, что на фронтах импульсов значение датчика меняется в разных точках цикла с последовательностью, которую мы обозначили 1, 2, 3, 4. Направление последовательности указывает направление и частоту вращения диска. Здесь мы видим четыре фазы, поэтому можем назвать это квадратурным кодированием.

В промышленных роботах используются интерфейсы **записи и воспроизведения**. Пользователь нажимает кнопку записи и приводит робота (например, манипулятор) в нужное положение путем выполнения определенных движений, а затем нажимает кнопку остановки. Робот записывает эти движения и может воспроизводить их по запросу пользователя.

Роботу с системой записи и воспроизведения (или с системой управления мышью/трекболом) требуется точная информация о направлении вращения, поэтому возникает дополнительная сложность с определением этого параметра.

Мы реализуем недорогой и более простой вариант – система с одним датчиком, который будет измерять относительную частоту вращения. При разработке кода мы будем полагать, что датчик скорости каждого колеса регистрирует вращение в соответствии с направлением движения нашего робота.

## Энкодеры, которые мы будем использовать

Мы будем использовать щелевые оптические энкодеры, поместив их на диски, установленные на двигателях в главе 6. Такие энкодеры имеют цифровые выходы, и в коде на Python мы сможем подсчитывать импульсы, поступающие с них, что позволит определять, на какой угол повернулось колесо. На рис.11.4 показаны две модели датчиков, которые я рекомендую использовать.

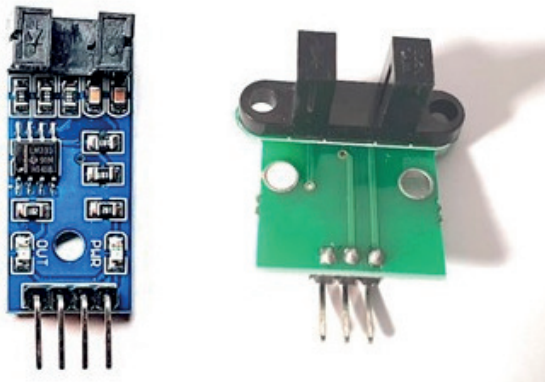


Рис. 11.4. Рекомендуемые модели энкодеров

Слева на рисунке показан модуль фотопрерывателя FC-03, а справа – модуль фотопрерывателя Waveshare. Нам подходят модули, рассчитанные на напряжение 3,3 В. Для модулей с рабочим напряжением 5 В потребуется преобразователь логического уровня и внесение изменений в схему подключения, как упоминалось в главе 8.

В нашем роботе диски энкодеров размещены на валах редукторов. Они должны вращаться синхронно с колесами, в противном случае нам придется учитывать разницу частоты вращения диска и колеса. Однако существуют такие факторы, которые сложно учесть, например проскальзывание колес, а также разный размер колес и шин. Если разместить по одному энкодеру на каждое направляющее колесо, мы получим более точные данные, однако использовать их в управлении роботом и сохранить неизменный контакт колес с полом довольно сложно.

В этом разделе мы узнали, что исходя из угла поворота колеса можно определить, на какое расстояние переместился робот. Также познакомились с различными типами энкодеров и выяснили, какой из них подойдет для нашего проекта (с рабочим напряжением 3,3 В). Помимо этого, мы кратко обсудили тему подсчета импульсов энкодерами. В следующем разделе установим энкодеры на нашего робота.

## УСТАНОВКА ЭНКОДЕРОВ НА РОБОТА

На данном этапе к нашему роботу подключено довольно много устройств. Raspberry Pi расположен прямо над прорезями на основании, которые предназначены для энкодеров, поэтому для их установки нужно будет снять Pi. После того как мы установим его обратно, подключим энкодеры к его порту GPIO, а также к источнику питания и «земле».

На рис. 11.5 показана блок-схема робота с энкодерами.

На этой блок-схеме мы видим левый и правый энкодеры, от которых к Raspberry Pi отходят стрелки, обозначающие поток данных. Новые компоненты выделены на схеме красным.

Прежде чем установить новые компоненты, нам необходимо узнать сколько пазов есть на диске энкодера. На рис. 11.6 показан мой диск, на котором 20 пазов. Сосчитайте, сколько их на вашем диске.

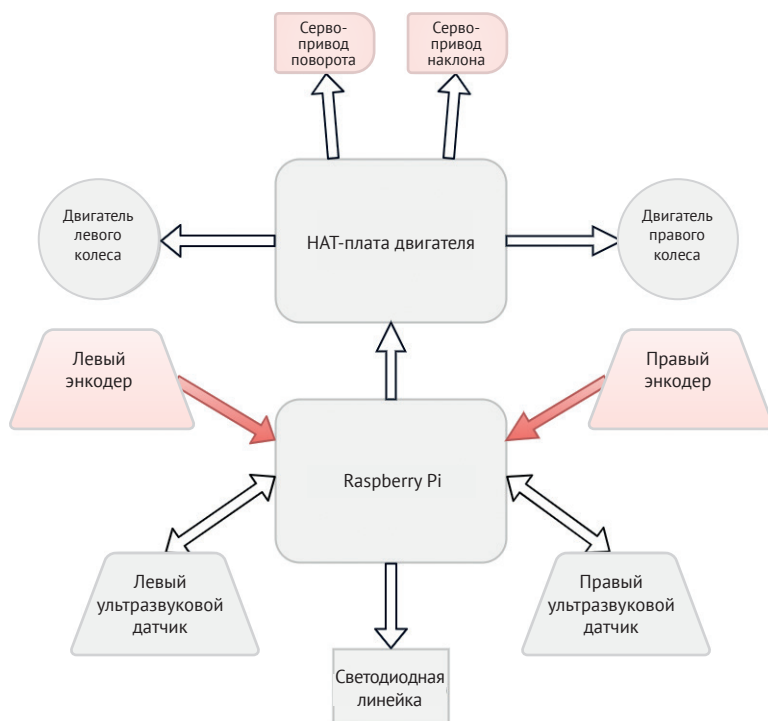


Рис. 11.5. Блок-схема робота с энкодерами



Рис. 11.6. Диск энкодера

## Подготовка энкодеров

Прежде чем начать пользоваться энкодерами, необходимо подготовить их и установить на нашего робота. Поскольку энкодеры находятся под Raspberry Pi, к ним нужно подключить соединительные провода типа «штекер–гнездо». Также я рекомендую закрыть все электрические контакты, торчащие из-под Raspberry Pi, изоляционной лентой.

На рис. 11.7 показано, что провода подключаются к контактам **GND** («земля»), **3V**, **Vin** или **VCC** (напряжение) и **D0/OUT** (цифровой выход). Контакт **A0** (аналоговый выход) остается свободным (при его наличии). По возможности для

провода, который будет подключаться к GND, выберите провода самого темного цвета, а для провода, который будет подключаться к контакту питающего напряжения, – самого светлого. Для того чтобы не запутаться, я предлагаю обернуть конец сигнального провода небольшим кусочком изоляционной ленты.



Рис. 11.7. Контакты для подключения проводов на датчике

### Важное примечание

Расположение контактов на датчиках может отличаться, поэтому, прежде чем устанавливать Raspberry Pi обратно, найдите в интернете фотографии, на которых показано расположение контактов датчиков.

## Снятие Raspberry Pi

Энкодеры должны находиться под Raspberry Pi, поэтому нам необходимо осторожно снять микрокомпьютер (не нарушив при этом подключение соединительных проводов). На рис. 11.8 показано, как это сделать:

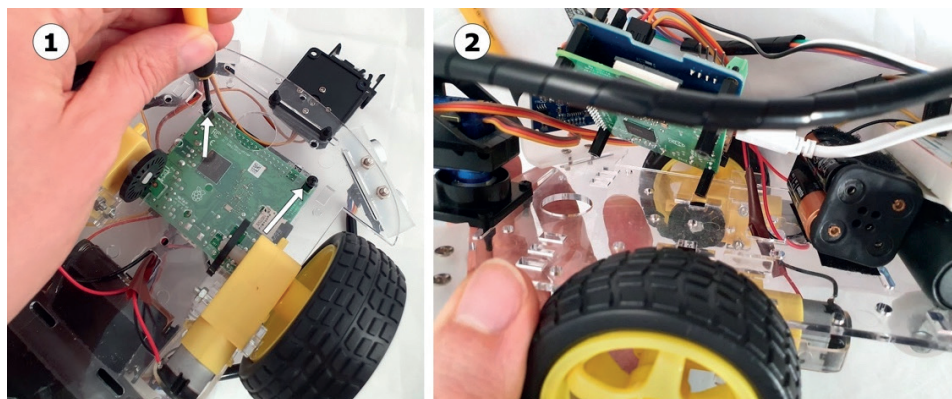


Рис. 11.8. Откручиваем винты и снимаем Raspberry Pi

Руководствуясь рис 11.8, выполните следующие действия.

1. Аккуратно открутите винты, с помощью которых Raspberry Pi крепится на шасси. Отложите их в сторону.



2. Осторожно снимите Raspberry Pi с робота, не нарушая подключение проводов. На этой части рисунка показано, как должен выглядеть робот на данном этапе.

Отлично! Вы сняли Raspberry Pi и освободили место для установки энкодеров.

## Установка энкодеров на шасси

Теперь мы установим энкодеры на шасси робота. На рис. 11.9 показана установка энкодеров на пустое шасси. Подобным образом устанавливаются все типы энкодеров.

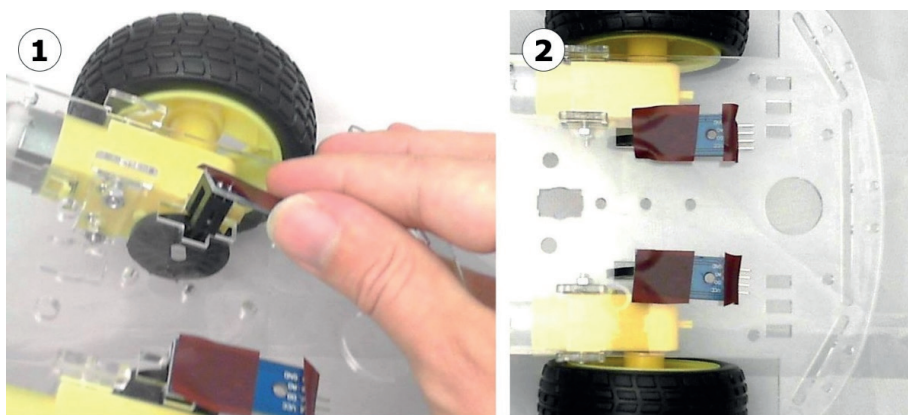


Рис. 11.9. Установка энкодеров на шасси

Руководствуясь рис. 11.9, выполните следующие шаги.

1. Аккуратно вставьте энкодеры в пазы рядом с диском энкодера.
2. Датчики должны входить с небольшим сопротивлением и фиксироваться в пазах.

Как только вы вставите энкодеры в пазы, Raspberry Pi можно установить обратно и закрепить винтами. Для этого вам могут понадобиться стойки разного размера.

### Важное примечание

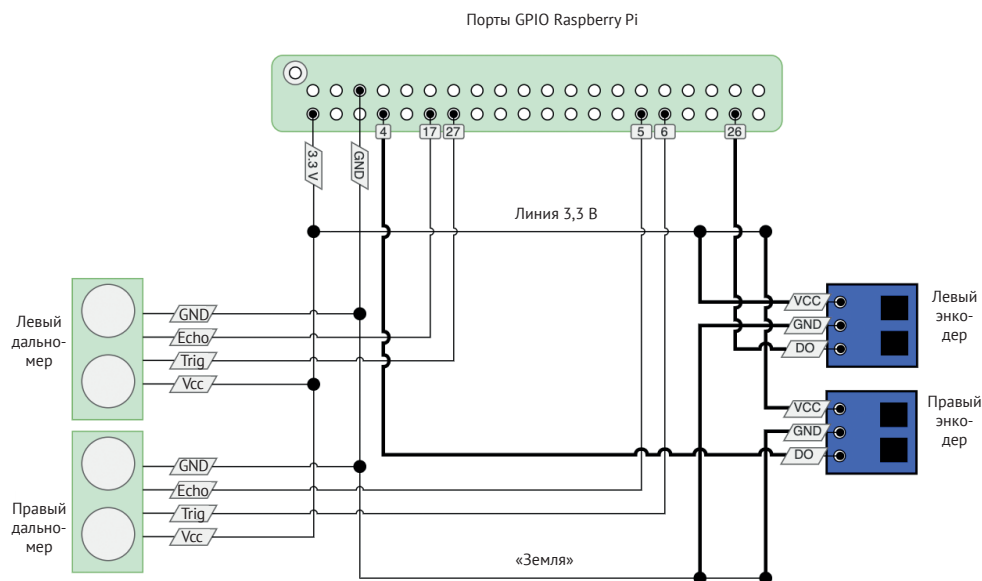
На этом этапе проверьте все подключения. Провода двигателей могли отсоединиться.

Когда вы установили энкодеры и вернули Raspberry Pi на место, можно перейти к подключению энкодеров к Pi.

## Подключение энкодеров к Raspberry Pi

Теперь мы можем приступить к подключению энкодеров к Pi с помощью макетной платы. Инструкция к подключению показана на рис. 11.10 в виде принципиальной электрической схемы.





**Рис. 11.10.** Принципиальная схема подключения энкодеров к Raspberry Pi

Данная схема представляет собой дополненную схему из главы 8. Новые подключения показаны толстыми линиями.

На рис. 11.10 справа находятся модули энкодеров (правый и левый). На датчике есть контакт с подписью **VCC**, **VIN**, **3V** или просто **V**. Его нужно подключить к линии 3,3 В. Контакт с подписью **GND**, **0V** или **G** подключается к черной/синей линии на макетной плате. Также на энкодерах есть контакт с подписью **DO** (Digital out – цифровой выход), **SIG** или **S** (Signal – сигнальный), подключенный к разъему GPIO для цифрового сигнала. Мы подключаем цифровой выход левого энкодера к контакту 4 GPIO, а правого – к контакту 26.

У некоторых энкодеров есть дополнительные контакты, например **AO** (**Analog Out – Аналоговый выход**). Этот контакт нам не понадобится, поэтому оставим его свободным.

Выполните следующие шаги.

1. Подключите соединительные провода от контактов датчика к макетной плате. При этом обращайте внимание на подписи контактов, поскольку их расположение может отличаться на разных датчиках.
2. Подключите контакты разъема GPIO Raspberry Pi (4 и 26) к макетной плате.
3. Подключите питание через макетную плату, используя провода из комплекта.

**Важное примечание**

Чем больше у нашего робота межмодульных соединений, тем сложнее реализовывать дополнительные подключения или производить ремонт. Создание **печатных плат (PCB – Printed Circuit Boards)** своими руками выходит за рамки этой книги, однако такой подход позволяет упорядочить все провода. Эти платы более прочные и занимают меньше места. Однако замена платы – процесс достаточно сложный.

Можно обойтись без макетной платы и подключить датчики к Raspberry Pi напрямую, но преимущество использования платы заключается в том, что она позволяет распределять подключения питания и группировать подключения датчиков.

Мы реализуем эту схему в работе, в котором есть множество других подключений. Если длины проводов для подключения датчиков к роботу недостаточно, удлините провода по тому же принципу, который описан в главе 10, с помощью еще одного набора соединительных проводов типа «штекер–гнездо».

В этом разделе мы разобрались, как подключить датчики, и теперь можем приступить к созданию кода для работы с ними. В следующем разделе разработаем тестовый код, а затем рабочий код для измерения пройденного расстояния.

## ИЗМЕРЕНИЕ ПРОЙДЕННОГО РАССТОЯНИЯ С ПОМОЩЬЮ КОДА НА PYTHON

Для использования кодовых датчиков нам необходимо подсчитывать количество импульсов. В этом разделе мы создадим код, который запустит двигатели и подсчитает количество импульсов. С его помощью мы убедимся, что энкодеры работают правильно. Затем на основе этого кода мы разработаем поведенческий скрипт и сделаем его частью класса `Robot`.

### Ведение журнала

Раньше для вывода информации мы использовали функцию `print`. Данный способ достаточно эффективный, но, если мы будем использовать данную функцию для отображения всей интересующей нас информации, ее вывод будет слишком объемным. Ведение журнала (журналирование) тоже позволяет отображать информацию, и при этом мы можем контролировать ее количество. Для реализации журналирования необходимо импортировать модуль `logging`.

В журнале есть уровни `debug`, `info`, `warning` и `error`. Уровень `debug` полезен на начальных этапах разработки и при устранении неполадок, здесь мы можем увидеть всю информацию. Уровень `info` отображает меньше данных. Уровни `warning` и `error` предназначены только для отображения соответствующих ошибок, поэтому, если вы достаточно уверены в других фрагментах кода, можете просмотреть только эти уровни. Функция `logging.basicConfig` позволяет нам настраивать уровни журнала и другие параметры. На этом этапе необходимо построить журнал и включить уровни `info` и `debug`.

Для нашего журнала необходимы регистраторы с разными именами, которые мы можем создать посредством функции `logging.getLogger`. С помощью регистраторов можно устанавливать разные уровни для разных модулей. Использование именованных регистраторов помогает включить уровень `debug` из используемого библиотечного модуля и пользоваться при этом уровнем `info` основного модуля.

В этой главе мы создадим код для ведения журнала и управления его уровнями, что позволит нам в дальнейшем получать более подробную информацию о том, что делает робот, включая и выключая уровни по желанию. Это пригодится нам тогда, когда мы введем ШИМ-контроллер. В следующем разделе с помощью ведения журнала мы сможем посчитать число импульсов датчика.

## Простой подсчет

Этот тестовый код подсчитывает количество нарастающих и спадающих фронтов на каждом сигнальном контакте, а затем выводит его на печать. Код покажет, правильно ли мы подключили датчики, и в случае возникновения ошибок мы исправим их на этом этапе. В дальнейшем, взяв его за основу, мы разработаем другой код, с помощью которого будем отслеживать параметры вращения колес. Здесь же мы запустим двигатели примерно на 1 с. Обратите внимание, что этот код является продолжением кода из главы 10.

### Важное примечание

Код вы можете найти по адресу [https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/test\\_encoders.py](https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/test_encoders.py).

1. Создайте файл с именем `test.encoders.py`. Начнем с обычных классов `import` и `time`, а также добавим журналирование:

```
from robot import Robot
import time
import logging
...
```

2. Далее добавим импорт библиотеки GPIO Zero для устройства ввода. Мы можем использовать заданный контакт для подсчета импульсов:

```
...
from gpiozero import DigitalInputDevice
logger = logging.getLogger("test_encoders")
...
```

Также для нашей системы мы настроили регистратор с именем, соответствующим имени файла.

3. Энкодеры генерируют импульсы, которые мы будем подсчитывать и отслеживать их состояние. Мы используем более чем один из них. Необходимо создать класс, из которого мы сможем передавать номера контактов ввода/вывода в конструктор. Сначала настроим счетчик импульсов:

```
...
class EncoderCounter(object):
    def __init__(self, pin_number):
        self.pulse_count = 0
    ...
```

4. В конструкторе нужно настроить устройство и реализовать подсчет импульсов с его помощью. У устройства есть объект `.pin`, который мы настраиваем, используя номера контакта. Объект `.pin` имеет событие `when_changed`, куда мы можем поместить обработчик, который будет вызываться каждый раз при изменении логического уровня на контакте. При прохождении каждой прорези энкодера через датчик напряжение на его выходе резко возрастает, а затем падает:

```
...
self.device = DigitalInputDevice(pin=pin_number)
self.device.pin.when_changed = self.when_changed
...
```

5. Необходимо определить метод `when_changed` для нашего класса, чтобы добавить к `pulse_count` единицу. Этот метод должен быть небольшим или выполняться быстро, поскольку GPIO Zero вызывает его в фоновом режиме при каждом изменении импульса. Метод принимает параметры `time_ticks` и `state`. Мы не будем использовать `time_ticks`, поэтому в коде вместо него поставим нижнее подчеркивание:

```
...
def when_changed(self, _, state):
    self.pulse_count += 1
...
```

6. Далее настроим объект `robot` и создадим `EncoderCounter` для каждого датчика. Устройства мы подключаем к контактам 4 и 26:

```
...
bot = Robot()
left_encoder = EncoderCounter(4)
right_encoder = EncoderCounter(26)
...
```

7. Чтобы отобразить значения, вместо функции `sleep` мы выполняем цикл и проверяем время его окончания. Перед записью данных устанавливаем параметры ведения журнала с помощью метода `logging.basicConfig`. Затем мы запускаем двигатели и переходим в основной цикл:

```
...
stop_at_time = time.time() + 1

logging.basicConfig(level=logging.INFO)
bot.set_left(90)
bot.set_right(90)

while time.time() < stop_at_time:
    ...
```

В этом цикле показания датчиков записываются в журнал.

8. В сплошном цикле могут возникать критические ошибки (например, GPIO Zero вызовет наш код из потока исполнения датчика), поэтому все же нужно задать небольшую паузу:

```
...
logger.info(f"Left: {left_encoder.pulse_count} Right:
{right_encoder.pulse_count}")
time.sleep(0.05)
```

По завершению цикла программа также завершится, а робот автоматически остановится. Обратите внимание, что мы использовали форматизирующую строку, о которой упоминалось в главе 8. Префикс `f` позволяет нам форматировать переменные в строку.

Далее нужно загрузить этот код в робота и запустить его. Теперь по значениям энкодеров вы будете видеть, как меняются параметры движения робота. Вывод должен выглядеть примерно так:

```
pi@myrobot:~ $ python3 test_encoders.py
INFO:test_encoders:Left: 0 Right: 0
INFO:test_encoders:Left: 0 Right: 1
INFO:test_encoders:Left: 2 Right: 2
INFO:test_encoders:Left: 3 Right: 4
INFO:test_encoders:Left: 5 Right: 7
INFO:test_encoders:Left: 8 Right: 10
INFO:test_encoders:Left: 10 Right: 14
...
INFO:test_encoders:Left: 56 Right: 74
```

По данным энкодеров мы видим, что левое колесо совершило меньше оборотов, чем правое, и робот повернул влево. Часть `INFO:test_encoders:` вводится в журнал, показывая уровень журнала и имя регистратора. Расстояние указано в *takтах* энкодера, где такт – это каждое подсчитываемое событие.

Если во время исполнения этого кода возникли ошибки, обратитесь к следующему разделу, где представлены советы по их решению.

## Устранение неполадок

Если на этом этапе вы обнаружили какие-либо ошибки, воспользуйтесь следующими советами:

- код из этой главы является продолжением кода из главы 10. Если загрузить код только из этой главы, вероятнее всего, произойдет конфликт кода, который уже настроен для энкодеров, и GPIO;
- если значения в коде не меняются (остаются со значением 0), выключите Raspberry Pi и еще раз проверьте подключения и номера используемых контактов;
- проверьте состояние проводов – если они нагреваются, немедленно отключите питание и проверьте правильность подключений.

Итак, вы протестировали работу энкодеров в системе робота, а также увидели вывод их показаний. Теперь можно перейти к разработке более сложных поведенческих скриптов, но для начала нужно добавить энкодеры в объект `Robot`.

## Добавляем энкодеры в объект Robot

Прежде чем использовать энкодеры в других поведенческих сценариях, необходимо добавить их в объект `Robot`. Мы можем импортировать код в объект и установить для левого и правого датчиков соответствующие номера контактов. Для обработчиков может потребоваться очистка цикла.

### Извлечение класса

Мы уже создали класс `EncoderCounter`, который теперь вы можете скопировать из файла `test_encoders.py` в `encoder_counter.py` ([https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/encoder\\_counter.py](https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/encoder_counter.py)). Этому коду требуется импортировать `DigitalInputDevice`, конструктор и обработчик `when_changed`.

1. Сначала добавим импорты и описание класса. Класс `EncoderCounter` начинается так:

```
from gpiozero import DigitalInputDevice
```

```
class EncoderCounter:
    def __init__(self, pin_number):
        self.pulse_count = 0
```

2. Для учета изменения направления движения на обратное добавим элемент `direction`:

```
self.direction = 1
```

3. Конструктор (`__init__`) завершается настройкой устройства и назначением обработчика `when_changed`:

```
self.device = DigitalInputDevice(pin=pin_number)
self.device.pin.when_changed = self.when_changed
```

4. Обработчик `when_changed` должен прибавить к счетчику не просто 1, а указатель направления, чтобы затем посчитать количество нарастающих и спадающих фронтов импульсов:

```
def when_changed(self, time_ticks, state):
    self.pulse_count += self.direction
```

5. Также нам необходим метод, задающий значение указателя направления. Для проверки мы вызываем исключение и смотрим, соответствует ли значение условиям, заданным в коде. Такой подход – это простой, но грубый способ проверить, имеют ли смысл наши значения:

```
def set_direction(self, direction):
    """Значение должно быть -1 или 1."""
    assert abs(direction)==1, "Direction %s should be
1 or -1" % direction
    self.direction = direction
```

6. Затем вводим метод `reset`, который обрабатывает перезапуск счетчиков между передвижениями:

```
def reset(self):
    self.pulse_count = 0
```

7. Для очистки данных нам нужен способ остановить счетчики, чтобы они не вызывали обработчик:

```
def stop(self):
    self.device.close()
```

Мы подготовили библиотеку энкодера и теперь можем использовать ее в коде. Благодаря библиотеке мы можем использовать счетчик энкодера в других местах, а также использовать его для других устройств со схожими параметрами. Для удобства использования счетчика в других поведенческих скриптах я рекомендую импортировать его в библиотеку робота.

### Добавляем устройство в объект Robot

Мы использовали объект Robot в качестве основного интерфейса между кодом, обрабатывающим данные, полученные от устройств, и поведенческим скриптом.

Теперь изменим код из файла robot.py, созданный в главе 10 (<https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter10/robot.py>) для работы с датчиками.

1. Начнем с импорта EncoderCounter:

```
...
import leds_led_shim
from servos import Servos
from encoder_counter import EncoderCounter
...
```

2. В методе конструктора \_\_init\_\_ нам необходимо настроить левый и правый энкодеры. Введем необходимый код сразу после кода для работы с датчиками расстояния:

```
...
# настройка датчиков расстояний
self.left_distance_sensor = DistanceSensor(echo=17,
trigger=27, queue_len=2)
self.right_distance_sensor =
DistanceSensor(echo=5, trigger=6, queue_len=2)

# настройка энкодеров
self.left_encoder = EncoderCounter(4)
self.right_encoder = EncoderCounter(26)
...
```

3. Чтобы удостовериться, что данные обработчиков энкодеров сбрасываются при завершении кода из объекта Robot, мы вызываем методы энкодеров stop в методе stop\_all:

```
...
# Сбрасываем значения светодиодов
self.leds.clear()
self.leds.show()

# Сбрасываем значения обработчиков датчиков
self.left_encoder.stop()
self.right_encoder.stop()
...
```



Полный код из файла `robot.py` с кодом для энкодеров можно найти на GitHub по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/robot.py>. Теперь на его основе мы можем разработать поведенческий скрипт для измерения пройденного расстояния в миллиметрах. Для этого нам необходимо разобраться в соотношении количества тактов энкодера и пройденного расстояния в миллиметрах.

## Преобразование тактов в миллиметры

Для расчета реального расстояния нужно измерить диаметр колес. Мы не можем учесть проскальзывание колеса, но можем узнать, на сколько градусов оно повернулось, – именно так работают энкодеры. Мы можем узнать, какое расстояние проехало колесо, зная его диаметр. Измерьте диаметр с помощью линейки или штангенциркуля, как показано на рис.11.11:

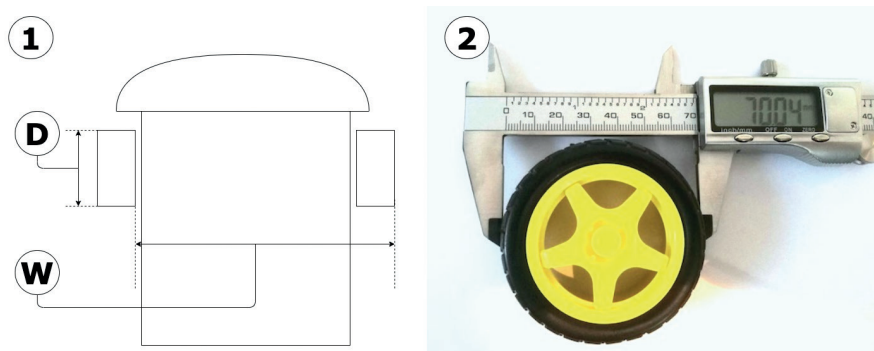


Рис. 11.11. Измерение колеса

На рисунке показаны все необходимые измерения.

1. Сначала необходимо измерить диаметр колеса (**D**) и расстояние между колесами (**W**). **W** – это расстояние от середины одного колеса, управляемого двигателем, до середины другого. Как показано на этой части рисунка, проще измерить расстояние от правой стороны одного колеса до правой стороны другого – результат получается такой же. У меня расстояние **W** составило примерно 130 мм.
2. Как показано на этой части рисунка, диаметр **D** можно измерить с помощью штангенциркуля по самой широкой части колеса.

Мы знаем, сколько прорезей есть у наших энкодеров, и предполагаем, что на каждую прорезь будет приходиться 2 фронта (нарастающий и спадающий)<sup>12</sup>. Чтобы узнать, какое количество тактов приходится на полный оборот колеса, необходимо умножить количество прорезей на 2. У меня получается 40.

Число пи ( $\pi$ ) представляет собой отношение диаметра колеса к длине окружности. Чтобы вычислить длину окружности, мы умножаем диаметр на  $\pi$ , полу-

<sup>12</sup> Мы учитываем оба фронта импульса – нарастание и спад, – потому что используем в управляющем коде метод `onChange`, который учитывает любые изменения логического уровня на входе GPIO. – Прим. ред.

чая  $\pi D$ , где  $D$  – диаметр. Далее делим  $\pi$  на общее количество тактов за оборот, а затем умножаем полученное число на количество подсчитанных тактов ( $T$ ) и диаметр ( $D$ ). Так мы получаем значение пройденного колесом расстояния ( $d$ ):

$$d = \frac{\pi}{40} \times D \times T.$$

Далее необходимо реализовать все вычисления в коде. Для этого выполните следующие шаги.

1. Создайте новый файл с именем `test_distance_travelled.py`. Сначала нужно импортировать библиотеку `math` для вычислений, объект `Robot` и библиотеку `time`:

```
from robot import Robot
import time
import math
import logging
logger = logging.getLogger("test_distance_travelled")
...
```

2. Затем определите константы – диаметр колес и количество тактов на оборот. Введите собственные значения, а не те, которые я представил в примере:

```
...
wheel_diameter_mm = 70.0
ticks_per_revolution = 40.0
...
```

3. Создайте функцию для преобразования подсчитанных тактов в расстояние. Такты должны преобразовываться в целые числа, поскольку это значение не может быть выражено в долях миллиметра. Преобразование не меняется, поэтому это значение мы также делаем константой:

```
...
ticks_to_mm_const = (math.pi / ticks_per_revolution) *
wheel_diameter_mm
```

```
def ticks_to_mm(ticks):
    return int(ticks_to_mm_const * ticks)
...
```

4. Теперь определите нашего робота, установите время паузы и запустите двигатели:

```
...
bot = Robot()
top_at_time = time.time() + 1

logging.basicConfig(level=logging.INFO)
bot.set_left(90)
bot.set_right(90)
...
```

5. В этом цикле расстояние отображается после вызова метода `ticks_to_mm` для счетчика импульсов:

```
...
while time.time() < stop_at_time:
    logger.info("Left: {} Right: {}".format(
        ticks_to_mm(bot.left_encoder.pulse_count),
        ticks_to_mm(bot.right_encoder.pulse_count)))
    time.sleep(0.05)
```

6. После загрузки и запуска кода вывод будет выглядеть следующим образом:

```
pi@myrobot:~ $ python3 test_distance_travelled.py
INFO:test_distance_travelled:Left: 0 Right: 0
INFO:test_distance_travelled:Left: 5 Right: 0
INFO:test_distance_travelled:Left: 16 Right: 10
INFO:test_distance_travelled:Left: 32 Right: 21
...
...
INFO:test_distance_travelled:Left: 368 Right: 384
INFO:test_distance_travelled:Left: 395 Right: 417
```

В этом выводе мы видим четкую разницу между ходом левого и правого двигателей. Правый движется немного быстрее левого. Со временем разница накапливается, и робот поворачивает. В следующем разделе мы воспользуемся полученной информацией и немного проясним ситуацию.

## ДВИЖЕНИЕ ПО ПРЯМОЙ

На данном этапе мы можете видеть различия в результатах, т. е. отклонение. Всего через 400 мм пройденного пути левое колесо начинает отставать от правого на 20 мм – ошибка накапливается. В зависимости от двигателей ваш робот также может немного отклоняться от направления. Редко бывает так, что роботы двигаются по идеальной прямой. Мы постараемся скорректировать движение с помощью датчиков.

### Совет

Лучше всего результат выполнения поведенческого сценария виден при движении робота по деревянному напольному покрытию или ДВП, а хуже всего – при движении по ковро.

Такая коррекция движения по-прежнему является точным расчетом траектории. Проскальзывание и неправильные измерения все еще могут сбить робота с курса. Итак, как с помощью двигателей и энкодеров можно скорректировать траекторию и заставить робота двигаться по прямой?

## Коррекция отклонения с помощью ПИД-регуляторов

Для автоматической коррекции отклонения и поддержания движения робота по прямой робот должен менять скорость двигателя до тех пор, пока колеса

не повернутся одинаково. Если колеса повернутся на одну и ту же величину достаточно быстро, то они будут учитывать значительные отклонения от курса.

Наш робот будет измерять поворот колес с помощью энкодеров. Затем мы можем определить разницу между ними и отрегулировать управление двигателями так, чтобы частота их вращения была одинаковой.

Нам необходимо выяснить, как разница в измерениях связана с регулировкой скорости двигателя. Далее мы рассмотрим, как использовать систему **пропорционально-интегрально-дифференцирующих регуляторов** (ПИД-регуляторов) для преобразования ошибки в значение регулировки и выходные значения.

Для движения по прямой необходимо реализовать замкнутую цепь **обратной связи**. На рис. 11.12 показано, как работает эта цепь:

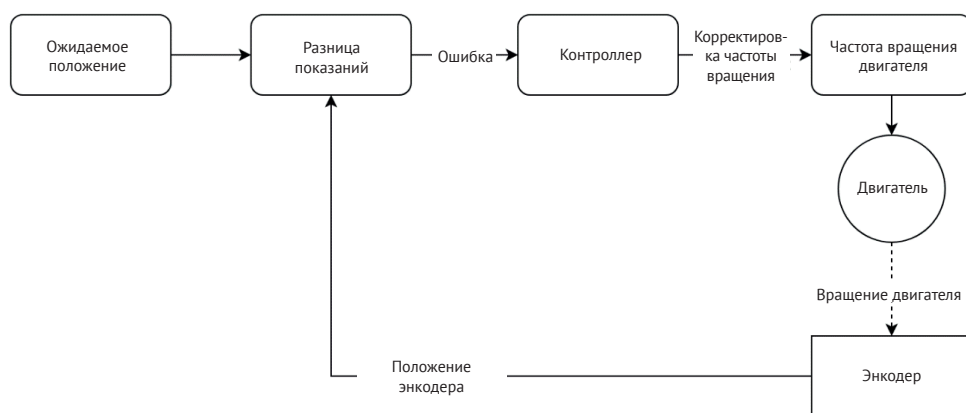


Рис. 11.12. Замкнутая цепь управления частотой вращения двигателей

Цепь начинается с *ожидаемого положения* или **уставки регулятора**. *Положение энкодера* передает данные обратной связи из реального мира. Мы получаем разницу между уставкой и реальным положением энкодера – *ошибку*. Посредством кода ошибка передается в *контроллер*, который производит *корректировку частоты вращения*. Затем система применяет эту корректировку к *вращению двигателя*, заставляя двигатель вращаться быстрее или медленнее, тем самым меняя обратную связь энкодера.

Для движения по прямой нам необходимо из значения частоты вращения левого двигателя вычесть значение частоты вращения правого двигателя и получить разницу показаний энкодеров. Наше *ожидаемое положение* – 0. Таким образом, *ошибка* равна разнице измерений энкодеров. На основе этого можно регулировать частоту вращения двигателей через контроллер.

Мы будем регулировать частоту вращения двигателей посредством ПИД-регуляторов, которые имеют три компонента:

- **пропорциональный (П)**: значение ошибки, умноженное на константу. Этот компонент реагирует на текущую ошибку;
- **интегральный (И)**: сумма значений текущих ошибок, умноженная на константу. Этот компонент имеет свойство накапливать ошибку;

- **дифференциальный (Д):** разница между последним зафиксированным значением ошибки и текущим значением ошибки умножается на константу. Этот компонент позволяет противодействовать внезапным отклонениям от траектории.

Путем изменения констант мы *осуществляем отстройку* регулятора – определяем влияние каждого фактора на результат работы контроллера. В нашем поведенческом скрипте мы не используем дифференцирующий компонент, поэтому его константа устанавливается равной 0.

Интегральный компонент позволяет реализовать автоматическую отстройку, но он должен иметь очень маленькую константу, так как высокие значения могут привести к неровному ходу робота. К значению частоты вращения одного двигателя мы прибавляем значение регулировки, а из значения частоты вращения второго двигателя вычитаем его.

Устанавливаем частоту вращения правого двигателя следующим образом:

```
...
integral_sum = integral_sum + error
right_motor_speed = speed + (error * proportional_constant) +
(integral_sum * integral_constant)
...
```

Для ускорения нам необходимо задействовать неиспользованную мощность двигателя. Если частота вращения близка к 100 %, мы получаем отсечение, которое в сценарии с задействованным интегральным компонентом может привести к тому, что робот будет вести себя странно. Именно поэтому крайне важно следить за отсечением на 100 %.

#### Совет

Оставьте запас для ПИД-регулятора – используйте не более 80 % мощности двигателя.

Когда мы разобрались в том, что представляет собой ПИД-регулятор, можно перейти к реализации в коде.

## Создание объекта ПИД-регулятора

Код для работы с ПИД-регулятором является важным строительным блоком для роботов, которые должны следовать по прямой. Мы будем использовать его в следующих главах, рассказывающих об управлении на основе данных камер. Базовые концепции из этого раздела вы сможете применить во многих роботизированных системах.

1. Нам необходимо создать упрощенный объект ПИД-регулятора, поскольку в дальнейшем мы будем использовать его в других частях кода. Создадим файл с именем `pid_controller.py`. Обратите внимание, сейчас мы создаем пропорционально-интегральный контроллер (ПИ). При необходимости позже можно добавить дифференцирующий компонент. Во фрагменте кода ниже показан класс и его конструктор:

```

import logging
logger = logging.getLogger("pid_controller")
class PIController:
    def __init__(self, proportional_constant=0, integral_
constant=0):
        self.proportional_constant = proportional_
constant
        self.integral_constant = integral_constant

    # Текущие суммы
    self.integral_sum = 0

...

```

Конструктор принимает константы. Я предварительно установил их значения равными нулю, поэтому вы можете изолировать компоненты. Значения хранятся в классе. Далее мы устанавливаем переменную для интегральной суммы, значение которой со временем будет увеличиваться. Обычно для удобства названия сокращают: `proportional_constant` – `pK`, а `integral_constant` – `iK`. Вы можете делать так, как удобно вам. В примерах я привожу полные названия, чтобы их было проще понять.

- Следующий фрагмент кода обрабатывает значения для двух компонентов. При обработке значений интегрального компонента значение интегральной суммы увеличивается:

```

...
def handle_proportional(self, error):
    return self.proportional_constant * error
def handle_integral(self, error):
    self.integral_sum += error
    return self.integral_constant * self.integral_sum
...

```

- Следующий фрагмент обрабатывает ошибку и рассчитывает значение регулировки:

```

...
def get_value(self, error):
    p = self.handle_proportional(error)
    i = self.handle_integral(error)
    logger.debug(f"P: {p}, I: {i:.2f}")
    return p + i

```

В этом фрагменте пропорциональная и интегральная части называются `p` и `i` соответственно. Данные значения регистрируются в журнале, поэтому мы можем настроить ведение журнала так, чтобы они отображались полностью при отладке и отстройке регулятора.

С помощью созданного в этом разделе кода с ПИ-регулятором наш робот сможет сопоставлять ошибки с предыдущими значениями и масштабировать их в соответствии с определенным движением. В следующем разделе мы будем использовать ПИД-регулятор для регулирования движения по прямой.

## Создание кода для движения по прямой

Создадим файл с именем `straight_line_drive.py`

([https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/straight\\_line\\_drive.py](https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/blob/master/chapter11/straight_line_drive.py)).

1. Сначала импортируем объект `Robot`, библиотеку `time` и наш ПИ-регулятор. Для получения отладочных данных ПИД-регулятора настроим ведение журнала соответствующим образом. Вы можете настроить его снова на `INFO` или убрать эту строку:

```
from robot import Robot
from pid_controller import PIDController
import time
import logging
logger = logging.getLogger("straight_line ")
logging.basicConfig(level=logging.INFO)
logging.getLogger("pid_controller").setLevel(logging.
DEBUG)
```

2. Далее настраиваем объект `Robot` и устанавливаем большее значение `stop_at_time`, чтобы наш робот проехал чуть дальше:

```
bot = Robot()
stop_at_time = time.time() + 15
...
```

3. Затем устанавливаем главное значение частоты вращения равным 80 и задаем его обоим двигателям:

```
...
speed = 80
bot.set_left(speed)
bot.set_right(speed)
...
```

4. Прежде чем перейти к главному циклу, настраиваем регулятор. Возможно, для него потребуется отстроить константы. Обратите внимание, насколько мала интегральная константа:

```
...
pid = PIDController(proportional_constant=5, integral_
constant=0.3)
...
```

5. С помощью функции `sleep` мы задаем небольшую задержку между проходами циклов, чтобы нашим энкодерам было что измерять. Для того чтобы дать возможность работать другим частям кода, следует избегать непрерывного прохода циклов и использовать функцию `sleep`. Далее получаем значения энкодеров и вычисляем ошибку:

```
...
while time.time() < stop_at_time:
    time.sleep(0.01)
    #Вычисляем ошибку
    left = bot.left_encoder.pulse_count
    right = bot.right_encoder.pulse_count
    error = left - right
    ...
```



6. Ошибка обрабатывается регулятором, а затем используется для вычисления `right_speed`:

```
...
# Вычисляем частоту вращения
adjustment = pid.get_value(error)
right_speed = int(speed + adjustment)
left_speed = int(speed - adjustment)
...
```

7. Здесь мы можем записать отладочные данные в журнал. Обратите внимание, что у нас есть два уровня: отладочные данные ошибки и регулировки, а также информация о частоте вращения двигателей. Если текущая конфигурация установлена на INFO, ее необходимо изменить для отображения отладочных данных:

```
...
logger.debug(f"error: {error} adjustment:
{adjustment:.2f}")
logger.info(f"left: {left} right: {right}, left_
speed: {left_speed} right_speed: {right_speed}")
...
```

8. Затем мы устанавливаем частоты вращения двигателей равными значениям регулировки и завершаем цикл:

```
...
bot.set_left(left_speed)
bot.set_right(right_speed)
```

После запуска кода робот должен начать двигаться по прямой. Он может немного отклоняться от заданного направления, но это решается настройкой констант:

```
pi@myrobot:~ $ python3 straight_line_drive.py
DEBUG:pid_controller:P: 0, I: 0.00
INFO:straight_line:left: 3 right: 3, left_speed: 80 right_
speed: 80
DEBUG:pid_controller:P: 0, I: 0.00
INFO:straight_line:left: 5 right: 5, left_speed: 80 right_
speed: 80
DEBUG:pid_controller:P: -4, I: -0.20
INFO:straight_line:left: 5 right: 6, left_speed: 84 right_
speed: 75
DEBUG:pid_controller:P: 0, I: -0.20
...
INFO:straight_line:left: 13 right: 15, left_speed: 89 right_
speed: 71
DEBUG:pid_controller:P: -8, I: -1.40
INFO:straight_line:left: 15 right: 17, left_speed: 89 right_
speed: 70
DEBUG:pid_controller:P: -8, I: -1.80
INFO:straight_line:left: 17 right: 19, left_speed: 89 right_
speed: 70
DEBUG:pid_controller:P: -8, I: -2.20
INFO:straight_line:left: 19 right: 21, left_speed: 90 right_
speed: 69
```

```

...
DEBUG:pid_controller:P: 0, I: 0.60
INFO:straight_line:left: 217 right: 217, left_speed: 79 right_
speed: 80
DEBUG:pid_controller:P: 0, I: 0.60
INFO:straight_line:left: 219 right: 219, left_speed: 79 right_
speed: 80
DEBUG:pid_controller:P: 0, I: 0.60
INFO:straight_line:left: 221 right: 221, left_speed: 79 right_
speed: 80
DEBUG:pid_controller:P: 0, I: 0.60
INFO:straight_line:left: 223 right: 223, left_speed: 79 right_
speed: 80

```

В самом начале ошибки нет, но затем частота вращения правого двигателя увеличивается. На 13 тактах сильно увеличивается значение регулировки. Обратите внимание, как меняется P. Через некоторое время I принимает значение константы, благодаря чему робот начинает двигаться по прямой.

Отстройка констант P и I в времени цикла может привести к более ранним корректировкам – начальные значения энкодеров слишком малы.

Обратите внимание, что робот по-прежнему может отклоняться от прямой. Он старается уменьшить отклонение, но может слишком поздно перестать вносить поправку, поэтому будет двигаться по S-образной траектории или с другой ошибкой. Но все же так робот двигается более прямо. Решить эту проблему отклонений можно путем отстройки регулятора.

## Устранение неполадок

Если робот виляет или ему не удастся двигаться строго по прямой, попробуйте предпринять следующие действия:

- если для выравнивания частоты вращения требуется слишком много времени, увеличьте значение пропорционального компонента;
- если робот выходит за установленные пределы (т. е. сначала сильно отклоняется в одну сторону, а затем в другую), уменьшите пропорциональный и интегральный компоненты ПИД-регулятора;
- если наблюдаются нарастающие колебательные движения робота, это значит, что значение интегрального компонента слишком велико, и значение частоты вращения правого двигателя может быть выше 100. Уменьшите значение интегрального компонента и, при необходимости, запрашиваемую частоту вращения;
- для отслеживания ошибки вы можете установить средство ведения журнала `straight_line` или настроить `basicConfig` для отладки.

На данном этапе мы исправили проблемы, касающиеся отклонения от прямой, и скорректировали разницу скорости вращения двигателей. Теперь, опираясь на полученные знания, можем заставить робота пройти определенное расстояние, а затем остановиться.

## ПЕРЕМЕЩЕНИЕ НА ОПРЕДЕЛЕННОЕ РАССТОЯНИЕ

Для перемещения на определенное расстояние мы снова будем использовать ПИ-регулятор. Затем внедрим измерения расстояния в объект энкодера. Мы вычислим количество тактов, необходимое для поворота колеса на определенный угол, а затем используем полученное значение вместо компонента текущего времени.

### Рефакторинг преобразования единиц измерения для класса EncoderCounter

Нам необходимо переместить преобразования для энкодеров в класс EncoderCounter, чтобы их можно было использовать в наших поведенческих скриптах. Рефакторинг – это процесс перемещения или улучшения кода с сохранением его функционала. В нашем случае преобразование расстояний является одной из главных целей использования энкодеров, поэтому нам нужно переместить код в соответствующий класс.

1. Откройте класс `encoder_counter.py`. Сначала необходимо импортировать библиотеку `math`:

```
from gpiozero import DigitalInputDevice
import math
...
```

2. В верхней части класса добавьте константу `ticks_to_mm_const` как переменную класса (а не как переменную экземпляра), чтобы использовать ее без экземпляров класса. Для того чтобы вычислить ее, сначала нужно установить значение `None`:

```
...
class EncoderCounter:
    ticks_to_mm_const = None #это значение вам нужно установить
    before using distance methods
...
```

3. В классе нам необходимо получать значение пройденного колесом расстояния непосредственно от энкодера, при этом выраженное в миллиметрах. В конце файла добавим следующий фрагмент кода:

```
...
def distance_in_mm(self):
    return int(self.pulse_count * EncoderCounter.
        ticks_to_mm_const)
...
```

Этот код предполагает, что все энкодеры имеют диски одинакового диаметра и с одинаковым количеством пазов. Именно поэтому мы извлекаем `ticks_to_mm_const` из класса, а не из экземпляра (`self`).

4. Также нам необходимо рассчитать количество тактов, приходящихся на расстояние в миллиметрах. Для этого разделите расстояние в миллиметрах на ту же константу, которую до этого мы использовали в умножении. Здесь мы используем `staticmethod`, поэтому для использования экземпляра не требуется дальнейший код:

```
...
    @staticmethod
    def mm_to_ticks(mm):
        return mm / EncoderCounter.ticks_to_mm_const
    ...
```

5. Добавьте способ, позволяющий задавать константы в файле (для разных конфигураций робота):

```
...
    @staticmethod
    def set_constants(wheel_diameter_mm, ticks_per_revolution):
        EncoderCounter.ticks_to_mm_const = (math.pi / ticks_per_revolution) *
        wheel_diameter_mm
    ...
```

После сохранения файла `EncoderCounter` сможет преобразовывать количество тактов энкодера в пройденное расстояние в миллиметрах. Теперь нам нужно задать диаметр колес вашего робота.

## Настройка констант

На данный момент мы можем использовать измеренные параметры робота в поведенческих скриптах. Теперь нам необходимо сохранить эти измеренные значения в объекте `Robot` и связать их с энкодерами. Для этого нужно выполнить два простых шага.

1. В файле `robot.py` непосредственно перед конструктором укажите нужные числа:

```
...
class Robot:
    wheel_diameter_mm = 70.0
    ticks_per_revolution = 40.0
    wheel_distance_mm = 140.0
    def __init__(self, motorhat_addr=0x6f):
    ...
```

2. Свяжите их с энкодерами:

```
...
# Настройка энкодеров
EncoderCounter.set_constants(self.wheel_diameter_
mm, self.ticks_per_revolution)
self.left_encoder = EncoderCounter(4)
self.right_encoder = EncoderCounter(26)
....
```

Настроив константы, мы подготовили энкодеры к измерению расстояния. Теперь с их помощью мы можем создать поведенческий скрипт, предполагающий движение на определенное расстояние.

## Разработка поведенческого скрипта для движения на определенное расстояние

Создадим файл `drive_distance.py` и поместим в него следующий код.

1. Сначала импортируем класс `EncoderCounter`, чтобы использовать хранящиеся в нем измерения, `PIController` и объект `Robot`, а также настроим регистратор журнала:

```
from robot import Robot, EncoderCounter
from pid_controller import PIController
import time
import logging
logger = logging.getLogger("drive_distance")
...
```

2. Определяем функцию `drive_distance`, которая принимает экземпляр робота, расстояние в тактах и произвольно задаваемую частоту вращения, установленную по умолчанию на 80. Начнем с принятия решения о главных и вторичных двигателях и контроллере:

```
...
def drive_distance(bot, distance, speed=80):
    # Левый двигатель - главный, а правый - вторичный
    set_primary = bot.set_left
    primary_encoder = bot.left_encoder
    set_secondary = bot.set_right
    secondary_encoder = bot.right_encoder
    ...
```

Обратите внимание, что функции `set_left` и `set_right` хранятся в переменных – мы можем просто вызывать переменные как функции.

3. Теперь мы четко определили главный и вторичный двигатели. Настроим `PIController` и запустим их:

```
...
controller = PIController(proportional_constant=5, integral_constant=0.3)
# запускаем двигатели и цикл
set_primary(speed)
set_secondary(speed)
...
```

4. Сейчас мы находимся в цикле движения на определенное расстояние. Цикл должен продолжаться до тех пор, пока энкодеры не достигнут нужного расстояния. Перед остальной частью цикла нам необходимо сделать паузу, чтобы получить данные для вычислений:

```
...
while primary_encoder.pulse_count < distance or
secondary_encoder.pulse_count < distance:
    time.sleep(0.01)
...
```

5. Получаем ошибку и загружаем в контроллер:

```
...
# Насколько мы отклонились?
error = primary_encoder.pulse_count - secondary_encoder.pulse_count
adjustment = controller.get_value(error)
...
```

6. Мы можем отправить эти данные двигателям и также зафиксировать отладочные данные. Поскольку значение корректировки не является целым числом, с помощью спецификатора `{:.2f}` мы разрешаем отображать два знака после запятой:

```
...
# Насколько быстро должны вращаться двигатели, чтобы робот достиг нужной
точки?
set_primary(int(speed - adjustment))
set_secondary(int(speed + adjustment))
# Отладка
logger.debug(f"Encoders: primary: {primary_encoder.pulse_count}, secondary: {secondary_encoder.pulse_count}, "
            f"e:{error} adjustment: {adjustment:.2f}")
logger.info(f"Distances: primary: {primary_encoder.distance_in_mm()} мм, secondary: {secondary_encoder.distance_in_mm()} мм")
...
```

7. Далее настраиваем робота и вычисляем нужное расстояние, а затем заставляем его двигаться:

```
...
logging.basicConfig(level=logging.INFO)
bot = Robot()
distance_to_drive = 1000 # в мм - это метр
distance_in_ticks = EncoderCounter.mm_to_ticks(distance_to_drive)
drive_distance(bot, distance_in_ticks)
```

8. Затем останавливаем двигатели с помощью модуля `atexit`:

После запуска этого кода робот проедет расстояние длиной в метр и затем остановится. После остановки показатели моего робота выглядели следующим образом:

```
INFO:drive_distance:Distances: primary: 997 мм, secondary: 991
мм
INFO:drive_distance:Distances: primary: 1002 мм, secondary:
1002 мм
```

Мы видим, что робот проехал на 2 мм больше. Такая погрешность может возникнуть при округлении значений и времени обнаружения фронтов импульсов. Такт может быть только целым числом.

Теперь вы знаете, как заставить робота передвигаться на определенное расстояние (возможно, с небольшой погрешностью), при этом сохраняя прямую траекторию. Вы объединили средства измерений и ПИД-регулировки, созданные в этой главе. Но как заставить робота выполнять повороты и измерять их параметры? Об этом мы поговорим в следующем разделе.

## Выполнение точного поворота

Следующая задача, для которой мы можем использовать энкодеры, – выполнение точного поворота. Когда робот поворачивает, каждое колесо проходит по дуге, как показано на рис. 11.13.

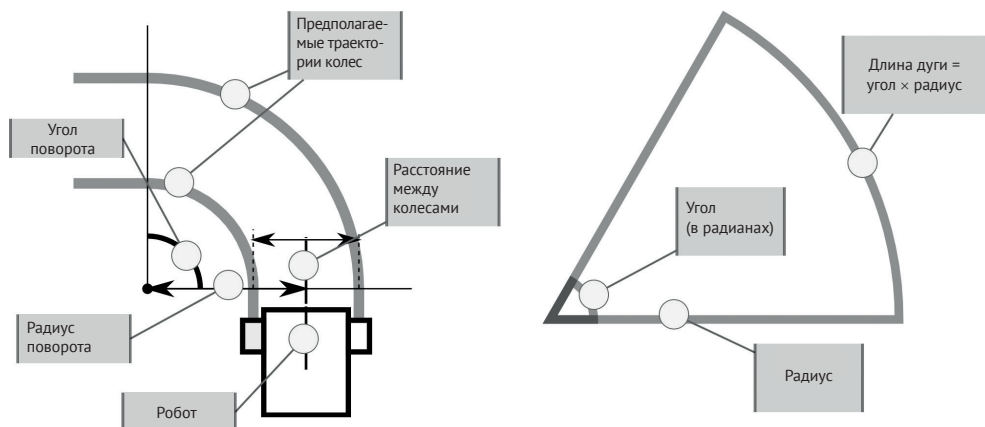


Рис. 11.13. Движение колес по дуге при повороте

Согласно базовым принципам дифференциального рулевого управления, при повороте внутреннее колесо описывает дугу меньшего радиуса, а внешнее – большего. Для выполнения точного поворота нам потребуется вычислить длину описываемой дуги для каждого колеса или отношение между ними. На рис. 11.14 показано, как колеса влияют на поворот.

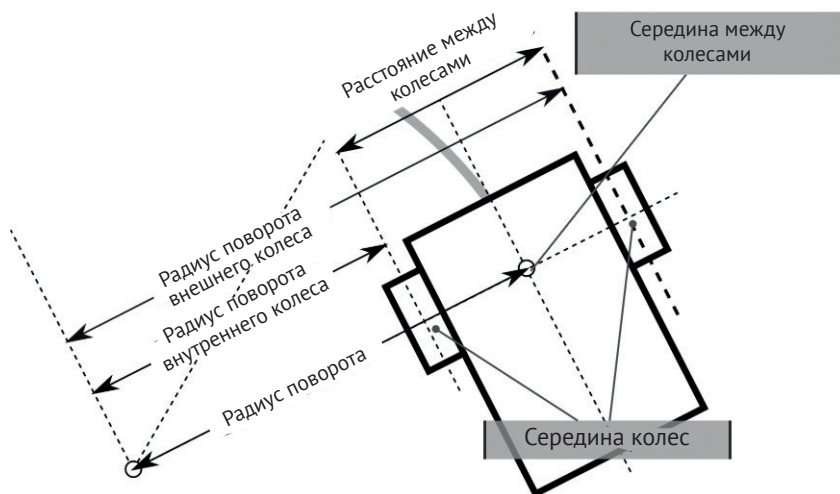


Рис. 11.14. Влияние колес на радиусы поворота



Если представить радиус поворота как параметр, отражающий центр поворота робота, то радиус поворота внутреннего колеса будет равен *разнице* между радиусом поворота и половиной расстояния между осями колес:

$$\text{радиус\_поворота\_внутреннего\_колеса} = \text{радиус\_поворота} - \frac{\text{расстояние\_между\_осями\_колес}}{2}.$$

Радиус поворота внешнего колеса будет равен сумме угла поворота и половины расстояния между осями колес:

$$\text{радиус\_поворота\_внутреннего\_колеса} = \text{радиус\_поворота} + \frac{\text{расстояние\_между\_осями\_колес}}{2}.$$

Мы переводим угол в радианы, а затем умножаем это значение на радиус поворота каждого колеса, чтобы получить расстояние, которое должно пройти каждое колесо:

$$\text{расстояние\_для\_внешнего\_колеса} = \text{угол\_в\_радианах} * \text{радиус\_поворота\_внешнего\_колеса}$$

$$\text{расстояние\_для\_внутреннего\_колеса} = \text{угол\_в\_радианах} * \text{радиус\_поворота\_внутреннего\_колеса}$$

Для перевода градусов в радианы в языке Python существует специальная математическая функция.

Разработаем демонстрационный код, под управлением которого робот будет двигаться по квадратной траектории и выполнять точные повороты на 90°.

1. Сделайте копию файла `drive_distance.py` и назовите ее `drive_square.py`. Импортируйте библиотеку `math`:

```
from robot import Robot, EncoderCounter
from pid_controller import PIController
import time
import math
import logging
logger = logging.getLogger("drive_square")
...
```

2. В конце этого файла мы указываем, что нам нужно. Здесь мы задаем имена необходимых функций, чтобы впоследствии реализовать их соответствующим образом. Расстояние снова задаем чуть меньше метра. Чтобы получить тестовый радиус, я прибавил к расстоянию между осями колес робота 100 мм. Все, что меньше расстояния между осями колес и центром поворота, находится между колесами, а не за ними:

```
...
bot = Robot()

distance_to_drive = 300 # в мм
distance_in_ticks = EncoderCounter.mm_to_ticks(distance_to_drive)
radius = bot.wheel_distance_mm + 100 # в мм
radius_in_ticks = EncoderCounter.mm_to_ticks(radius)
...
```

3. Поскольку робот будет двигаться по квадратной траектории, ему потребуется проехать четыре отрезка. Для того чтобы робот двигался по прямой, каждое колесо должно проехать одинаковое расстояние. Затем мы задаем дуги  $90^\circ$  для нашего радиуса. Во избежание проскальзывания уменьшим скорость при движении по дуге:

```
...
for n in range(4):
    drive_distances(bot, distance_in_ticks, distance_in_ticks)
    drive_arc(bot, 90, radius_in_ticks, speed=50)
```

4. Затем мы возвращаемся, чтобы задать движение на два определенных расстояния. Здесь я изменил имя функции `drive_distance` на `drive_distances`:

```
...
def drive_distances(bot, left_distance, right_distance,
speed=80):
    ...
```

5. В зависимости от желаемого угла поворота любой двигатель может быть внешним, а его колесо будет проходить большее расстояние. Поскольку существует верхний предел частоты вращения, мы выбираем главный и вторичный двигатели на основе большего расстояния. Замените фрагмент кода, назначающий главный/вторичный двигатели на следующий фрагмент:

```
...
# нам нужно, чтобы расстояние, проходимое «главной» стороной, было
# больше, следовательно, соответствующий двигатель должен вращаться быстрее
if abs(left_distance) >= abs(right_distance):
    logger.info("левая сторона - главная")
    set_primary = bot.set_left
    primary_encoder = bot.left_encoder
    set_secondary = bot.set_right
    secondary_encoder = bot.right_encoder
    primary_distance = left_distance
    secondary_distance = right_distance
else:
    logger.info("правая сторона - главная")
    set_primary = bot.set_right
    primary_encoder = bot.right_encoder
    set_secondary = bot.set_left
    secondary_encoder = bot.left_encoder
    primary_distance = right_distance
    secondary_distance = left_distance
primary_to_secondary_ratio = secondary_distance /
primary_distance
secondary_speed = speed * primary_to_secondary_ratio
logger.debug("Targets - primary: %d, secondary: %d,
ratio: %.2f" % (primary_distance, secondary_distance,
primary_to_secondary_ratio))
...
```

Энкодеры и двигатели не изменились, однако теперь для принятия решения в коде используется абсолютное значение `abs`, поскольку для об-

ратного хода большее расстояние должна *по-прежнему* проходить главная сторона. Таким образом, чтобы определить, какое расстояние должна пройти вторичная сторона, мы вычисляем соотношение – умножаем на текущую скорость, а затем на выходные данные главного энкодера.

6. Поскольку мы используем этот метод несколько раз, сбросим счетчики энкодера. Я сделал это перед настройкой PIController:

```
...
    primary_encoder.reset()
    secondary_encoder.reset()

    controller = PIController(proportional_constant=5,
                              integral_constant=0.2)
...
```

7. Наш робот может двигаться в любом направлении, поэтому зададим направление энкодера. В Python существует метод `copysign`, который определяет знак значения. Затем запустим двигатели:

```
...
    # задаем энкодеру направление
    primary_encoder.set_direction(math.copysign(1,
speed))
    secondary_encoder.set_direction(math.copysign(1,
secondary_speed))
    # запускаем двигатель и цикл
    set_primary(speed)
    set_secondary(int(secondary_speed))
...
```

8. При запуске цикла нам необходимо отдавать себе отчет в том, что один или оба двигателя могут вращаться в обратную сторону. Чтобы убрать знак, снова используем `abs`:

```
...
    while abs(primary_encoder.pulse_count) < abs(primary_
distance) or abs(secondary_encoder.pulse_count) <
abs(secondary_distance):
        time.sleep(0.01)
...
```

9. Ошибка для вторичного двигателя вычисляется как отношение между двумя расстояниями:

```
...
    # Насколько мы отделились?
    secondary_target = primary_encoder.pulse_count *
primary_to_secondary_ratio
    error = secondary_target - secondary_encoder.
pulse_count
    adjustment = controller.get_value(error)
...
```

10. Мы снова переходим к расчету значения регулировки с помощью `pid`, однако здесь значение регулировки может вызвать изменение направления. Теперь мы устанавливаем частоту вращения вторичного двигателя:

```

...
# Насколько быстро должны вращаться двигатели, чтобы робот достиг нужной
точки?
set_secondary(int(secondary_speed + adjustment))
secondary_encoder.set_direction(math.copysign(1,
secondary_speed+adjustment))

# Отладка
logger.debug(f"Encoders: primary: {primary_encoder.
pulse_count}, secondary: {secondary_encoder.pulse_count},
e:{error} adjustment: {adjustment:.2f}")
logger.info(f"Distances: primary: {primary_encoder.
distance_in_mm()} мм, secondary: {secondary_encoder.
distance_in_mm()} мм")
...

```

11. Вы можете расширить отладку, которую мы должны были учитывать для скорости и целей вторичного двигателя. Теперь, поскольку мы стремимся к точности, главный двигатель может достичь своей цели раньше вторичного. Также он не должен быть настроен на обратный ход. Остановите этот двигатель, когда он достигнет цели, а затем установите константу частоты вращения вторичного двигателя на 0 – это будет означать, что применяются только значения регулировки (при наличии). Обратите внимание, что мы по-прежнему используем абсолютные значения:

```

...
# Stop the primary if we need to
if abs(primary_encoder.pulse_count) >=
abs(primary_distance):
    logger.info("primary stop")
    set_primary(0)
    secondary_speed = 0
...

```

Мы закончили с функциями дистанции движения. Теперь можно использовать такие функции для движения по прямой или для движения по дуге, указав свое расстояние до цели для каждого колеса. О последнем мы поговорим в следующем разделе.

## Создание функции drive\_arc

Здесь мы преобразуем градусы в радианы, определяем внутренний радиус и устанавливаем расстояния для каждого колеса. Добавим следующий фрагмент кода в файл `drive_square_behaviour.py` после функции `drive_distances`:

Начнем с определения функции и строки документации:

```

...
def drive_arc(bot, turn_in_degrees, radius, speed=80):
    """ Поворот основывается на изменении направления. """
    ...

```

1. Далее преобразовываем ширину робота в такты – внутреннюю единицу измерения расстояния. Половина полученного значения будет являться радиусом поворота колеса. Затем определяем, какое колесо является внутренним:

```

...
# Преобразуем ширину в такты
half_width_ticks = EncoderCounter.mm_to_ticks(bot.
wheel_distance_mm/2.0)
if turn_in_degrees < 0:
    left_radius = radius - half_width_ticks
    right_radius = radius + half_width_ticks
else:
    left_radius = radius + half_width_ticks
    right_radius = radius - half_width_ticks
logger.info(f"Arc left radius {left_radius:.2f},
right_radius {right_radius:.2f}")
...

```

2. Отобразив отладочные данные, мы увидим радиусы. Чтобы получить расстояния, умножим радиусы на поворот в радианах. Мы не хотим, чтобы робот повернул на месте за счет вращения одного мотора в обратную сторону. Нам нужен плавный поворот по дуге:

```

...
radians = math.radians(abs(turn_in_degrees))
left_distance = int(left_radius * radians)
right_distance = int(right_radius * radians)
logger.info(f"Arc left distance {left_distance},
right_distance {right_distance}")
...

```

3. Наконец, передаем полученные расстояния в функцию `drive_distances`:

```

...
drive_distances(bot, left_distance, right_distance,
speed=speed)
...

```

Теперь робот должен уметь двигаться по квадратной траектории. Из-за проскальзывания колес или неточностей в измерениях он может немного отклоняться от заданного пути, поэтому необходимо отстроить управляющие значения пропорционального и интегрального компонентов.

Если просмотреть коды для функций `drive_distances` и `drive_arc`, вы увидите, что в них встречаются повторы (например, при определении внутренней/внешней и главной/второстепенной частей), поэтому при желании вы можете выполнить рефакторинг кода.

При движении по углу задним ходом этот код может работать неправильно.

## Выводы

В этой главе мы оснастили нашего робота энкодерами и научились использовать их для определения того, на какой угол повернулось каждое колесо. Мы узнали, как заставить робота двигаться по прямой с помощью усеченного ПИД-регулятора, а затем научили его проезжать определенное расстояние. Затем продолжили расчеты и вычислили значения для колес, позволяющие выполнить поворот в углах при движении по квадратной траектории.

ПИД-регулятор можно использовать во многих ситуациях, когда вам нужно применить разницу между измерением и ожидаемым значением. В этой

главе вы узнали, как это работает с датчиками. Вы можете использовать ту же систему для управления нагревательным элементом, подключенным к термодатчику. Также энкодеры можно использовать для роботов, движение которых должно быть точным (в случаях, когда ограниченный диапазон движения, используемый в серводвигателях, не имеет смысла).

В следующих нескольких главах мы сделаем нашего робота более интерактивным и интеллектуальным. Эти главы посвящены обработке изображений с помощью камеры Raspberry Pi, обработке речи с помощью Microsoft и реализации удаленного управления или выбора режимов на роботе через смартфон.

## Задание

1. Попробуйте поэкспериментировать, включив разные уровни журналирования и регистраторы с разными именами, настраивая объем выходных данных поведенческого сценария робота.
2. Для поведенческих сценариев с ПИД настройте ПИД-регуляторы. Попробуйте увеличить коэффициенты для пропорционального или интегрального компонентов и наблюдайте, как это влияет на поведение робота. Для удобства можете построить график в `matplotlib`.
3. Есть несколько способов оптимизировать наш код. Мы можем применить **ПИД-регулятор** для определения расстояния, пройденного главной стороной, что позволит рассчитать более точное расстояние, которое нужно пройти. Чтобы робот не пытался продолжить движение после принудительной остановки, попробуйте реализовать обнаружение отсутствия движения в любом энкодере для остановки кода.
4. Теперь вы можете использовать этот код, чтобы создавать траектории движения по другим геометрическим фигурам или следовать пути без линии. Попробуйте добавить высокоуровневые функции поворота влево/вправо на  $90^\circ$  в качестве строительных блоков для построения траектории движения под прямым углом, а затем используйте их для создания траекторий.
5. Поразмышляйте об объединении энкодеров и датчиков расстояния. Посредством такого подхода можно начать сохранять данные о расстоянии между стенами.

## Дополнительные материалы

- ПИД-регулирование – весьма обширная тема. Данная область является ключевой при разработке самобалансирующихся роботов, дронов и других автономных систем. Ниже приведена ссылка на серию отличных видеоматериалов, которые помогут вам углубиться в эту тему:

YouTube: Брайан Дуглас (Brian Douglas) – «*PID Control – A brief introduction*»: <https://www.youtube.com/watch?v=UR0hOmjaHp0>.

- В этой главе я значительно упростил некоторые алгоритмы поворота. Вот подробная статья, автор которой использует эти алгоритмы для робота, победившего в конкурсе LEGO Mindstorms:

Дж. В. Лукас (G. W. Lucas) GW Lucas – «*Using a PID-based Technique For Competitive Odometry and Dead- Reckoning*»: [https://archive.seattlerobotics.org/encoder/200108/using\\_a\\_pid.html](https://archive.seattlerobotics.org/encoder/200108/using_a_pid.html).



# Глава 12

## Код на Python для работы с IMU

Современным роботам необходимо знать свое положение в пространстве. В главе 11 мы узнали, как с помощью энкодеров можно определить, на какое расстояние переместился робот и как он повернулся. Однако эти данные были относительными, а не абсолютными, т. е. опирались на определенную точку, где робот находился до этого. При этом на точность измерений могло значительно влиять проскальзывание колес. В этой главе мы рассмотрим другие способы, посредством которых робот может определять изменение положения и измерять параметры движения.

В теории мы легко можем определить абсолютное положение робота с помощью **инерциального измерительного модуля (IMU – Inertial Measurement Unit)**, и при этом на точность измерений не будет влиять проскальзывание колес. Но на практике такие системы довольно сложны. Эта глава представляет собой небольшой практический обзор на тему того, как добавить IMU в систему вашего робота. Здесь я расскажу об основных компонентах этого модуля. Также вы научитесь припаивать разъемы к плате модуля. В дальнейшем этот навык понадобится вам для оснащения робота самыми разными компонентами.

Мы разработаем код, чтобы опробовать различные функции и просмотреть выходные данные датчиков, которые затем визуализируем на информационной панели. К концу этой главы вы научитесь работать с продвинутыми датчиками, паять и реализовывать информационные панели для мониторинга датчиков. По мере углубления в робототехнику вы узнаете, что использование таких анимированных панелей является лучшим способом увидеть то, что видит ваш робот.

В этой главе мы рассмотрим следующие темы:

- подробнее об инерциальных измерительных модулях (IMU);
- пайку – подключение контактов к IMU;
- установку IMU на робота;
- считывание данных датчика температуры;
- считывание данных гироскопа;
- считывание данных акселерометра;
- считывание данных магнитометра.

## ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам потребуются:

- робот из главы 7;
- код, разработанный в главе 11 (его вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter11>);
- отладочная плата с IMU ICM20948, например модуль Pimoroni PIM448;
- паяльник и подставка;
- очиститель жала паяльника;
- припой (выбирайте припой с флюсовым сердечником для пайки электронных схем);
- отсос припоя;
- хорошо освещенный рабочий стол;
- вентилируемое помещение или вытяжка;
- защитные очки;
- макетная плата;
- набор стоек 2,5 мм;
- гибкие соединительные провода Dupont с разъемами «гнездо–гнездо».

Полный код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter12>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/38FJgsr>.

## ПОДРОБНЕЕ ОБ ИНЕРЦИАЛЬНЫХ ИЗМЕРИТЕЛЬНЫХ МОДУЛЯХ (IMU)

IMU представляет собой набор датчиков, предназначенных для отслеживания движения робота в трехмерном пространстве. Они могут использоваться как в дронах, так и в роботах, предназначенных для передвижения по поверхности. Кроме того, эти устройства важны для балансировки роботов. IMU – это не один датчик, а набор датчиков, предназначенных для совместного использования и объединения показаний.

Сейчас эти устройства миниатюрные, но они уходят корнями в авиационное оборудование с огромными вращающимися гироскопами. В IMU используется технология **микроэлектромеханических систем (МЭМС)**, объединяющая механические устройства и микрочипы. Модули имеют крошечные движущиеся части и измеряют параметры их движения с помощью электронных датчиков.

Поскольку некоторые измерения являются аналоговыми (см. главу 2), IMU часто имеют **аналого-цифровой преобразователь (АЦП)** и обмениваются данными через шину I2C.

IMU могут включать в себя различные датчики, такие как:

- датчик температуры, предназначенный для учета влияния температуры на другие датчики;
- гироскоп, измеряющий угловую скорость вращения;

- акселерометр, измеряющий ускорение под действием внешних сил;
- магнитометр, служащий для измерения характеристик магнитного поля, который может работать как компас.

Работая с каждым из этих датчиков, мы узнаем больше об их особенностях.

Теперь, когда мы познакомились с IMU, давайте разберемся, как выбрать подходящий модуль.

## Предлагаемые модели IMU

IMU могут иметь отдельные акселерометр, гироскоп и магнитометр, а также устройства для преобразования выходного сигнала датчика. В целях экономии пространства и упрощения соединений я предлагаю использовать плату, включающую в себя сразу все устройства, или решение на одной микросхеме. По этой же причине я рекомендую IMU, поддерживающие передачу данных через I2C или последовательную шину.

Системы IMU имеют разные **степени свободы (degrees-of-freedom – DOF)**, которые указывают на количество осей датчиков. Система с 9 степенями свободы (9-DOF) имеет три оси (X, Y и Z) для каждого датчика.

Для датчиков BNO разработать код проще всего, но они несовместимы с Raspberry Pi из-за того, как они используют шину I2C. Также для них может потребоваться промежуточная интерфейсная микросхема.

Еще одна вещь, которую следует учитывать, – это наличие документации (файлы readme и руководства) и поддерживаемой библиотеки для управления устройством с помощью Python. На рис. 12.1 показано, как выглядит предлагаемая коммутационная плата IMU.



Рис. 12.1. Модуль ICM20948

На рис. 12.1 показана отладочная плата PIM448 для ICM20948 – модуль с 9 степенями свободы, с широкой поддержкой библиотек Python. Также здесь предусмотрен датчик температуры, имеющий широкую поддержку. Поскольку IMU – это сложные устройства, я настоятельно рекомендую использовать для нашего проекта именно PIM448.

Теперь, когда мы узнали, что представляет собой устройство IMU и как выбрать подходящее, пришло время подготовить PIM448 для нашего робота. Для этого нужно освоить новый навык – пайку.

## ПАЙКА – ПОДСОЕДИНЕНИЕ КОНТАКТОВ К IMU

В комплекте с большинством коммутационных плат для IMU, в том числе и с PIM448, поставляются контакты, которые необходимо припаять. Узнаем, как это сделать.

На рис. 12.2 показан комплект поставки PIM448. Слева мы видим плату ICM20948, на которой есть только разъемы, в центре показаны контакты типа «штекер», а справа – контакты типа «гнездо». Мы будем использовать контакты типа «штекер», поскольку их легче припаивать.



Рис. 12.2. Исходный модуль PIM448 и два варианта разъемов

Как упоминалось в разделе «Технические требования», нам понадобится паяльник и припой, подставка для паяльника, защитные очки, вытяжка или хорошо вентилируемое помещение, дополнительная макетная плата и хорошо освещенное рабочее пространство. При пайке образуются пары, вдыхать которые опасно.

На этом этапе наденьте защитные очки. Затем нагрейте паяльник (это может занять несколько минут) и подготовьте припой.

### Создание паянного соединения

Приступим к пайке.

На рис. 12.3 процесс пайки модуля показан пошагово.

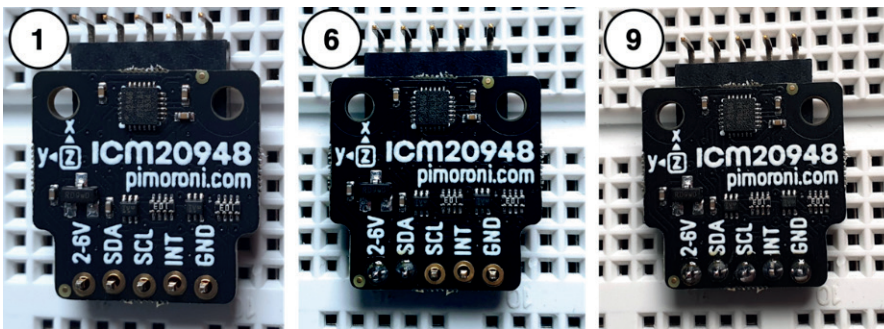


Рис. 12.3. Этапы пайки PIM448

Чтобы создать паянное соединение, выполните следующие шаги (обращаясь к рис. 12.3).

1. Нам необходимо, чтобы компонент оставался неподвижным, пока припой остывает. На рис. 12.3 мы видим, что модуль PIM448 надет на контакты типа «штекер», которые входят в макетную плату снизу. Отличный способ зафиксировать компонент – вставить длинную сторону разъемов в макетную плату так, чтобы наше устройство было сверху; для этого мы используем контакты типа «штекер». Контакты типа «гнездо» мы используем в качестве опоры с другой стороны.
2. Жало паяльника нужно нагреть (примерно до 300 °C) и залудить. Чтобы проверить, достаточно ли нагрелось жало, расплавьте на нем небольшое количество припоя. Прежде чем перейти к пайке, нужно залудить жало. Лужение – это процесс нанесения на жало паяльника тонкого слоя припоя с целью улучшения теплопроводности и защиты от окисления (ржавления при нагревании). Чтобы залудить жало, нанесите на него немного припоя так, чтобы он прилип. Припой должен легко плавиться.
3. Очистите жало. Для этого нагретое жало вставьте в латунный очиститель и «протрите» его о проволоку (иногда в комплекте с паяльником идет специальная термостойкая губка, которую можно слегка смочить водой или глицерином).
4. Нагрейте вывод контакта и контактную площадку (металлизированное отверстие, через которое проходит вывод). Мы начнем с контакта **2-6V**. Чтобы получить хорошо пропайное соединение, нужно нагреть и вывод, и контактную площадку, иначе припой не будет должным образом заполнять ее. Непропаенные соединения слабы как с точки зрения электроники, так и механики.
5. Через секунду (или около того) нанесите немного припоя с обратной стороны. Когда вывод достаточно нагреется, припой расплавится и заполнит контактную площадку, образуя форму, напоминающую шатер. Этого припоя будет достаточно. Вы увидите, как из припоя вытекает флюс.
6. В середине на рис. 12.3 показан следующий шаг. Здесь я припаял два вывода. С этого момента работать становится проще, поскольку плата уже зафиксирована. Переходим к следующему контакту и повторяем действие – нагреваем вывод и контактную площадку, затем наносим припой.
7. Если вы нанесли слишком много припоя, воспользуйтесь отсосом и уберите излишки. Для этого опустите поршень, поднесите отсос к соединению, расплавьте припой и нажмите на спусковую кнопку поршня. Теперь можно припаять контакт заново, но с меньшим количеством припоя.
8. Если вы соединили одной каплей припоя два вывода (сделали между ними перемычку), их можно разделить, проведя между ними горячим паяльником. Возможно, необходимо будет убрать лишний припой (как описано в шаге 7).
9. Повторите шаги для оставшихся контактов. Справа на рис. 12.3 показано, как должен выглядеть IMU с припаянными контактами.

**Важное примечание**

Помните о безопасности. По завершении работы поместите паяльник на подставку и выключите его. Неосторожное обращение может привести к ожогам или возгоранию.

10. Когда наш модуль остынет, отсоедините его от макетной платы. Чтобы убрать остатки флюса, протрите место пайки ватной палочкой, смоченной в изопропиловом или этиловом спирте.

Прежде чем мы перейдем к подключению, проверьте следующее:

- все выводы должны быть припаяны;
- каждый припаянный вывод должен быть накрыт «шатром» из припоя. Если припоя слишком много или слишком мало, необходимо переделать соединение;
- между выводами не должны образовываться *перемычки* (соединяющие их капли припоя).

Поздравляем – вы выполнили пайку своего первого компонента! Этот навык пригодится вам при работе с другими роботизированными или электронными устройствами. Теперь, когда вы спаяли модуль ICM20948, давайте добавим его к нашему роботу.

## УСТАНОВКА IMU НА РОБОТА

Прежде чем мы сможем приступить к работе с IMU и разработке кода, необходимо надежно установить его на шасси робота и подключить к Raspberry Pi.

### Физическое размещение

Магнитометр IMU чувствителен к магнитным полям и должен находиться вдали от двигателей. По этой причине модуль обычно размещают над роботом, устанавливая на специальных стойках из немагнитного материала.

Ориентация датчиков IMU имеет критически важное значение.

На рис. 12.4 показана схема, описывающая, как IMU должен быть расположен на роботе. Ось X должна быть направлена вперед, ось Z – вверх (при этом маленький квадрат на IMU должен быть обращен вверх), а ось Y – влево.

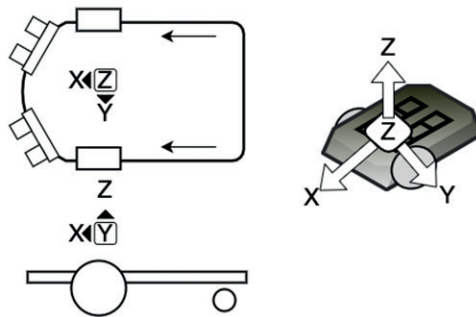
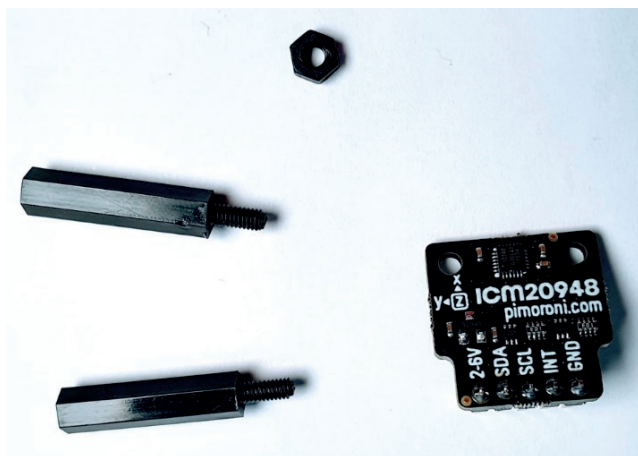


Рис. 12.4. Схема размещения IMU на роботе

Модуль использует шину I2C, в связи с чем важным условием является небольшая длина провода, поэтому мы установим его над Raspberry Pi и платой управления двигателем. На рис. 12.5 показаны необходимые для установки компоненты.





**Рис. 12.5.** Компоненты, необходимые для установки IMU

На этом этапе нам понадобятся (см. рис. 12.5):

- IMU с припаянными контактами;
- стойки M2,5;
- гайка M2,5.

С помощью этих компонентов мы сделаем длинную стойку, как показано на рис. 12.6.

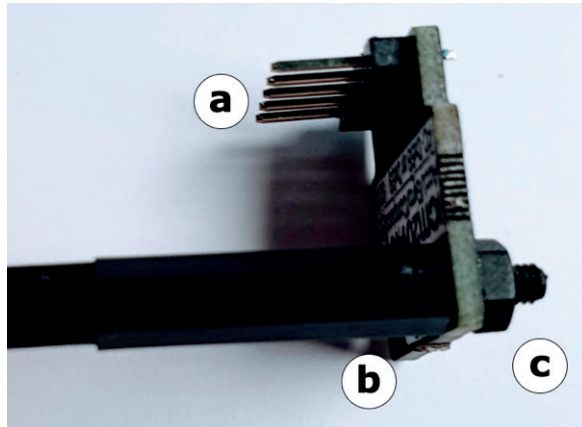


**Рис. 12.6.** Создание длинной стойки

Чтобы собрать длинную стойку, выполните следующие шаги.

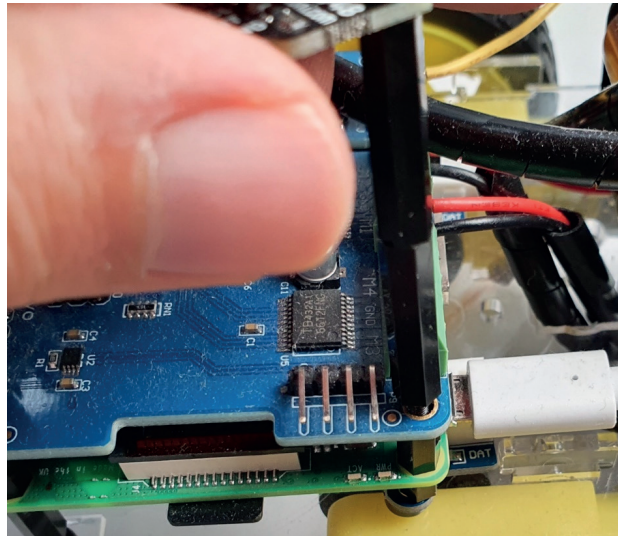
1. Как показано на рис. 12.6, вам нужно просто вкрутить резьбу одной стойки в отверстие другой, чтобы в итоге получилась одна длинная стойка. Так IMU будет находиться над роботом. Длина стойки должна быть чуть меньше длины провода.
2. Как показано на рис. 12.7, вставьте резьбу стойки в отверстие напротив маркеров осей IMU (a). Контакты (b) должны быть обращены вниз к стойке. Резьба входит в отверстие достаточно плотно. Закрепите конструкцию гайкой (c).





**Рис. 12.7.** Установка IMU на стойку

3. На рис. 12.8 показана стойка с IMU, ввинченная в резьбу стойки, торчащей из платы двигателя. Слева в задней части платы двигателя, о которой мы говорили в главе 6, есть разъем I2C. Мы можем разместить стойку IMU ближе к этому разъему.



**Рис. 12.8.** Установка стойки IMU на Raspberry Pi

4. На рис. 12.9 показан ICM20948, установленный на стойку, размещенную поверх платы двигателя. Выводы модуля готовы к подключению. Отрегулируйте конструкцию так, чтобы ось X была направлена вперед, а ось Y влево. Затем затяните гайку. Чем ближе модуль расположен к геометрическому центру робота, тем точнее будут результаты измерений!

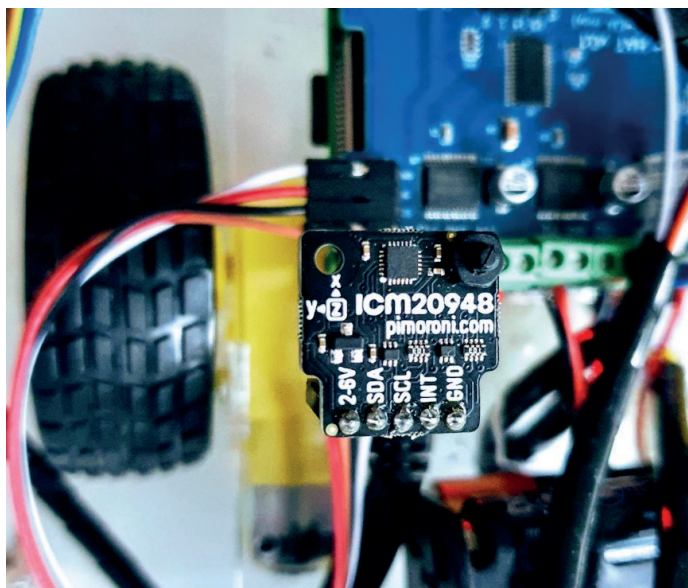


Рис. 12.9. IMU, готовый к подключению

Итак, IMU установлен на робота. Мы выровняли оси, поэтому уверены в точности измерений. Модуль надежно закреплен, но при необходимости его можно снять. На этом этапе он готов к подключению.

## Подключение IMU к Raspberry Pi

Теперь нам необходимо подключить IMU к выводам I2C на Raspberry Pi. На первый взгляд это кажется простой задачей, но главная сложность заключается в том, что некоторые подключения проходят не напрямую.

Разъем I2C на плате двигателя достаточно удобный, что облегчает задачу.

На рис. 12.10 показано что подключение устроено совсем несложно: **GND** от IMU идет к **GND** на разъеме I2C платы двигателя, **SDA** идет к **SDA**, **SCL** идет к **SCL**, а **2-6V** к **5V** (в пределах диапазона).

**GND** на плате двигателя находится слева и идет к **GND** на IMU, который находится справа. Четыре провода образуют изгибы, которые пересекает линия **5V**.

На практике у нас получается подключение из четырех соединительных проводов (на схеме показано пунктирными линиями). Провода, подключенные к IMU, идут без пересечений. На конце, идущем к плате двигателя, другие провода пересекает провод питания (см. рис. 12.11).

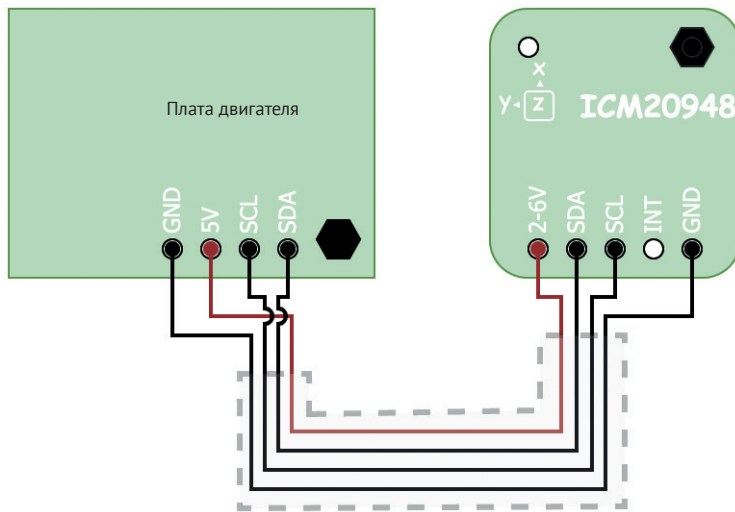


Рис. 12.10. Подключение ICM20948

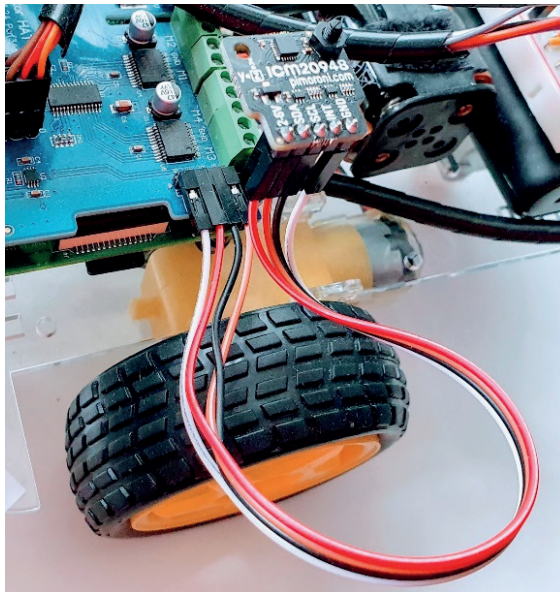


Рис. 12.11. ICM20948, подключенный к плате двигателя

Как видно на рис. 12.11, я использовал короткие соединительные провода с разъемами «гнездо–гнездо». Для наглядности плата IMU расположена под углом  $90^\circ$  от правильной ориентации. Ось X должна быть обращена вперед. Обратите внимание, что здесь образуется перекрещивание проводов, поэтому линия GND (провод белого цвета) подходит к контакту GND с другой стороны. Для реализации этих подключений выполните следующие шаги.

1. Аккуратно отделите четыре провода. Провод GND должен быть самого темного цвета, а 5V – самого яркого.
2. Подключите одну сторону напрямую к IMU, пропустив контакт INT.
3. Когда вы подведете провода к плате двигателя снизу, поверните их так, чтобы они смотрели в другую сторону.
4. Сначала подключите GND, чтобы задать ориентацию.
5. Затем подключите линию 5V, которая должна пересечь два других провода.
6. Последние два провода теперь должны быть правильно ориентированы для SDA и SCL. Подключите их.
7. Проверьте правильность подключения (по цвету проводов).

Здесь мы не задействуем контакт INT. Он предназначен для отправки на Pi сигнала прерывания, уведомляющего о движении при поведенческих сценариях типа «пробуждение при движении» (wake-on-motion). Данная тема выходит за рамки этой книги.

Теперь, когда мы установили IMU на робота и подключили его, можно перейти к созданию кода. Начнем с простого – считывания температуры.

## СЧИТЫВАНИЕ ДАННЫХ ДАТЧИКА ТЕМПЕРАТУРЫ

Теперь, когда устройство установлено и подключено, необходимо протестировать на нем код, чтобы убедиться, что мы можем общаться с ним и получать данные. Установим нужные инструменты и заставим их работать.

## Установка программного обеспечения

Прежде чем начать взаимодействовать с модулем, как и для связи с большинством устройств, для него нам понадобится вспомогательная библиотека. Pimoroni, поставщики модуля ICM20948, который я предложил использовать в нашем проекте, разработали удобную библиотеку для Python. Я рекомендую использовать последнюю версию, опубликованную на GitHub.

Для установки библиотеки выполните следующие шаги.

1. Запустите Raspberry Pi. Ранее мы подключали к нему плату двигателя и светодиодную линейку, поэтому шина I2C должна быть включена. В ином случае вернитесь к главе 7 и подготовьте I2C.
2. Проверьте, что устройство определено верно. Для этого введите в терминале команду `i2cdetect -y 1`. Вывод должен выглядеть следующим образом:

```
pi@myrobot:~ $ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  68  --  --  --  --  --  6f
70: 70  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

3. Мы видим, что наше устройство имеет адрес 0x68. Если у вас получился другой результат, выключите Raspberry Pi и проверьте подключения. Два других устройства (0x6f и 0x70) – это плата двигателя и светодиодная линейка.

4. Далее устанавливаем библиотеку:

```
pi@myrobot:~ $ git clone https://github.com/pimoroni/
icm20948-python
pi@myrobot:~ $ cd icm20948-python/
pi@myrobot:~ $ sudo ./install.sh
pi@myrobot:~ $ cd
```

5. Теперь мы знаем, что устройство ICM20948 подключено к шине I2C. Мы установили библиотеку Python от Pimoroni, с помощью которой робот будет взаимодействовать с устройством. Сейчас мы готовы пообщаться с ним.

Также мы добавим новое программное обеспечение для визуализации данных в режиме реального времени. Существует система под названием **Visual Python (VPython)**, предназначенная для создания графиков и трехмерных изображений в реальном времени:

```
pi@myrobot:~ $ pip3 install git+https://github.com/orionrobots/
vpython-jupyter.git
```

На этом этапе мы установили устройство и библиотеку. При возникновении проблем обратитесь к следующему разделу «Устранение неполадок».

## Устранение неполадок

На этом этапе что-то может пойти не так. Чтобы это исправить, выполните следующие шаги.

1. Критически важным является этап с командой `i2cdetect`. В выводе мы должны видеть устройство 0x68. Если вы видите другой результат, проверьте подключения. Провода *не должны* нагреваться.
2. Проверьте все паяные соединения.
3. Если библиотеки не устанавливаются, проверьте подключение к интернету. Для работы библиотек может потребоваться новейшая версия Raspbian.

Теперь, когда мы установили устройство и устранили возможные проблемы, можно провести первый эксперимент и считать показания датчика температуры.

## Считывание регистра температуры

В этом разделе мы настроим интерфейс для IMU, а затем создадим график, отображающий данные температуры от Raspberry Pi в реальном времени.

### Создание интерфейса

Как и в случае с другими датчиками и выводами, на рынке существует множество IMU. Благодаря универсальному интерфейсу мы сможем менять их без необходимости переписывать поведенческие скрипты.

1. Создайте файл с именем `robot_imu.py`.
2. Начнем с импорта библиотеки для устройства Pimoroni (если вы используете другой IMU, код на этом шаге будет отличаться):

```
from icm20948 import ICM20948
```

3. Для представления нашего устройства мы создадим класс IMU. Он устанавливает один IMU:

```
class RobotImu:
    def __init__(self):
        self._imu = ICM20948()
```

4. Для нашего эксперимента нужны только данные о температуре. Для этого просто обернем метод:

```
def read_temperature(self):
    return self._imu.read_temperature()
```

Теперь наш интерфейс готов, и мы можем использовать его для считывания температуры устройства.

### Что такое VPython?

VPython, или Visual Python, – это система, предназначенная для создания визуальных (в том числе трехмерных) представлений в Python. Она создана научным сообществом и очень пригодится нам в этой главе. VPython выводит данные в браузере, а с установленной здесь конкретной версией ее можно запустить на Raspberry Pi, отображая вывод на компьютере или смартфоне. У нее есть несколько особенностей, например медленный запуск, но результат того стоит.

### Создание графика температуры

Наблюдать за изменением температуры удобно, отображая ее на графике.

Давайте воспользуемся VPython и создадим график, показывающий температуру нашего IMU.

1. Создайте файл с именем `plot_temperature.py`.
2. Начнем с импорта VPython и интерфейса IMU:

```
import vpython as vp
from robot_imu import RobotImu
```

Обратите внимание, что мы импортировали `vpython` с сокращенным именем как `vp`.

3. Нам необходимо отобразить зависимость температуры от времени на графике, поэтому понадобится источник меток времени. Кроме того, мы будем использовать ведение журнала:

```
import time
import logging
```

4. Настройте ведение журнала так, чтобы видеть все записи уровня INFO:

```
logging.basicConfig(level=logging.INFO)
```



5. Создайте экземпляр IMU:

```
imu = RobotImu()
```

6. В графике нам необходимо отразить несколько факторов. На оси X отражено время в секундах, поэтому установим минимум на 0, а максимум на 60 и получим данные за минуту. Также нам необходимо, чтобы график прокручивался и мы могли просматривать новые данные, записанные в каждую последующую минуту:

```
vp.graph(xmin=0, xmax=60, scroll=True)  
temp_graph = vp.gcurve()
```

7. Теперь, когда у нас есть метки времени, запишем время начала, прежде чем войти в цикл:

```
start = time.time()
```

8. Основной цикл имеет тип `while true`. Чтобы сообщить VPython, какие данные мы визуализируем, используем `vp.rate` и устанавливаем частоту кадров / частоту обновления для нашей системы:

```
while True:  
    vp.rate(100)
```

9. Теперь можем зафиксировать температуру и записать ее в журнал:

```
    temperature = imu.read_temperature()  
    logging.info("Temperature: {}".format(temperature))
```

10. Для создания графика нам нужно вычислить промежуток времени для оси X. Из текущего времени вычтем время начала:

```
    elapsed = time.time() - start
```

11. Наконец, отображаем данные на графике, где `x` – это временной промежуток, а `y` – температура:

```
    temp_graph.plot(elapsed, temperature)
```

Код для построения графика температуры готов. Давайте запустим его на Raspberry Pi.

## Запуск построителя графика температуры

Мы скопировали файлы на Raspberry Pi, но для запуска построителя потребуется выполнить еще несколько шагов. У нашего робота нет собственного дисплея, поэтому необходимо просматривать результаты VPython удаленно. Сначала нам нужно сообщить это VPython, а затем подключиться к сетевому порту для просмотра данных. Затем мы сможем просматривать результаты через веб-браузер. Рассмотрим, как это сделать.

1. В сеансе SSH на Raspberry Pi введите следующее:

```
$ VPYTHON_PORT=9020 VPYTHON_NOBROWSER=true python3 plot_  
temperature.py
```

Порт можно выбирать произвольно, но число должно быть больше 1000. Мы выбрали порт 9020. Позже в этой книге мы будем использовать дру-



гие веб-сервисы на разных портах, и это число не входит в их диапазоны. При запуске он должен зарегистрировать несколько сообщений, и это будет означать, что порт готов:

```
INFO:vpypython.no_notebook:Creating server
http://localhost:9020
INFO:vpypython.no_notebook:Server created
INFO:vpypython.no_notebook:Starting serve forever loop
INFO:vpypython.no_notebook:Started
```

Обратите внимание, что здесь показан адрес локального хоста. Мы планируем использовать его удаленно.

2. В браузере (Chrome, Firefox или Safari) укажите Raspberry Pi с номером порта. В моем случае, исходя из имени хоста робота, это будет `http://myrobot.local:9020`.
3. Далее нужно будет немного подождать, поскольку настройка VPython занимает некоторое время. В результате вы увидите или график, или сообщение о какой-либо ошибке/проблеме.

После запуска вы увидите график, отображающий данные датчика температуры. Здесь появляется возможность немного поэкспериментировать, например можно аккуратно приложить палец к датчику (большой черный квадрат на РИМ448) и посмотреть, как меняется график. Также можно использовать другие нагретые предметы, например фен. Будьте осторожны, не допустите попадания на робота влаги и не касайтесь контактов металлическими частями.

На рис. 12.12 показан график, отражающий зависимость температуры в градусах (ось Y) от времени в секундах (ось X). Толстая черная линия показывает текущие считываемые данные температуры. Показания сильно колеблются – система зашумлена.

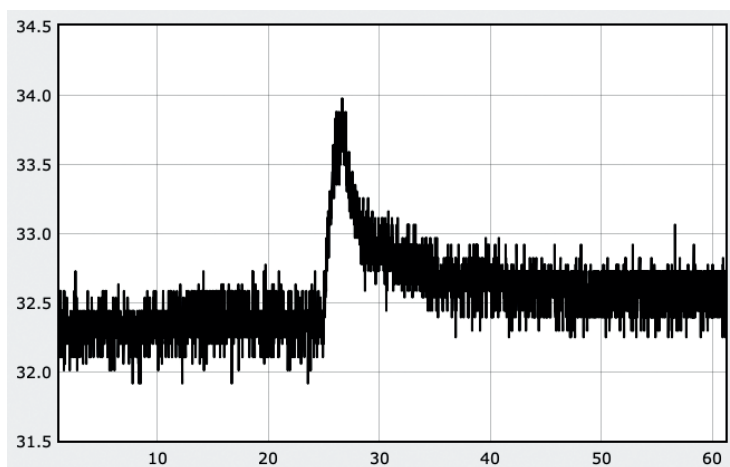


Рис. 12.12. График температуры

Я приложил палец к датчику примерно на 25-й секунде. Как видно на рис. 12.12, за несколько секунд внешняя температура поднялась от 31 °C до примерно 34 °C. Если бы я держал палец дольше, температура бы продолжила увели-

чиваться. В помещении находился вентилятор, поэтому падение температуры было довольно резким. В зависимости от условий она может падать гораздо медленнее. Температура записывается в консоль в виде кода:

```
INFO:root:Temperature 32.43858387995327
INFO:root:Temperature 32.726120945278105
INFO:root:Temperature 32.726120945278105
INFO:root:Temperature 32.39066103573247
INFO:root:Temperature 32.39066103573247
INFO:root:Temperature 32.63027525683649
```

Значения в десятичных разрядах сильно зашумлены, поэтому их можно игнорировать. Когда вы закроете эту вкладку браузера, код перестанет отображаться в виде графика.

#### Важное примечание

При тестировании температуры соблюдайте следующие правила: не кладите на датчик металлические предметы – это может привести к короткому замыканию контактов и повреждению робота. Не допускайте попадания воды. На холодных предметах может образовываться конденсат. При попадании он закоротит контакты и повредит датчик, а возможно, и Raspberry Pi.

## Устранение неполадок

Мы добавили в код робота два новых компонента, поэтому что-то могло пойти не так. Обратите внимание на следующие моменты.

1. VPython может работать медленно, поэтому его запуск обычно занимает много времени. Попробуйте обновить вкладку браузера через 30 с.
2. В VPython сообщение об ошибке может приходиться с задержкой. При тестировании нового кода запаситесь терпением.
3. При возникновении ошибок ввода/вывода или ошибок связи проверьте подключение IMU. Обратитесь к предыдущему разделу «Устранение неполадок» в этой главе. Ошибки ввода/вывода могли возникнуть из-за того, что вы задели провод, когда прикладывали палец к датчику. Также они могли возникнуть, если вы прикладывали к датчику металлический предмет. **НЕ КЛАДИТЕ МЕТАЛЛИЧЕСКИЕ ПРЕДМЕТЫ НА ДАТЧИК!**
4. При возникновении ошибок импорта проверьте правильность введенного кода и еще раз просмотрите рекомендации в предыдущем разделе «Устранение неполадок» в этой главе.
5. Данные температуры могут считываться с задержкой, поскольку изоляция IMU имеет низкую теплопроводность, и для того, чтобы нагреться или остыть, ему требуется время (в результате это все равно произойдет). Плата также обладает собственной удельной теплоемкостью. Это означает, что плата тоже будет нагреваться или охлаждаться, что потребует больше времени для достижения определенной температуры.
6. На точность данных температуры может влиять несколько факторов. Например, тепло может выделять сам IMU – мы уже упоминали удельную

теплоемкость. Сделать результат более точным можно, применив значение калибровки смещения, но не ожидайте, что результат будет идеально соответствовать термометру с точностью до доли градуса. Разумеется, датчик должен определять температуру пальца или ладони ближе к  $37^{\circ}$ , но, как показывает практика, немного подождяв, можно добиться и  $36^{\circ}$ .

Сейчас все работает, но запуск тестов можно упростить.

## Упрощение командной строки VPython

В этой главе мы будем часто использовать VPython, поэтому вводить множество настроек для запуска каждого файла Python будет неудобно. Упростить процесс можно с помощью псевдонима (ярлыка командной строки).

1. Настроим ярлык для текущей сессии. Команда `alias` создает псевдоним, который мы можем использовать позже. Здесь – это `vpython`. Он содержит настройки и команду `python3`:

```
pi@myrobot:~ $ alias vpython="VPYTHON_PORT=9020 VPYTHON_NOBROWSER=true python3"
```

2. Чтобы мы могли использовать псевдоним снова, поместим его в файл `.bashrc` текущего пользователя – файл, который Raspbian автоматически запускает при входе по `ssh`:

```
pi@myrobot:~ $ echo 'alias vpython="VPYTHON_PORT=9020 VPYTHON_NOBROWSER=true python3"' >> ~/.bashrc
```

Обертка в `echo` позволит выводить текст вместо запуска команды. Знак `>>` добавляет все в файл – в данном случае в `.bashrc`. Знак `~` указывает на домашний каталог текущего пользователя.

3. Вы можете повторно запустить демонстрацию температуры с помощью команды `vpython plot_temperature.py`.

В этом разделе вы научились получать данные от устройства IMU и увидели, как оно реагирует на изменения температуры. Вы убедились, что IMU отвечает. Затем записали данные в журнал и изобразили их в виде графика, а в процессе познакомились с системой VPython – мощным инструментом для отображения графики. Далее в этой главе мы будем использовать IMU и VPython для других задач. В следующем разделе перейдем к считыванию данных гироскопа, показывающих параметры поворота робота.

## СЧИТЫВАНИЕ ДАННЫХ ГИРОСКОПА С ПОМОЩЬЮ PУТНОН

В этом разделе мы будем использовать гироскоп в IMU. С его помощью мы определим направление «взгляда» робота в трех измерениях.

Но прежде разберемся, как он работает.

### Как работает гироскоп

Гироскоп измеряет скорость изменения угла вращения, например в градусах в секунду. При каждом измерении он определяет скорость вращения вокруг каждой оси.

Как показано на рис. 12.13, традиционно гироскоп представляет собой механическую систему. У него есть карданный подвес – несколько концентрических колец, соединенных шарнирами и способных вращаться вокруг осей X, Y и Z. В середине находится вращающаяся масса – ротор. Когда ротор вращается и изменяется положение основания (показано в виде подставки снизу на рис. 12.13), вращающаяся масса не меняет положения в пространстве, при этом карданный подвес позволяет ей свободно вращаться.

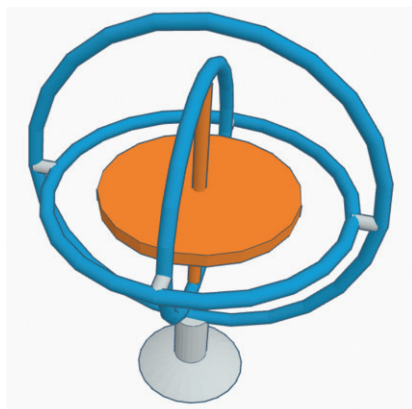


Рис. 12.13. Устройство механического гироскопа

МЭМС-гироскопы способны перемещать крошечную массу вперед и назад очень быстро (т. е. заставлять ее колебаться). При изменении ориентации гироскопа масса будет стремиться двигаться в прежнем направлении. Ее движение изменит электрическое поле, что обнаружится датчиком. В исходной ориентации это движение будет считаться силой Кориолиса.

Прежде чем перейти к созданию кода для работы с гироскопом, необходимо рассмотреть системы координат в работе и в VPython.

### Системы координат и вращения

В этой главе мы будем использовать системы координат и вращения. Обратимся к рис. 12.14.

На рис. 12.14 показано две системы, которыми мы будем пользоваться. Рассмотрим их в отдельности.

1. На этой части рисунка показана **система координат осей корпуса** робота – условный трехмерный эскиз робота с тремя осями, обозначенными стрелками. Ось X указывает в сторону передней части робота. Вращение вокруг этой оси называется **крен**. Ось Y указывает в левую сторону робота (если вы смотрите на робота, то для вас это будет право). Вращение вокруг этой оси называется **тангаж**. Вверх направлена ось Z. Вращение вокруг нее называется **рыскание**.

Важным параметром является направление вращения. Для его определения существует эмпирический метод: поднимите большой палец правой руки вверх; если большой палец направлен вдоль оси, то направление вращения будет совпадать с направлением остальных пальцев.

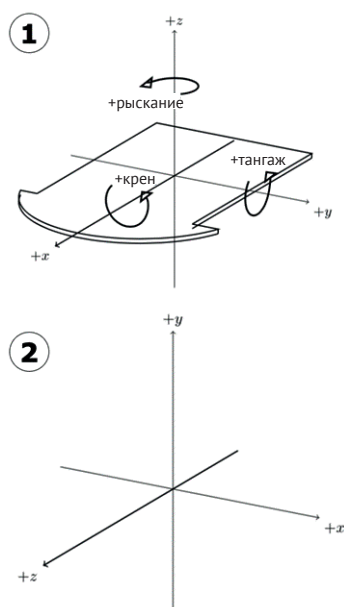


Рис. 12.14. Система координат осей корпуса робота

2. На этой части рисунка показана **международная система координат** VPython. В ней мы представляем трехмерные изображения в VPython. Она представляет собой вращение системы тела робота.

Мы видим, что ось Y направлена вверх, ось X – вправо, а ось Z указывает на переднюю часть робота.

Мы будем представлять координаты в трехмерном пространстве как компоненты X, Y и Z – также их называют **вектором**.

Применяя измерения к объектам в системе VPython, мы ориентируемся на систему координат робота. Когда мы говорим об одной системе координат относительно другой, это называется **позицией**. Это позиция робота относительно системы координат VPython. Представим в виде кода.

1. Создайте файл с именем `robot_pose.py`.
2. Мы работаем с VPython, поэтому нужно импортировать его:

```
import vpython as vp
```

3. Затем можем добавить функцию для настройки отображения объекта. Назовем ее `robot_view`:

```
def robot_view():
```

4. В этой функции нам нужно установить два свойства, которые VPython использует для управления ориентацией камеры.

На рис. 12.15 показана камера, смотрящая на робота в координатном пространстве. Камера будет смотреть в заданном направлении `forward` относительно нуля. Для ограничения крена камеры понадобится направление `up`.

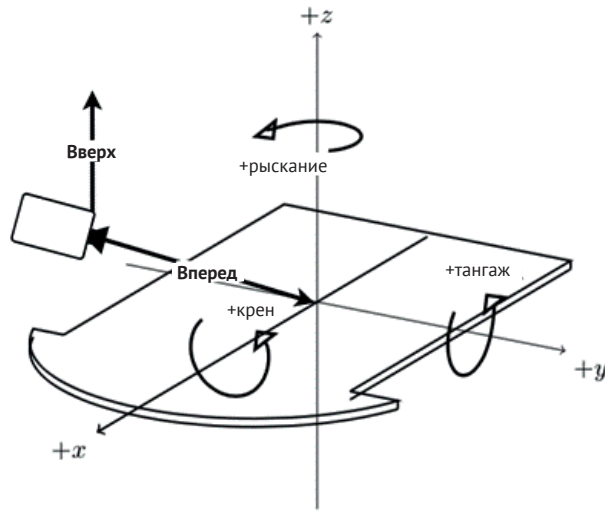


Рис. 12.15. Направление камеры

Камера должна смотреть на робота спереди, поэтому направление forward следует указывать в отрицательном направлении X. Также нам необходимо, чтобы камера была немного выше и сбоку:

```
vp.scene.forward = vp.vector(-3, -1, -1)
```

- Ось указывает нам на направление вдоль, а не *вверх*. Нам необходимо, чтобы для камеры направление вверх было таким же, как и для робота (ось Z); иначе векторы могут быть перевернуты или направлены в другую сторону:

```
vp.scene.up = vp.vector(0, 0, 1)
```

Мы будем использовать эту позицию в следующих разделах. Однако на данный момент отметим, что ось Z теперь вверх, и мы можем видеть, как робот вращается вокруг разных осей.

Далее настроим гироскоп для считывания его показаний.

## Добавление гироскопа в интерфейс

Для считывания данных гироскопа нам необходимо добавить его в интерфейс robot\_imu.py.

- Мы будем взаимодействовать с тремя группами данных IMU – x, y и z. Для их хранения импортируем векторный тип данных:

```
from icm20948 import ICM20948
from vpython import vector
```

- Векторные данные – это форма представления трехкомпонентных систем координат. Нам необходимо извлечь данные гироскопа из базовой библиотеки IMU и сохранить их в векторном формате:

```
def read_gyroscope(self):
    _, _, _, x, y, z = self._imu.read_accelerometer_
    gyro_data()
```

Из библиотеки ICM20948 от Pimoroni нельзя вызвать только данные гироскопа, но можно вызвать данные акселерометра и гироскопа вместе.

Библиотека ICM20948 возвращает данные в виде списка из шести элементов. В Python при распаковке возвращаемых значений символом нижнего подчеркивания () обозначаются элементы, которые следует игнорировать.

3. Теперь мы можем подставить значения гироскопа в вектор и вызвать их:

```
return vector(x, y, z)
```

На этом этапе можно получать данные гироскопа из библиотеки IMU. Далее мы займемся считыванием и построением графика.

## Построение графика данных гироскопа

Как упоминалось ранее, гироскоп измеряет скорость вращения в градусах в секунду по каждой оси.

Построим график его вывода.

1. Создайте файл с именем `plot_gyroscope.py`.
2. Начнем с импортов, настройки ведения журнала и IMU, как делали это раньше:

```
import vpython as vp
import logging
import time
from robot_imu import RobotImu

logging.basicConfig(level=logging.INFO)
imu = RobotImu()
```

3. Создадим три линии графика для трех осей гироскопа – вращение по оси X, вращение по оси Y и вращение по оси Z. Обратите внимание, что каждой линии графика мы задаем свой цвет:

```
vp.graph(xmin=0, xmax=60, scroll=True)
graph_x = vp.gcurve(color=vp.color.red)
graph_y = vp.gcurve(color=vp.color.green)
graph_z = vp.gcurve(color=vp.color.blue)
```

Три графика будут накладываться на одну ось.

4. Теперь нам нужно задать время начала, запустить цикл и измерить прошедшее время:

```
start = time.time()
while True:
    vp.rate(100)
    elapsed = time.time() - start
```

5. Мы можем считать данные IMU и нанести показания на графики:

```
gyro = imu.read_gyroscope()
graph_x.plot(elapsed, gyro.x)
```

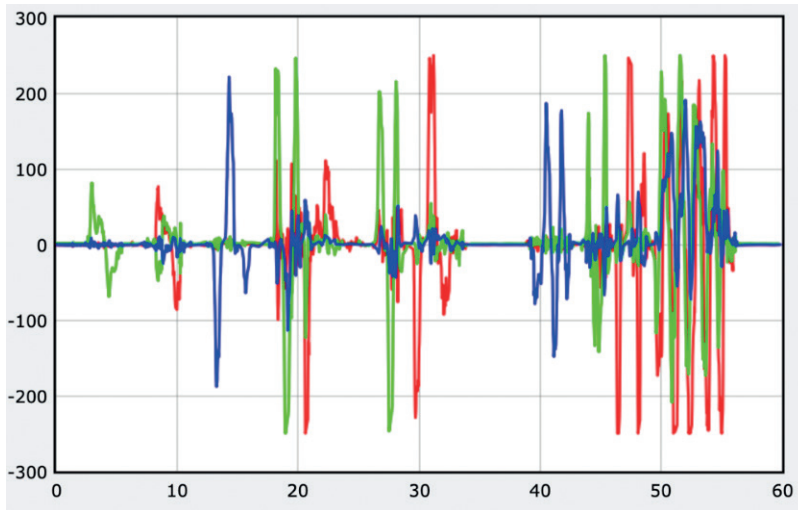


```
graph_y.plot(elapsed, gyro.y)
graph_z.plot(elapsed, gyro.z)
```

6. Загрузите файлы и запустите их с помощью `vpython plot_gyroscope.py`.
7. Подождите около минуты, а затем введите в браузере `myrobot.local:9020` – это может занять до 1 мин.
8. Начните изменять положение робота – поднимите его и попробуйте наклонить по каждой из трех осей. В результате получится график, как на рис. 12.16.

На графике есть три линии. По оси Y показана скорость движения в градусах в секунду, а по оси X время в секундах с момента запуска программы. Линии показаны красным, зеленым и синим цветом.

Когда вы меняете положение робота, график резко возрастает, а затем снова возвращается к нулю. Попробуйте наклонить переднюю часть робота (нос) – график возрастет по оси Y. Зеленая линия начнет двигаться вверх (на рис. 12.16 это происходит примерно через 3 с). Если задержать робота в таком положении, линия сгладится. Когда вы вернете его в исходное горизонтальное положение, график начнет резко убывать. Если поднять правую часть робота (по оси X) – красная линия на графике начнет возрастать. Когда вы вернете его в исходное положение – на графике появится отрицательный пик. Теперь попробуйте повернуть робота влево, и на графике начнет возрастать синяя линия. Если повернуть его вправо, появится отрицательный синий пик. Попробуйте менять положение робота по разным осям, чтобы почувствовать, как работают эти измерения.



**Рис. 12.16.** График данных гироскопа в VPython

Если не вращать робота постоянно, становится ясно, что поддерживать вращающую силу достаточно сложно. Так мы видим, что данные гироскопа представляют собой скорость поворота, а не оценку направления. Было бы полезнее аппроксимировать курс робота. По мере углубления в эту тему мы узнаем, как использовать для этого данные гироскопа.

В этом разделе мы познакомились с гироскопом и узнали, как он измеряет скорость вращения, визуализировав данные на графике. Далее перейдем к акселерометру и посмотрим, какие силы действуют на нашего робота.

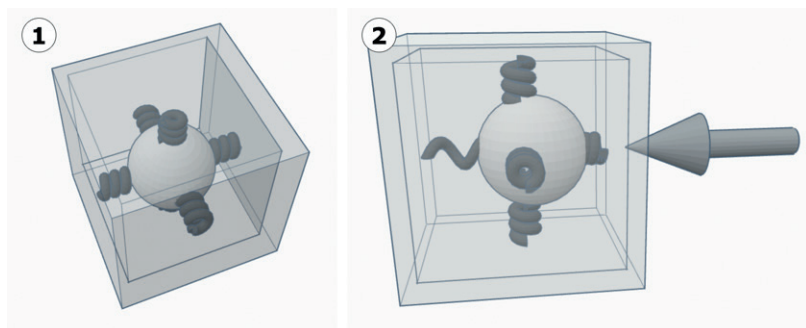
## СЧИТЫВАНИЕ ДАННЫХ АКСЕЛЕРОМЕТРА С ПОМОЩЬЮ PYTHON

В этом разделе мы узнаем, как использовать акселерометр для измерения сил, действующих на робота, и для определения того, где у робота находится низ. Сначала мы узнаем, как работает акселерометр, а затем разработаем код.

### Как работает акселерометр

Акселерометр измеряет *ускорение*, т. е. скорость изменения скорости по двум параметрам – величине и направлению. Он выдает три значения – по одному для осей X, Y и Z.

На рис. 12.17 показано обобщенное устройство акселерометра. Рассмотрим его детально.



**Рис. 12.17.** Обобщенное устройство акселерометра – масса с пружинами

1. На этой части рисунка показан шарик (масса), подвешенный в коробке на шести пружинах. Когда на коробку не действуют силы, шарик остается в центре.
2. На этой части рисунка показано, как ведет себя система, когда на нее действует сила (показана большой стрелкой). По инерции масса сдвигается вправо, сжимая правую пружину и растягивая левую.

Путем измерения положения массы мы можем узнать направление и величину ускорения. МЭМС-акселерометр работает по тому же принципу и имеет крошечную силиконовую массу и пружины.

На Земле масса притягивается к поверхности под действием силы тяжести (ускорения свободного падения). Система ведет себя так, будто коробку удерживает сила, поэтому акселерометр регистрирует силу, направленную вверх. Используя это измерение, мы можем определить, где находится низ, и почувствовать наклон робота.

## Добавление акселерометра в интерфейс

Начнем с добавления измерений акселерометра в библиотеку RobotImu.

1. Откройте файл `robot_imu.py`.
2. Для считывания данных добавьте следующий фрагмент кода:

```
def read_accelerometer(self):
    accel_x, accel_y, accel_z, _, _, _ = self._imu.
read_accelerometer_gyro_data()
    return vector(accel_x, accel_y, accel_z)
```

Здесь мы используем тот же вызов библиотеки, что и при работе с гироскопом, но теперь игнорируем последние три элемента данных.

Теперь, когда мы можем считывать данные акселерометра, их нужно визуализировать.

## Отображение данных акселерометра в виде вектора

Ускорение – это вектор в трехмерном пространстве. Оно имеет направление и величину. Ускорение можно представить в виде стрелки в трехмерном пространстве. Чтобы уточнить положение этого вектора, мы можем построить индикатор для осей X, Y и Z.

1. Создайте файл с именем `accelerometer_vector.py`. Начнем с импорта представления робота, настроек ведения журнала и инициализации IMU:

```
import vpython as vp
import logging
from robot_imu import RobotImu
from robot_pose import robot_view
logging.basicConfig(level=logging.INFO)
imu = RobotImu()
```

2. Укажем на то, под каким углом мы смотрим на робота:

```
robot_view()
```

3. Теперь нам нужно определить четыре стрелки. Стрелки в VPython указывают вдоль оси, и им можно задать цвет и длину:

```
accel_arrow = vp.arrow(axis=vp.vector(0, 0, 0))
x_arrow = vp.arrow(axis=vp.vector(1, 0, 0),
    color=vp.color.red)
y_arrow = vp.arrow(axis=vp.vector(0, 1, 0),
    color=vp.color.green)
z_arrow = vp.arrow(axis=vp.vector(0, 0, 1),
    color=vp.color.blue)
```

4. Запускаем главный цикл:

```
while True:
    vp.rate(100)
```

5. Считываем данные акселерометра и записываем данные в журнал:

```
accel = imu.read_accelerometer()
print(f"Accelerometer: {accel}")
```

6. Поскольку выбросы данных могут сбить нашу шкалу, мы нормализуем вектор так, чтобы его длина равнялась 1. Применяем нормализацию к оси стрелки:

```
accel_arrow.axis = accel.norm()
```

7. Загрузите код в Raspberry Pi и запустите с помощью `python accelerometer_vector.py`. Запустите ваш браузер, чтобы увидеть вывод, как на рис. 12.18.

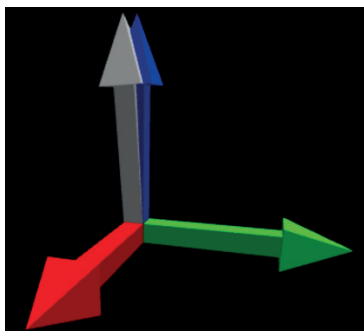


Рис. 12.18. Вектор акселерометра

На рис. 12.18 показаны три цветные стрелки: красная для оси X (указывающая на смотрящего), зеленая для оси Y (указывающая влево) и синяя для оси Z (указывающая вверх). Серая стрелка показывает вектор акселерометра. Она указывает вверх, что говорит нам о том, что вектор акселерометра направлен против вектора гравитации.

8. Теперь, если наклонить робота, стрелка будет наклоняться и показывать, где относительно робота находится верх. Попробуйте наклонить робота в разные стороны и понаблюдать как ведут себя стрелки.

Мы знаем, где находится верх относительно нашего робота, – это было очень увлекательно. Чтобы использовать эти данные для вращения, необходимо превратить вектор в углы крена и тангажа, чем мы займемся по мере углубления в тему.

В этом разделе вы узнали, как считывать данные с акселерометра IMU и отображать их в виде вектора. Теперь перейдем к следующему компоненту IMU – магнитометру – и научимся считывать магнитные поля, действующие на нашу систему.

## СЧИТЫВАНИЕ ДАННЫХ МАГНИТОМЕТРА

Магнитометр считывает напряженность магнитного поля и генерирует вектор в трехмерном пространстве. Код, который мы разработаем, будет определять магнитный север так же, как это делает компас. В этом разделе мы познакомимся с устройством поближе, узнаем, как получить от него данные, и рассмотрим, какие векторы оно генерирует.

Нам потребуется пространство с несколькими магнитами. Давайте разберемся, как работает магнитометр.

## Как работает магнитометр

Компас указывает на направление магнитных полюсов Земли с помощью магнитной стрелки или диска. На рис. 12.19 показано, как выглядит это устройство.



Рис. 12.19. Компас

У компаса, показанного на рис. 12.19, есть вращающийся намагниченный диск, уравновешенный на центральном штифте. Такие компасы называются *микрокомпасами* (*button compass*). Их диаметр составляет около 25 мм.

В нашем IMU есть компонент под названием **магнитометр**. Он измеряет магнитное поле при помощи электронного датчика и может использоваться как компас.

Большинство магнитометров пропускает электричество через материал, который генерирует ток при воздействии на него магнитного поля, как показано на рис. 12.20.

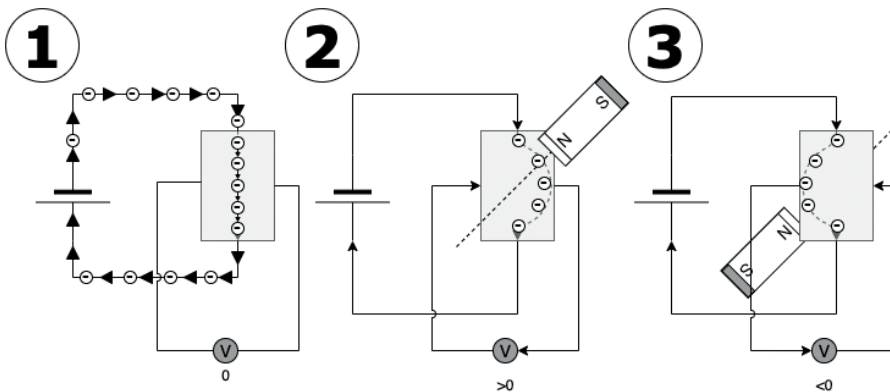


Рис. 12.20. Условное изображение датчика на эффекте Холла

На рис. 12.20 показано, как это работает.

1. На этой части рисунка показано, что в цепи электрический ток проходит от батареи (слева) через проводящую пластину (серый прямоуголь-

ник). Стрелки показывают направление электронов (носителей отрицательного заряда), которые движутся по цепи от верхней части пластины к нижней. Небольшой кружок с буквой V внутри – это датчик напряжения (электрического потока), который подключен к пластине с двух сторон. Поток не идет к датчику, поэтому он показывает 0.

2. Здесь над пластиной находится магнит, отклоняющий электроны в сторону. Они дают одной стороне пластины отрицательный заряд, а другой стороне – положительный. Такая разница заставляет напряжение проходить через датчик так, как показано стрелками. Сейчас показание датчика больше 0.
3. Если поместить магнит с другой стороны датчика, магнитное поле изменится. Электроны отклоняются в другую сторону, вызывая обратное напряжение. Стрелки, ведущие к магнитометру, принимают противоположное направление, а показание датчика составляет меньше 0.

Описанное явление называется эффектом Холла. Измерив три пластины, вы можете измерить магнитные поля в трех измерениях. Магнитометры чувствительны к магнитным полям и металлическим предметам.

Еще одна особенность заключается в том, что в некоторых IMU оси магнитометра отличаются от осей других устройств.

Слева на рис. 12.20 мы видим, что оси, которые мы рассматривали ранее, – это оси гироскопа и акселерометра. Справа показаны оси магнитометра. Здесь мы видим, что ось Z направлена вниз, а ось Y теперь направлена назад. Это выглядит так, будто мы повернулись на  $180^\circ$  вокруг оси X.

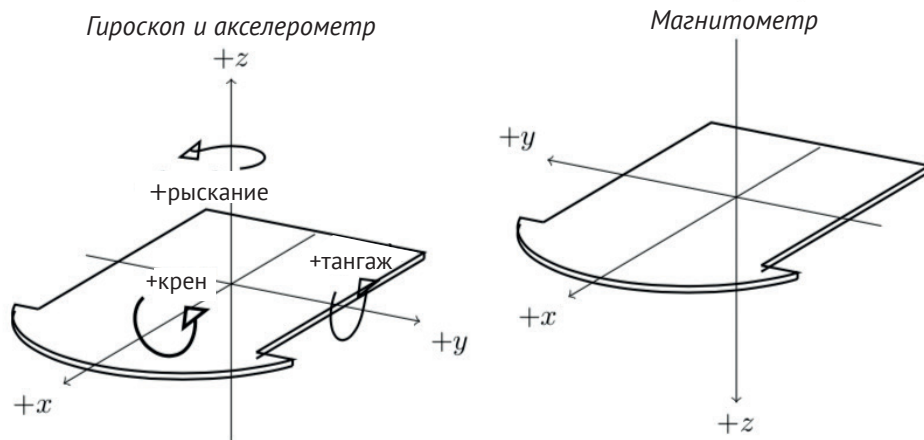


Рис. 12.21. Оси магнитометра

Теперь перейдем к созданию кода, с помощью которого мы сможем считывать эти данные.

## Добавление интерфейса магнитометра

Мы обернем данные магнитометра в функцию так же, как и в предыдущих случаях. Для этого добавим его в библиотеку интерфейса.

1. Откройте файл `robot_imu.py`.
2. В классе `RobotIMU` после метода `read_gyroscope` добавьте новый метод чтения:

```
def read_magnetometer(self):
```

3. В отличие от данных акселерометра и гироскопа данный метод считывает данные из отдельного вызова в базовую библиотеку устройства. Обращаем его и возвращаем вектор. Для резкого поворота на  $180^\circ$  выполняем операцию смены знака для осей Y и Z:

```
mag_x, mag_y, mag_z = self._imu.read_magnetometer_data()
return vector(mag_x, -mag_y, -mag_z)
```

Теперь, когда интерфейс готов к использованию, давайте получим показания от магнитометра.

## Отображение данных магнитометра

Один из способов визуализации – преобразование выходных данных магнитометра в вектор.

1. Создайте файл с именем `magnetometer_vector.py`.
2. Добавьте уже знакомый импорт и настройку:

```
import vpython as vp
import logging
from robot_imu import RobotImu
from robot_pose import robot_view
logging.basicConfig(level=logging.INFO)
imu = RobotImu()
robot_view()
```

3. Создадим стрелку для показаний магнитометра, а также опорные оси X, Y и Z:

```
mag_arrow = vp.arrow(pos=vp.vector(0, 0, 0))
x_arrow = vp.arrow(axis=vp.vector(1, 0, 0), color=vp.color.red)
y_arrow = vp.arrow(axis=vp.vector(0, 1, 0), color=vp.color.green)
z_arrow = vp.arrow(axis=vp.vector(0, 0, 1), color=vp.color.blue)
```

4. Затем запустим главный цикл:

```
while True:
    vp.rate(100)
```

5. Теперь мы можем считать данные магнитометра:

```
mag = imu.read_magnetometer()
```

6. Наконец, давайте настроим ось стрелки, которая будет соответствовать этому вектору. Для нормализации вектора мы можем использовать метод `.norm()`. Также нам нужно вывести данные на печать:



```
mag_arrow.axis = mag.norm()
print(f"Magnetometer: {mag}")
```

7. Загрузите код в память робота и запустите его с обычными настройками VPython. Результат показан на рис. 12.22.

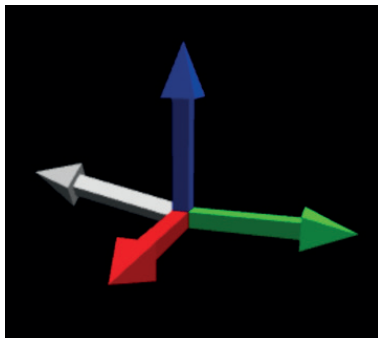


Рис. 12.22. Результат считывания данных магнитометра

На рис. 12.22 мы видим холст с красной стрелкой для оси X, указывающей вперед, синей стрелкой для оси Z, указывающей вверх, и зеленой стрелкой для оси Y, указывающей вправо. Серая стрелка показывает вектор магнитометра (только XZ), указывающий в обратном направлении.

У вас серая стрелка может указывать в другом направлении. Это связано с тем, что она, вероятнее всего, будет указывать туда, где находятся штыревые разъемы контактов на вашем IMU. Почему так происходит?

Штыревые разъемы обычно изготавливаются из магнитного металла. Вы можете проверить это сами, взяв магнит и посмотрев, притягивается ли он к разъемам (используйте запасные компоненты или делайте это при выключенном питании). Также можете посмотреть, как это влияет на направление стрелки. Вы можете взять металлический объект, например отвертку, и помахивать им вокруг магнитометра. Тогда он зафиксирует перемещение магнитного объекта с разных сторон.

Позже нам нужно будет компенсировать влияние близлежащих металлических объектов, поскольку они могут создавать смещение, достаточно большое, чтобы полностью перекрыть относительно слабое магнитное поле Земли.

## Выводы

В этой главе вы узнали, как считывать данные с четырех датчиков IMU, а также как визуализировать их или представить в виде графиков. Вы получили опыт пайки – критически важный навык, когда дело доходит до создания роботов, а также узнали больше о системах координат роботов.

Позже в этой книге мы займемся объединением датчиков IMU, которое позволит узнать приблизительную ориентацию робота.

В следующей главе мы рассмотрим компьютерное зрение, т. е. узнаем, как получить данные камеры и заставить робота реагировать на то, что он видит.

## ЗАДАНИЕ

- На графике и в выводе датчика температуры вы заметите много шума. В Python существует функция `round`, принимающая число и количество десятичных разрядов, которые необходимо сохранить (по умолчанию 0). Округлите значение температуры с помощью этой функции.
- Попробуйте нанести значения акселерометра на график X, Y и Z, как мы делали это для гироскопа. Понаблюдайте за изменениями на графике при перемещении робота. График плавный, или есть шум?
- Можно ли отображать значения гироскопа в виде вектора?
- Подумайте, какие еще интересные и нужные датчики мы могли бы добавить нашему роботу.

## ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

Дополнительную информацию по темам из этой главы вы можете найти в следующих источниках:

- обширной справке по VPython: <https://www.glowscript.org/docs/VPython-Docs/index.html>;
- экспериментах Пола МакУортера (Paul McWarter) с Arduino и IMU: <https://toptechboy.com/arduino-based-9-axis-inertial-measurement-unit-imu-based-on-bno055-sensor/>;
- руководствах по использованию IMU с библиотеками от Adafruit: <https://learn.adafruit.com/adafruit-sensorlab-magnetometer-calibration>;
- видеоролике от Google на YouTube («*Sensor Fusion on Android Devices: A Revolution in Motion Processing*»): <https://www.youtube.com/watch?v=C7JQ7Rpwn2k>.



## Часть 3

# Слух и зрение – «интеллектуальные» датчики для робота

В этой части вы сделаете робота более «интеллектуальным» и интерактивным с помощью OpenCV и Mucroft, а также разработаете веб-приложение для управления роботом через смартфон.

Часть включает в себя следующие главы:

- главу 13. Система технического зрения робота – камера Pi и OpenCV;
- главу 14. Код на Python для отслеживания линий с помощью камеры;
- главу 15. Голосовая связь с роботом посредством Mucroft;
- главу 16. Погружаемся глубже в работу IMU;
- главу 17. Разработка кода на Python для управления роботом с помощью смартфона.



# Глава 13

## Система технического зрения робота – камера Pi и OpenCV

Мы можем значительно расширить возможности робота, предоставив ему возможность видеть мир. Сегодня компьютерное зрение все еще активно исследуется, но некоторые базовые функции мы уже можем реализовать в коде. Для этого нам понадобятся камера, подключенная к Pi, и немного усердия.

В этой главе мы научим нашего робота отслеживать объекты и лица с помощью камеры и механизма поворота и наклона. Мы расширим применение ПИД-алгоритма, а также передадим видеопоток камеры на веб-страницу, что даст нам возможность увидеть то, что видит робот.

В этой главе рассмотрим следующие темы:

- настройку камеры Raspberry Pi;
- настройку программного обеспечения для задач компьютерного зрения;
- создание приложения для потоковой передачи данных камеры Raspberry Pi.
- запуск фоновых задач во время потоковой передачи;
- отслеживание цветных объектов с помощью кода на Python;
- отслеживание лиц с помощью кода на Python.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам потребуются:

- робот с механизмом поворота и наклона из главы 11;
- код, разработанный в главе 11. Его вы можете найти на GitHub по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter11>. Мы внесем в него некоторые изменения для реализации новых функций;
- камера Raspberry Pi;
- шлейф для подключения камеры к Pi длиной 300 мм (провод из комплекта поставки камеры слишком короткий). Проверьте, чтобы шлейф был не для Pi Zero (у этой модели другие разъемы);
- два болта M2 и гайка M2;
- небольшой квадратный кусок тонкого картона (подойдет картон от коробки из-под хлопьев);
- миниатюрная отвертка;
- карандаш;

- детский набор для боулинга с разноцветными кеглями (без рисунков);
- хорошо освещенное пространство, в котором будет перемещаться робот;
- доступ к интернету.

Полный код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter13>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/39xfDJ9>.

## НАСТРОЙКА КАМЕРЫ RASPBERRY PI

Прежде чем мы перейдем к компьютерному зрению, нужно подготовить камеру. Для этого установим программное и аппаратное обеспечение.

Когда мы установим камеру, блок-схема робота будет выглядеть так, как показано на рис. 13.1.

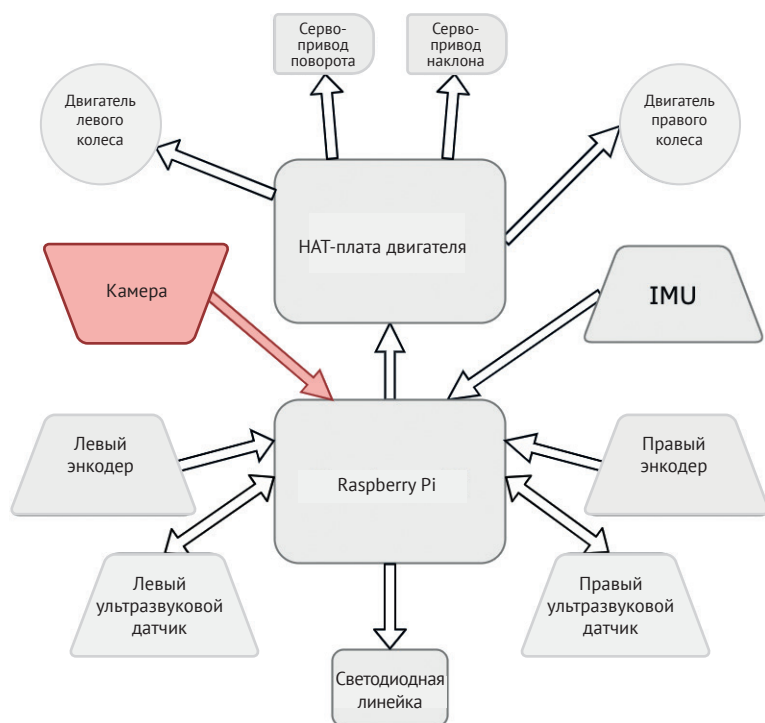


Рис. 13.1. Блок-схема робота с камерой

На рис. 13.1 показана блок-схема робота со всеми компонентами, которые мы добавили ранее, и камерой, которая подключается к Raspberry Pi. Новый компонент выделен красным цветом в левой части рисунка.

Сначала мы установим камеру на механизм поворота и наклона. Затем с помощью длинного провода подключим камеру к Pi. Приступим к подготовке и установке камеры.



## Установка камеры на механизм поворота и наклона

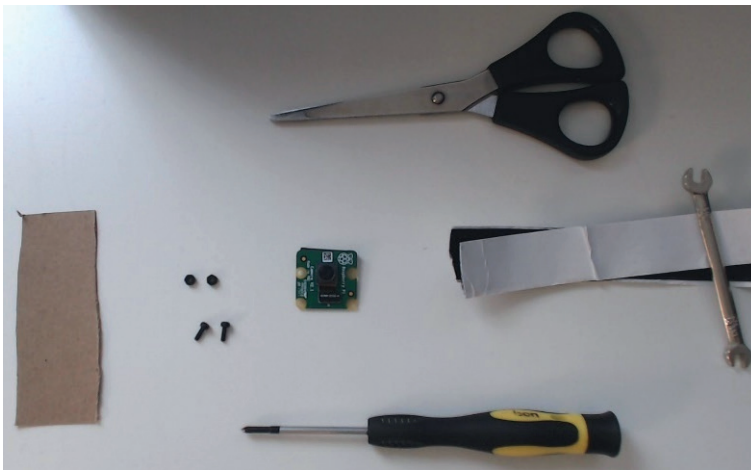
В главе 10 мы добавили роботу механизм поворота и наклона. Нам необходимо установить камеру на его переднюю панель. Для этого существуют специальные наборы и крепежи, однако приобрести их можно не везде. По возможности воспользуйтесь ими. Если такой возможности нет, попробуем несколько других способов.

Помимо технических навыков, создание роботов требует творческого и гибкого мышления. Прежде чем что-нибудь приобрести, я часто просматриваю имеющиеся у меня материалы в поисках возможных решений. Иногда с первого раза что-то может не получиться, и на этот случай нужен план Б. Мой план А состоял в том, чтобы прикрепить камеру на «липучку», но оказалось, что она плохо прилегает к задней части камеры. Тогда я перешел к плану Б: я проделал два отверстия в кусочке картона и прикрепил его к камере 2-мм винтами, а затем установил всю конструкцию на панель механизма с помощью «липучки». Другой вариант – просверлить дополнительные отверстия в механизме поворота и наклона, чтобы они совпадали с отверстиями для винтов на камере.

### Совет

Можно ли использовать клей? Да, приклеить камеру к механизму поворота и наклона можно (даже на суперклей). Это значительно облегчит процесс сборки. Однако в дальнейшем вам может понадобиться снять камеру (например, если запутается провод) или заменить ее (например, на другой датчик или камеру с характеристиками получше). Именно по этой причине я не рекомендую использовать клей. В своих проектах я всегда стараюсь искать модульные решения, допускающие замену компонентов.

На рис. 13.2 показаны необходимые компоненты.



**Рис. 13.2.** Компоненты, необходимые для установки модуля камеры

Нам понадобятся следующие инструменты и материалы: тонкий картон, 2-мм болты и винты, модуль камеры Pi, ножницы, небольшой гаечный ключ (или плоскогубцы), «липучка», миниатюрная отвертка и карандаш.

Во время работы старайтесь не касаться объектива камеры. Итак, начнем. На рис. 13.3 показаны первые шаги установки камеры.

Выполните следующие шаги.

1. Сначала отрезаем немного «липучки» и прикрепляем одну сторону к механизму поворота и наклона, как показано на рис. 13.3.
2. Размечаем на картоне и вырезаем небольшой квадрат размером чуть больше камеры.

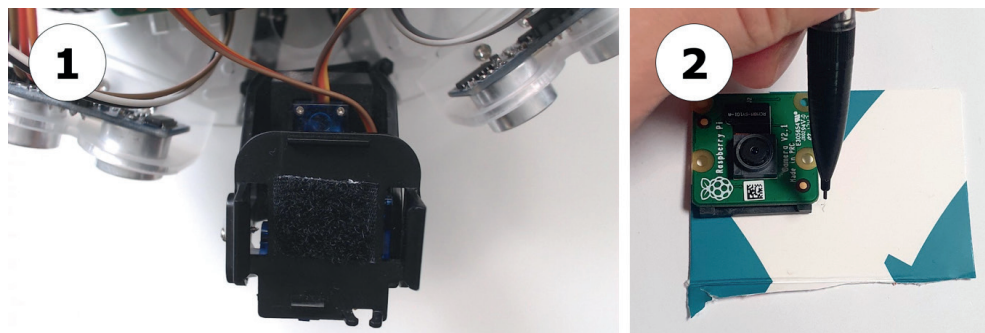


Рис. 13.3. Установка камеры, шаги 1-2

3. Ручкой или карандашом отмечаем на картоне места отверстий для винтов камеры, как показано на рис. 13.4. Затем берем остроконечный инструмент (например, острие крестовой отвертки или циркуль) и на твердой поверхности проделываем отверстия в отмеченных местах.

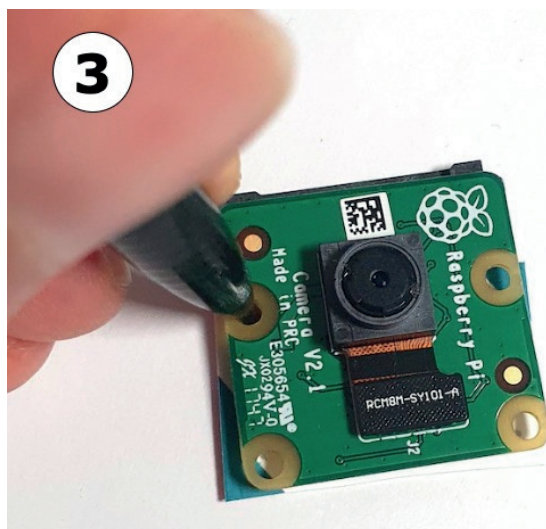
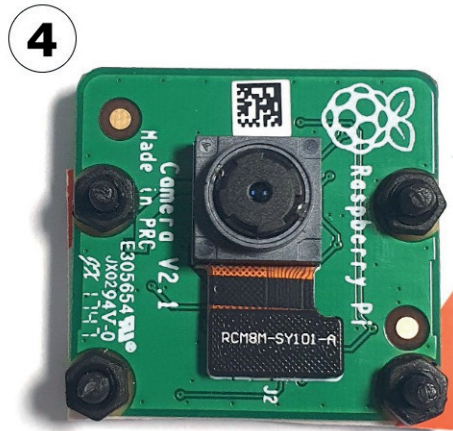


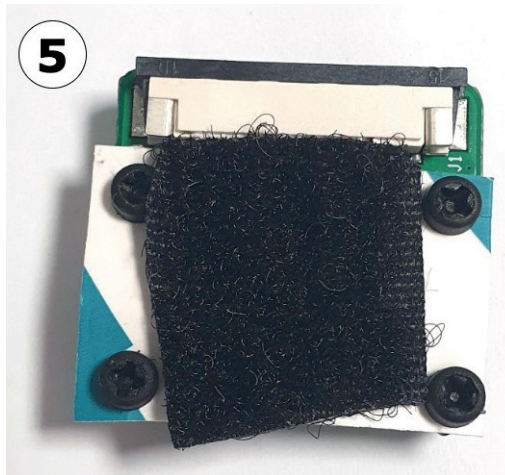
Рис. 13.4. Разметка положения винтов

4. Закрепляем камеру на картонной подложке болтами M2, как показано на рис. 13.5. Обратите внимание, что головки болтов должны находиться сзади, чтобы торчащая резьба не мешала использовать «липучку».



**Рис. 13.5.** Камера на картонной подложке

5. Возьмите другую сторону «липучки» и приклейте его к обратной стороне картонной подложки, как показано на рис. 13.6.



**Рис. 13.6.** Задняя часть конструкции (камера на картонной подложке) с «липучкой»

Если на объективе камеры есть защитная пленка – снимите ее.

Камера готова к установке на робота, но прежде необходимо подключить провод. В следующем разделе описано, как это сделать.

## Подключение камеры

Теперь, когда камера готова, нам понадобится специальный ленточный кабель (шлейф), чтобы подключить ее к Pi. Чтобы добраться до разъема Raspberry Pi и вставить в него шлейф, необходимо будет снять некоторые компоненты.

На рис. 13.7 показана последовательность подключения.

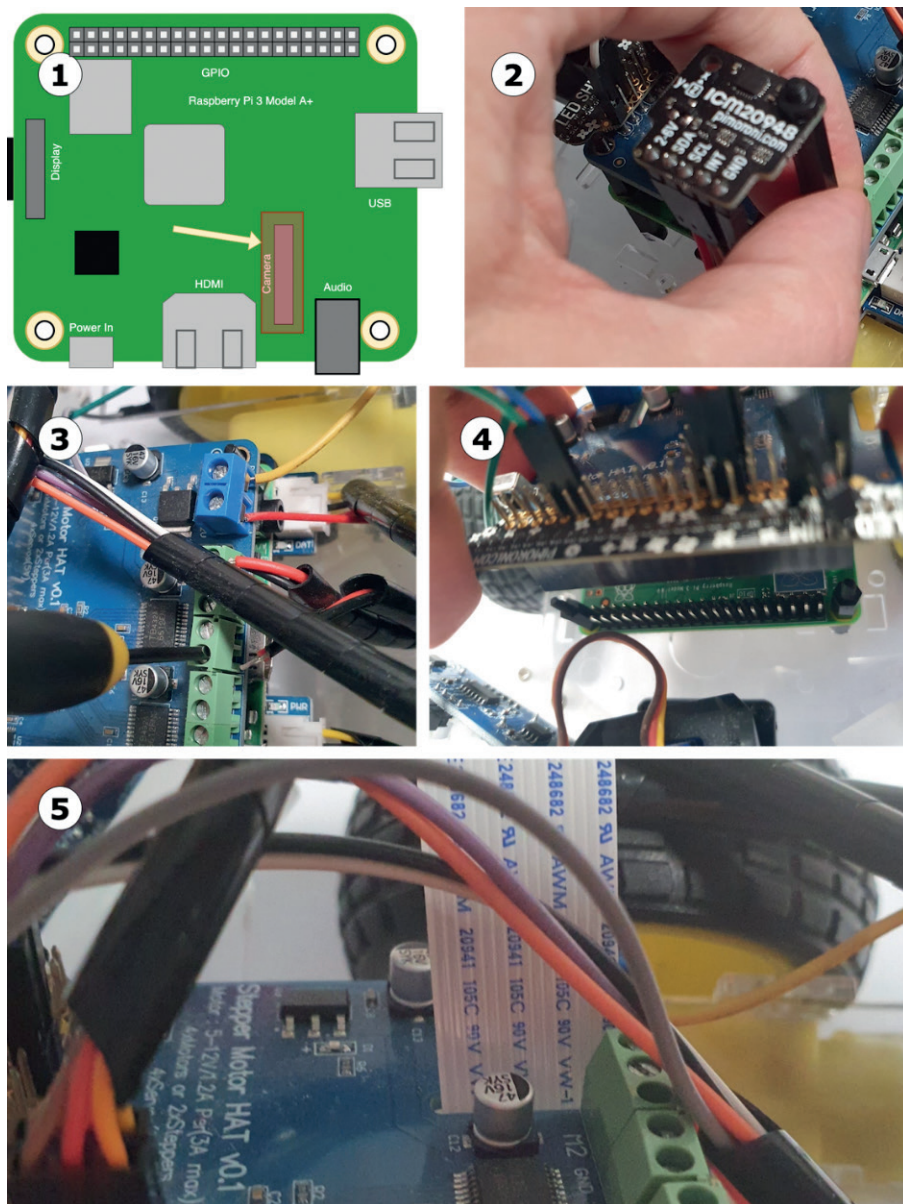


Рис. 13.7. Щелевое гнездо разъема камеры и макетная плата



Чтобы подготовить разъем, выполните следующие шаги, руководствуясь рис. 13.7.

1. В Raspberry Pi есть специальный разъем для подключения камеры. Мы будем подключать камеру к нему, но сейчас он находится под платой двигателя.
2. Чтобы добраться до разъема, необходимо убрать некоторые платы на Pi. Сначала уберем **инерциальный измерительный модуль (IMU)**, чтобы он не закрывал плату двигателя. Ослабьте гайку на IMU, затем вручную поверните нижнюю распорную стойку и снимите модуль. Отделять IMU от стойки для этого не нужно.
3. Отсоедините провода двигателя (запомните, как они подключены, или сделайте фото).
4. Теперь аккуратно снимите плату двигателя с Raspberry Pi.
5. Переходя к следующему шагу, помните, что, когда вы подключаете камеру к Pi, длинный шлейф должен проходить через плату двигателя.

Для подключения камеры посредством 300-мм шлейфа обратитесь к инструкции в официальном руководстве от Raspberry Pi («*Connect ribbon cable to camera* в *The Official Raspberry Pi Camera Guide*»). Его вы можете найти по адресу <https://magpi.raspberrypi.com/books/camera-guide>). Следуя инструкциям, вы должны правильно подключить шлейф к камере, а затем пропустить его через прорезь в плате двигателя и вставить в порт на Raspberry Pi.

Перед тем как установить плату двигателя обратно, дважды проверьте правильность подключений. В дальнейшем это сэкономит вам время.

На рис. 13.8 показано, как завершить сборку.

Выполните следующие шаги, опираясь на рис. 13.8.

1. Аккуратно установите плату двигателя, совместив ее контакты с разъемом GPIO Raspberry Pi, а отверстия платы со стойками.
2. Прикрутите IMU обратно.
3. Подключите провода двигателя (по памяти или опираясь на сделанное ранее фото).
4. Закрепите камеру на механизме поворота и наклона с помощью «липучки» так, чтобы провод был направлен вверх.

Вы подключили камеру к Raspberry Pi, и теперь она готова к использованию. Далее приступим к подготовке программного обеспечения для получения изображений с камеры.

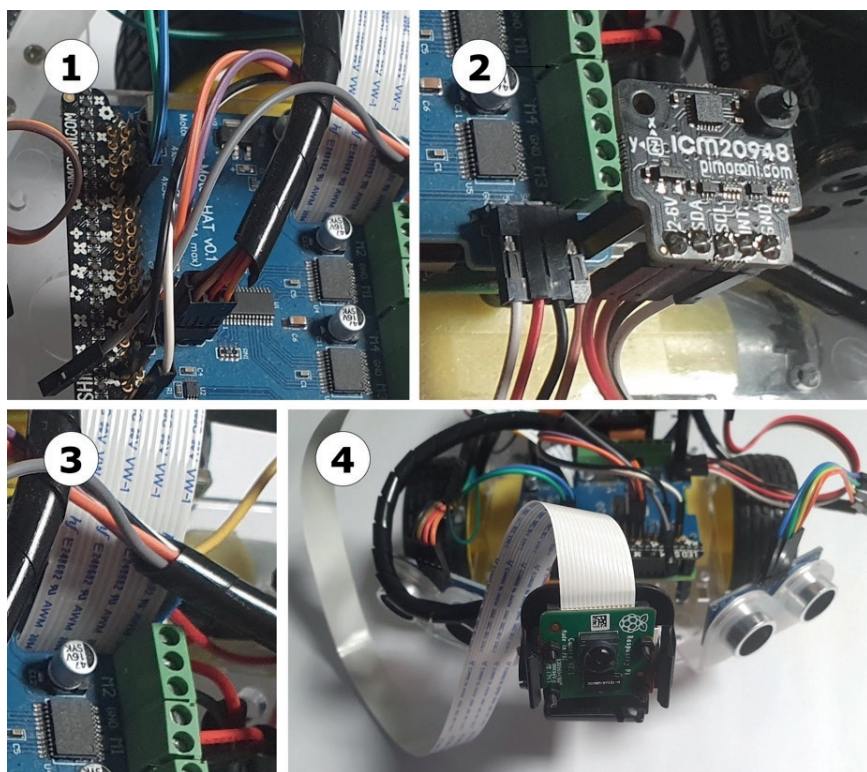


Рис. 13.8. Завершение установки камеры

## НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ЗАДАЧ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Прежде чем перейти к созданию кода, нам понадобится установить драйверы, инструменты и библиотеки для взаимодействия с камерой, а также программное обеспечение для компьютерного зрения.

В этом разделе мы активируем камеру в ОС Raspberry Pi и получим тестовое изображение. Затем добавим нужные библиотеки для обработки видеоданных.

Далее мы создадим наше первое приложение, которое позволит удостовериться, что все компоненты подключены правильно. Оно станет отправной точкой для разработки поведенческого сценария. Приступим к настройке программного обеспечения.

### Настройка программного обеспечения камеры Pi

Сейчас камера готова к использованию, и нам нужно включить ее.

1. Включите Pi от внешнего источника питания (от USB-адаптера). Двигатели должны быть выключены.
2. Войдите через SSH. В терминале введите следующее:

```
pi@myrobot:~ $ sudo raspi-config
```

3. Вы увидите инструмент `gaspi-config`. С помощью клавиш управления курсором выберите **Interfacing Options** (Параметры интерфейса) и нажмите **Ввод**.
4. Выберите **Pi Camera** (Камера Pi). Здесь `gaspi-config` спросит, хотите ли вы включить интерфейс камеры – нажмите **Yes** (Да) и **Ok**, а затем **Finish** (Завершить).
5. Для вступления изменений в силу нужно перезагрузить Pi:

```
pi@myrobot:~ $ sudo reboot
```

Чтобы получать изображения, нам понадобится пакет `picamera`. На момент написания этой книги копия `picamera` уже установлена в ОС Raspberry Pi по умолчанию.

Теперь, когда камера включена, попробуем получить первое изображение.

## Получение изображения с Raspberry Pi

Чтобы проверить правильность настройки, необходимо сделать снимок с помощью камеры. Если Pi не видит ее, проверьте, правильно ли вы подключили провод, установили ли вы `picamera` и включили ли камеру Raspberry Pi в `gaspi-config`.

1. Для получения изображения снова подключитесь к Raspberry Pi и введите следующее:

```
pi@myrobot:~ $ raspistill -o test.jpg
```

Команда `raspistill` делает статичный снимок, а параметр `-o` указывает ей сохранить это изображение в файле `test.jpg`. Выполнение команды может занять некоторое время. На длительность также влияют условия освещения.

2. Чтобы загрузить это изображение и просмотреть его на своем компьютере, используйте SFTP-клиент из главы 4. Вы увидите, что изображение перевернуто (потому что так установлена камера). Не беспокойтесь – далее мы исправим это программным способом.

Получив изображение, вы поймете, что камера работает. Теперь мы можем установить другие программы, необходимые для использования приложений обработки видеоданных камеры.

## Установка OpenCV и вспомогательных библиотек

Для выполнения наиболее сложных задач и отображения выходных данных камеры удобным способом нам потребуется несколько вспомогательных библиотек. **Open Computer Vision (OpenCV)** – это библиотека с набором инструментов для обработки изображений и извлечения информации. В коде все инструменты OpenCV можно использовать вместе, что позволяет создавать поведенческие сценарии и конвейеры обработки видеоданных.

Чтобы запустить код на Raspberry Pi, нам потребуется установить библиотеку OpenCV для Python.

1. OpenCV имеет несколько зависимостей, которые необходимы нам в первую очередь:



```
pi@myrobot:~ $ sudo apt install -y libavcodec58
libilmbase23 libgtk-3-0 libatk1.0-0 libpango-1.0-0
libavutil56 libavformat58 libjasper1 libopenexr23
libswscale5 libpangocairo-1.0-0 libtiff5 libcairo2
libwebp6 libgdk-pixbuf2.0-0 libcairo-gobject2 libhdf5-dev
pi@myrobot:~ $ sudo pip3 install "opencv_python_
headless<4.5" "opencv_contrib_python_headless<4.5"
```

2. ОС Raspberry Pi требует, чтобы вы каждый раз указывали библиотеку для работы OpenCV. Следующая строка определяет библиотеку каждый раз, когда вы входите в Pi. Сделаем это для текущей сессии:

```
pi@myrobot:~ $ echo export LD_PRELOAD=/usr/lib/arm-linux-
glibc/libatomic.so.1 >>.bashrc
pi@myrobot:~ $ source .bashrc
```

3. Flask – это библиотека для создания веб-серверов, которые мы будем использовать для потоковой передачи видеоданных в браузер:

```
pi@myrobot:~ $ sudo pip3 install flask
```

4. NumPy – это библиотека численного анализа для Python, позволяющая работать с большими массивами данных. Изображения хранятся на компьютере в виде больших блоков чисел, где каждая крошечная точка описывается тремя числовыми значениями цветов, аналогичными тем, которые мы отправляли на светодиоды в главе 9:

```
pi@myrobot:~ $ sudo apt install -y libgfortran5
libatlas3-base
pi@myrobot:~ $ sudo pip3 install numpy
```

5. Нам нужно установить расширение для поддержки большого массива для pi-camera. Так мы сможем преобразовывать данные из этого массива для использования в NumPy и OpenCV:

```
pi@myrobot:~ $ sudo pip3 install picamera[array]
```

Следующие несколько операций мы также будем тестировать при подключении к внешнему источнику питания.

Вы подготовили программные библиотеки и убедились, что камера может делать снимки. Далее создадим приложение для потоковой передачи видео с камеры в веб-браузер.

## СОЗДАНИЕ ПРИЛОЖЕНИЯ ДЛЯ ПОТОКОВОЙ ПЕРЕДАЧИ ДАННЫХ КАМЕРЫ RASPBERRY PI

Загрузка изображений по одному – это хорошо, но нужно иметь возможность что-то делать с ними на нашем роботе. Также нам необходим удобный способ для просмотра того, что робот делает с данными камеры. В этом разделе вы узнаете, как просматривать потоковый вывод камеры на смартфоне или ноутбуке с помощью веб-сервера Flask. На ядре этого приложения мы разработаем несколько поведенческих сценариев. Все они будут использовать одно базовое приложение.

Видео или видеопоток – это последовательность изображений, обычно называемых **кадрами**.

Перейдем к разработке нашего потокового сервера.

## Разработка потокового сервера OpenCV

На рис. 13.9 в виде схемы показан конвейер данных изображения. Он начинается с камеры, затем происходит обработка, и изображение попадает в наш веб-браузер.

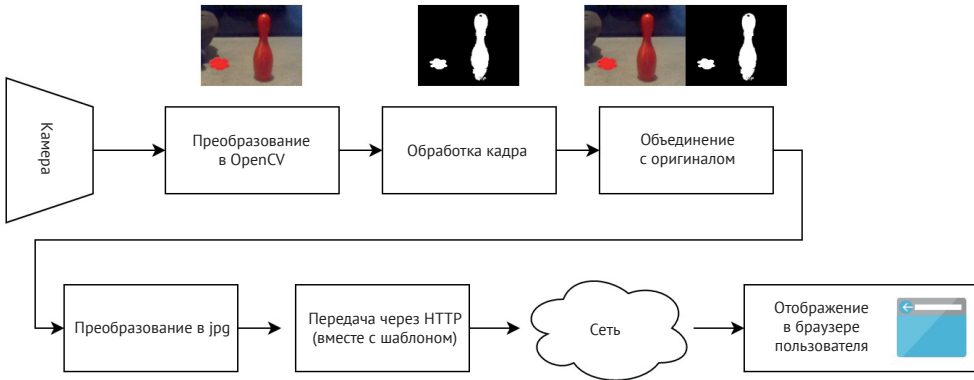


Рис. 13.9. Приложение сервера изображений

Как показано на рис. 13.9, процесс начинается с камеры. Она передает данные изображения (необработанный снимок) на шаг **преобразования в OpenCV**. Данные изображения нуждаются в обработке, чтобы OpenCV мог ими оперировать.

За шагом **преобразования в OpenCV** следует шаг **обработки кадра** (этот шаг может быть всем, что нам нужно). В этом примере мы применим цветовую маску, которую рассмотрим более подробно в следующем разделе. На рис. 13.9 показан пример изображения после наложения маски красного цвета.

Необработанный и обработанный кадры переходят в следующий шаг – **объединение с оригиналом**. На этом шаге из двух изображений создается одно составное. На рис. 13.9 мы видим два изображения объединенных в один кадр.

Затем кадр переходит на шаг **преобразования в jpg**. Нам нужно закодировать его с помощью jpeg. Так мы преобразуем изображение в формат, который браузер может отображать в виде последовательности кадров.

Закодированные данные переходят на шаг **передачи через HTTP**, передавая данные в систему, которую вы можете просматривать через веб-браузер. Для этого используется шаблон (макет и текст для браузера).

Затем выходное изображение **передается по HTTP** через сеть в браузер к пользователю. Наконец, браузер отображает изображение для пользователя. Вы можете просматривать изображение в браузере как на ноутбуке, так и на смартфоне.

Пришло время приступить к созданию кода. Мы разделим его на две основные части: объект `CameraStream`, который будет отправлять наши кадры во вторую часть – скрипт `image_server.py`.

## Создание объекта CameraStream

В рамках нашей системы мы создадим вспомогательную библиотеку для настройки камеры и получения от нее потоков данных.

1. Начнем файл скрипта camera\_stream.py со следующих импортов:

```
from picamera.array import PiRGBArray
from picamera import PiCamera
import numpy as np
import cv2
```

Эти импорты дают нам код PiCamera, необходимый для доступа к камере. cv2 – это OpenCV, библиотека компьютерного зрения, используемая для обработки изображений. NumPy здесь имеет *сокращенное* имя np.

2. Следующие несколько строк задают параметры размера захвата и качества изображения:

```
size = (320, 240)
encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 90]
```

Мы будем сохранять изображения с небольшим разрешением 320 на 240. Так данных для отправки и обработки будет меньше, что позволит системе работать быстрее. Высокое разрешение может привести к возникновению шума и дефектов краев, которые требуют очистки с помощью дополнительных фильтров. Преобразование изображений для отправки в браузер осуществляется с помощью параметра encode.

3. Добавим функцию для настройки камеры:

```
def setup_camera():
    camera = PiCamera()
    camera.resolution = size
    camera.rotation = 180
    return camera
```

После инициализации камеры устанавливаем ее разрешение и размер захвата. Я упомянул, что камера установлена вверх ногами, поэтому, чтобы повернуть изображение, зададим поворот на 180°.

4. Нам понадобится функция для запуска захвата потока изображений (как видео, но по одному кадру за раз):

```
def start_stream(camera):
    image_storage = PiRGBArray(camera, size=size)
    cam_stream = camera.capture_continuous(image_storage,
    format="bgr", use_video_port=True)
```

Чтобы сохранить наше изображение, нужно создать экземпляр PiRGBArray (тип данных для хранения изображений RGB). Затем мы настраиваем поток данных с помощью capture\_continuous – метода picamera для циклической съемки. Мы передаем его в хранилище изображений и задаем форматирование выходных данных как bgr (синий, зеленый, красный) – именно в таком виде OpenCV хранит данные о цвете. Последним параметром является use\_video\_port, который, если установлено значение true, снижает качество изображения, за счет чего кадры генерируются быстрее.

- Мы можем выполнять цикл `cam_stream` до тех пор, пока не решим остановиться. В Python существуют **итераторы** – структуры данных для таких последовательностей, как списки и генераторы. Генераторы – это последовательности, которые производят следующую порцию данных только тогда, когда это необходимо:

```
for raw_frame in cam_stream:
    yield raw_frame.array
    image_storage.truncate(0)
```

Цикл `for` – это генератор. Каждый цикл производит необработанный модуль `.array` из кадра, захваченного потоком. Это означает, что цикл может использовать вывод функции `start_stream`, поэтому при заикли-вании код в цикле `for` будет выполняться ровно столько, сколько нужно для создания одного необработанного кадра, затем для следующего кадра и т. д. Генераторы Python – это способ построения конвейеров обработки.

Последняя строка цикла вызывает метод `truncate` для сброса функции `image_storage`, готовой к сохранению следующего изображения. `PiRGBArray` может сохранять множество изображений последовательно, но нам нужно только последнее. Во время обработки кадра могло прийти более одного изображения, поэтому мы должны отбросить лишние.

- Последнее, что мы добавляем в сценарий `camera_stream.py`, – это функция для кодирования изображения в формат `jpeg`, а затем в байты для отправки, как показано в следующем фрагменте кода:

```
def get_encoded_bytes_for_frame(frame):
    result, encoded_image = cv2.imencode('.jpg', frame,
    encode_param)
    return encoded_image.tostring()
```

Мы будем использовать библиотеку `camera_stream` для других поведенческих скриптов. Это дает возможность извлекать и кодировать кадры с камеры, как готовые для ввода, так и закодированные для отображения. Теперь, когда все готово, протестируем приложение для отображения кадров в браузере.

## Создание главного приложения сервера изображений

Эта часть приложения настроит Flask, запустит поток с нашей камеры и свяжет их вместе. Создадим новый скрипт с именем `image_server.py`.

- Нам нужно импортировать все указанные ниже компоненты и настроить приложение Flask:

```
from flask import Flask, render_template, Response
import camera_stream
import time

app = Flask(__name__)
```

Мы импортируем несколько сервисов из Flask: объект приложения `Flask`, отвечающий за маршрутизацию; способ рендеринга шаблонов в выходные данные и способ, которым наше веб-приложение будет отвечать. За-

тем импортируем недавно созданную библиотеку `camera_stream` и библиотеку `time` для ограничения частоты кадров. После импорта мы создаем объект Flask `app` для регистрации.

2. Flask работает с маршрутами, представляющими собой ссылки между адресом, по которому вы обращаетесь к веб-серверу, и зарегистрированной функцией-обработчиком. Запрос к определенному адресу в серверном приложении запустит соответствующую функцию. Настроим самый простой маршрут:

```
@app.route('/')  
def index():  
    return render_template('image_server.html')
```

Маршрут `'/'` является страницей `index`, которая откроется по умолчанию, если вы зайдете на сервер приложений робота. Функция выполняет рендеринг шаблона, который мы создадим в следующем разделе.

3. Теперь мы подошли к самому сложному – трансляции видео. Несмотря на то что `camera_stream` выполняет кодирование, нам нужно преобразовать кадры в поток данных HTTP, т. е. в данные такого формата, которые браузер будет считать непрерывными. Для этого объявляем функцию `frame_generator`, которую нам придется разбить на части. Начнем с настройки потока данных камеры:

```
def frame_generator():  
    camera = camera_stream.setup_camera()  
    time.sleep(0.1)
```

Метод `time.sleep` необходим, чтобы дать камере время для запуска после включения. Иначе мы не сможем получить от нее пригодные для использования кадры.

4. Далее пройдемся по кадрам из `camera_stream`:

```
for frame in camera_stream.start_stream(camera):  
    encoded_bytes = camera_stream.get_encoded_bytes_  
for_frame(frame)
```

Эта функция – еще один генератор Python, перебирающий каждый кадр, поступающий из `start_stream` и кодирующий их в JPG.

5. Чтобы отправить байты закодированного кадра в браузер, мы используем другой генератор с `yield`. Flask считает это составным потоком – ответом, состоящим из нескольких фрагментов данных, с частями, отложенными на потом, который будет состоять из множества кадров одного и того же видео. Обратите внимание, что в объявлениях содержимого HTTP к закодированным байтам добавляются префиксы:

```
yield (b'--frame\r\n'  
       b'Content-Type: image/jpeg\r\n\r\n' +  
       encoded_bytes + b'\r\n')
```

Мы помещаем `b` перед строкой, чтобы указать Python обрабатывать ее как необработанные байты и не выполнять дальнейшее кодирование информации. Элементы `\r` и `\n` – это необработанные символы конца строки. На этом функция `frame_generator` завершается.

- Следующая функция с именем `display` задает маршрут из Flask в зацикленный поток HTTP-кадров из `frame_generator`:

```
@app.route('/display')
def display():
    return Response(frame_generator(),
                    mimetype='multipart/x-mixed-replace;
                    boundary=frame')
```

Маршрут `display` генерирует ответ от `frame_generator`. Поскольку это генератор, Flask будет продолжать отправлять его элементы в браузер.

В ответе также указывается тип содержимого с границей между элементами. Эта граница должна быть строкой символов. Мы использовали `frame`. Граница должна совпадать в `mimetype` и границей (`--frame`) в содержимом (*шаг 5*).

- Наконец мы можем добавить код для запуска Flask. Я настроил приложение на порт 5001:

```
app.run(host="0.0.0.0", debug=True, port=5001)
```

Приложение почти готово, но ранее мы говорили, что необходимо создать шаблон. Давайте воспользуемся им, чтобы описать, что будет происходить на веб-странице, отображающей поток данных камеры.

## Создание шаблона

Flask создает веб-страницы, используя шаблоны HTML, которые маршрутизируют функции в выходные данные, при необходимости заменяя некоторые элементы. Создайте папку `templates`, а в ней файл с именем `image_server.html`.

- Наш шаблон начинается с тега `<HTML>`, тега `<title>` и заголовка первого уровня:

```
<html>
<head>
  <title>Robot Image Server</title>
</head>
<body>
  <h1>Robot Image Server</h1>
```

- Теперь мы добавляем ссылку на изображение, которая будет отображать выходные данные сервера:

```

```

Обратите внимание на элемент `url_for`. Для вставки URL-адреса из маршрута по имени его функции Flask может использовать средство рендеринга шаблонов Jinja.

- Наконец, закрываем теги в шаблоне:

```
</body>
</html>
```

Мы можем использовать этот шаблон в основном серверном приложении.

Теперь можем загрузить все три части кода, но не забудьте загрузить шаблон в каталог `templates` на Pi.

Когда код сервера и шаблоны готовы, мы можем перейти к запуску сервера изображений.

## Запуск сервера изображений

Запустите приложение с помощью команды `python3 image_server.py`.

В браузере перейдите по адресу <http://myrobot.local:5001> (для вашего робота может быть другой адрес). Вы должны увидеть изображение с камеры, как показано на рис. 13.10.



Рис. 13.10. Скриншот сервера изображений робота

На рис. 13.10 показан скриншот вывода нашего сервера изображений в браузере. Сверху мы видим панель поиска браузера с адресом `myrobot.local:5001`. Ниже находится заголовок **Robot Image Server** из шаблона, а под заголовком снятое камерой изображение красной кегли из детского набора для боулинга, полученное из кода видеопотока.

## Устранение неполадок

Если у вас возникли проблемы с запуском сервера и просмотром изображения, воспользуйтесь следующими советами:

- в случае возникновения ошибок при запуске кода выполните следующие шаги:
  - а) проверьте, осуществляется ли захват изображения посредством `raspistill`;
  - б) проверьте, установили ли вы все необходимые зависимости;
  - в) при возникновении ошибки `libatomic` проверьте, выполнены ли экспорты `LD_PRELOAD`;
  - г) проверьте код на предмет опечаток;



- если изображение слишком темное, возможно, камере не хватает света. Камера Raspberry Pi чувствительна к условиям освещения, поэтому пространство должно быть хорошо освещено. Обратите внимание, что, если камера не будет получать достаточно света, ни один из сценариев отслеживания, о которых мы поговорим в следующих разделах, работать не будет;
- помните, что частота кадров и качество изображения будут низкими – не ждите от этой камеры слишком многого.

Теперь вы можете передавать изображения с Raspberry Pi в браузер. Далее мы реализуем возможность запуска фоновых задач и добавим механизм управления, так как сейчас весь сервер зависит от медленного цикла запросов браузера.

## ЗАПУСК ФОНОВЫХ ЗАДАЧ ВО ВРЕМЯ ПОТОКОВОЙ ПЕРЕДАЧИ

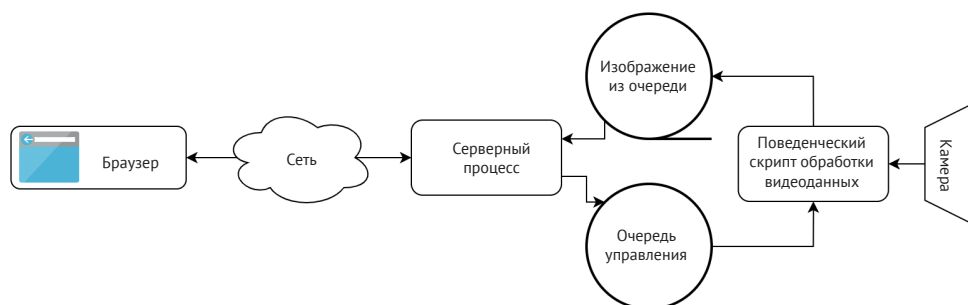
Сейчас наш сервис изображений работает, но у него есть один существенный недостаток. Перед выполнением каждого действия он будет делать паузу между запросами. Но что, если мы хотим, чтобы наш робот выполнял какое-либо действие? Для этого нужно иметь возможность запускать поведенческий скрипт параллельно с сервером. И скрипт, и сервер должны иметь доступ к данным изображения.

Мы реализуем следующий подход: веб-приложение Flask будет вторичным процессом, а поведенческий скрипт – основным (во время его выполнения). В Python есть удобный инструмент для такой структуры – *многопроцессная обработка*<sup>13</sup>. Узнать об этом больше можно в материалах по адресу <https://docs.python.org/3/library/multiprocessing.html>.

Связь между несколькими процессами сложна. Если два процесса одновременно пытаются получить доступ к одним и тем же данным (например, для чтения или записи), результаты могут быть непредсказуемыми и робот может начать вести себя странно. Чтобы решить эту проблему, мы будем использовать объект многопроцессной очереди. Такой подход позволит одному процессу размещать данные на одном конце, а второму безопасно получать их на другом – это односторонний поток информации. Мы будем использовать одну очередь для отправки изображений на сервер, а другую – для получения данных управления от действий пользователя в браузере.

На рис. 13.11 показана схема прохождения потока данных в таких поведенческих сценариях.

<sup>13</sup> Не путайте с *многопроцессорной* обработкой. – Прим. ред.



**Рис. 13.11.** Поток данных между браузером, серверным процессом и поведенческим скриптом робота

На рис. 13.11 показано, что мы сокращаем некоторые шаги с рис. 13.9. Во-первых, данные с камеры переходят в поведенческий скрипт обработки видеоданных (например, отслеживание объекта). Этот скрипт будет выводить кадры изображения в очередь. На выходе мы получим объединенное и полностью обработанное изображение.

Серверный процесс (веб-приложение) будет брать изображения из очереди и передавать их в браузер через сеть. Одновременно с этим веб-приложение будет обрабатывать команды пользователя. Приложение поместит их в очередь управления в виде сообщений. Поведенческий скрипт обработки видеоданных будет считывать сообщения из очереди управления и обрабатывать их.

Обратите внимание на некоторые моменты: поведенческий скрипт обработки видеоданных будет помещать изображения в очередь изображений только тогда, когда она пуста, поэтому очередь всегда будет содержать только одно изображение. Это предотвратит перезапись изображения в общую память, когда сервер будет пытаться вывести его. В очереди управления тако-го ограничения нет, поэтому мы просто ожидаем, что команды пользователя не будут генерировать управляющие сообщения быстрее, чем цикл поведения может их использовать.

Мы создадим ядро веб-приложения, а затем на его основе разработаем поведенческий скрипт. Ядро мы сможем использовать несколько раз. Перейдем к коду.

## Создание ядра веб-приложения

В этой структуре ядро приложения будет обрабатывать настройку очередей, запуск серверного процесса и маршрутизацию на основе Flask. Мы создадим библиотеку в стиле Flask, используя простые функции Python в модуле.

Выступая в роли интерфейсов к ядру, наши поведенческие скрипты будут делать следующее:

- `start_server_process(template_name)` будет запускать сервер веб-приложений, используя шаблон с определенным именем;
- `put_output_image(encoded_bytes)` будет помещать изображения в очередь на отображение;
- `get_control_instruction()` будет использоваться для проверки и возврата инструкций из очереди управления. Эта функция возвращает словарь управляющей информации.

Часть приложения Flask/веб-сервер независима от поведенческого скрипта, что позволяет пользователю *подстроить под себя* ее отображение. Однако это не должно останавливать работу приложения, когда пользователь отсутствует или браузер зависает.

1. Начнем с импортов. Мы поместим этот код в файл `image_app_core.py`:

```
import time
from multiprocessing import Process, Queue

from flask import Flask, render_template, Response
```

Чтобы создать процесс и взаимодействовать с ним, мы импортируем `Queue` и `Process`. Затем импортируем `Flask`, так же как делали это ранее. Обратите внимание, в этом модуле мы *не* импортируем части кода для камеры.

2. Далее определяем приложение Flask и очереди. На самом деле нам нужен только один кадр в очереди, но мы помещаем туда еще один на случай сбоев при передаче. Хотя мы и можем проверить, пуст ли экземпляр очереди, но такой подход не на 100 % надежен, и мы не хотим, чтобы какая-то часть приложения ждала другую:

```
app = Flask(__name__)
control_queue = Queue()
display_queue = Queue(maxsize=2)
```

3. Определяем глобальный шаблон `display_template`, в котором будет храниться основной шаблон приложения:

```
display_template = 'image_server.html'
```

4. Теперь добавляем маршруты для приложения Flask. Маршрут индексной страницы отличается только тем, что он использует `display_template`:

```
@app.route('/')
def index():
    return render_template(display_template)
```

5. Далее создаем цикл для получения кадров (модифицированную версию `frame_generator`). Эта функция является нашим основным источником видеосигнала. Чтобы в цикле *не было длительных задержек* (т. е. он работал очень быстро в сплошном цикле), мы ставим паузу длительностью 0,05 с, чтобы ограничить частоту кадров в секунду до 20:

```
def frame_generator():
    while True:
        time.sleep(0.05)
```

6. После паузы необходимо получить данные из `display_queue` (позже мы поместим кадры в очередь). Как и в случае с `image_server`, этот цикл превращает обычные данные в данные, состоящие из нескольких частей:

```
encoded_bytes = display_queue.get()
yield (b'--frame\r\n'
       b'Content-Type: image/jpeg\r\n\r\n' +
       encoded_bytes + b'\r\n')
```

7. Теперь откроем к ним доступ через блок отображения:

```
@app.route('/display')
def display():
    return Response(frame_generator(),
                    mimetype='multipart/x-mixed-replace;
                    boundary=frame')
```

8. Нам нужен способ отправлять управляющие сообщения приложению. Маршрут `control` принимает их, берет данные формы (словарь с командами) и передает в поведенческий скрипт с помощью `control_queue.put`:

```
@app.route('/control', methods=['POST'])
def control():
    control_queue.put(request.form)
    return Response('queued')
```

9. Теперь мы имеем доступ ко всем внутренним функциям ядра, но одновременно с этим нам нужно запустить серверный процесс. Часть предыдущего приложения, которая запускала сервер, мы помещаем в функцию с именем `start_server_process`:

```
def start_server_process(template_name):
    global display_template
    display_template = template_name
    server = Process(target=app.run, kwargs={"host":
    "0.0.0.0", "port": 5001})
    server.start()
    return server
```

Мы предполагаем, что поведенческий скрипт запускает эту функцию, передавая имя пользовательского шаблона. Он сохраняет `template_name` в глобальном `display_template`. Предыдущий маршрут `index` использует шаблон. Вместо вызова `app.run` мы создаем объект `Process`. Параметр `target` объекта `Process` – это функция, которую необходимо запустить (`app.run`). Для нее также необходимо указать некоторые параметры (настройки хоста и порта). Затем запускаем серверный процесс и возвращаем дескриптор процесса, чтобы позже наш код мог остановить его.

10. Следующая задача интерфейса – поместить изображение в очередь, которую мы создали в *шаге 1*. Чтобы не допустить использования чрезмерного количества памяти, мы предполагаем, что длина очереди будет равна единице. Первый кадр будет устаревшим, но следующий прибудет достаточно быстро, поэтому пользователь ничего не заметит:

```
def put_output_image(encoded_bytes):
    if display_queue.empty():
        display_queue.put(encoded_bytes)
```

11. Наконец, для этого интерфейса нам нужна функция вывода управляющих сообщений. Эта функция не будет ждать и вернет сообщение, если оно есть, а если *очередь пуста*, то вернет ответ `None`:

```
def get_control_instruction():
    if control_queue.empty():
        return None
    else:
        return control_queue.get()
```

Файл `image_app_core.py` образует управляемую основу для построения поведенческого скрипта, обрабатывающего визуальную информацию, или вообще любого скрипта с веб-интерфейсом, инструкциями управления, потоком выходных данных и фоновым процессом. Далее проверим ядро с помощью простого поведенческого скрипта.

## Разработка управляемого поведенческого скрипта

Мы можем протестировать наше ядро посредством поведенческого скрипта, который отправляет изображения на веб-сервер и принимает простое управляющее сообщение `exit`.

1. Создадим новый файл с именем `control_image_behavior.py`, начиная с импорта интерфейса `image_app_core` и `camera_stream`:

```
import time

from image_app_core import start_server_process, get_
control_instruction, put_output_image
import camera_stream
```

2. Затем мы добавляем функцию, которая запускает наш простой скрипт с главным циклом. Я разбил эту функцию на две части, так как она достаточно сложная. Сначала настроим камеру и зададим паузу, во время которой она сможет войти в рабочий режим:

```
def controlled_image_server_behavior():
    camera = camera_stream.setup_camera()
    time.sleep(0.1)
```

3. Мы получаем кадры из потока камеры в цикле `for` и помещаем их в виде закодированных байтов в очередь вывода:

```
for frame in camera_stream.start_stream(camera):
    encoded_bytes = camera_stream.get_encoded_bytes_
for_frame(frame)
    put_output_image(encoded_bytes)
```

4. Находясь в цикле, мы принимаем команду управления для выхода из него. Как правило, приходит инструкция `None`, которая сигнализирует об отсутствии ожидающих команд управления. Но, если нам приходит сообщение, перед выходом мы должны сопоставить команду в нем следующим образом:

```
instruction = get_control_instruction()
if instruction and instruction['command'] ==
"exit":
    print("Stopping")
    return
```

При получении из очереди управления инструкции `exit` обработчик использует `return` и останавливает выполнение скрипта.

- Затем нам нужно запустить сервер и поведенческий скрипт. Процесс веб-сервера нужно остановить. При использовании конструкции `try finally` он *всегда* будет запускать что-либо в части `finally`. Так мы узнаем, что процесс завершен (остановлен):

```
process = start_server_process('control_image_behavior.
html')
try:
    controlled_image_server_behavior()
finally:
    process.terminate()
```

Итак, мы разработали простой управляемый поведенческий скрипт. Однако в нем мы упоминали шаблон `control_image_behavior.html`. Перейдем к его созданию.

## Создание шаблона элемента управления

Шаблон в `templates/control_image_behavior.html` похож на предыдущий, но имеет два важных отличия (выделены жирным шрифтом во фрагменте кода ниже):

```
<html>
<head>
    <script src="https://code.jquery.com/jquery-3.3.1.min.
js"></script>
    <title>Robot Image Server</title>
</head>
<body>
    <h1>Robot Image Server</h1>
    <br>
    <a href="#" onclick="$$.post('/control', {'command':
'exit'}); ">Exit</a>
</body>
</html>
```

Отличия заключаются в следующем:

- в этом шаблоне мы загружаем в наш браузер удобную библиотеку для интерактивных веб-страниц под названием `jquery`. Документацию по `jQuery` можно найти по адресу <https://api.jquery.com/>;
- у нас есть изображение и заголовок, которые мы уже видели раньше, но также в этом коде появляется тег `a` (от слова *anchor* – привязка), который при нажатии отправляет команду `exit` на маршрут `/control` в нашем веб-приложении. Тег `<br>` создает разрыв строки, чтобы отобразить под изображением ссылку для выхода.

Если вы хотите запустить скрипт при отсутствии соединения с интернетом, вам понадобится сервер для получения библиотеки `jquery`. Этот шаблон указывает браузеру загружать `jquery` прямо из интернета.

Теперь, когда все компоненты готовы, мы можем запустить наш управляемый поведенческий скрипт.

## Запуск управляемого сервера изображений

Запустим наши компоненты и протестируем команды.

1. Для запуска сервера изображений необходимо загрузить три файла:
  - a) `image_app_core.py`.
  - b) `control_image_behavior.py`.
  - c) `templates/control_image_behavior.html`.
2. На Pi запустите процесс с помощью команды `python3 control_image_behavior.py`.

Введите в браузере <http://myrobot.local:5001> (или адрес вашего робота). Вы снова увидите изображения.

Нажатие на ссылку **Exit** (Выход) под изображением отправит приложению соответствующую управляющую команду, и оно корректно завершит работу.

Теперь вы знаете, как получать данные изображения из поведенческого скрипта, отправляя ему управляющие данные. Протестировав и подготовив методы управления и потоковой передачи, а также используя подходящую структуру, мы можем разработать более интересный поведенческий скрипт. В следующем разделе заставим робота следовать за объектом определенного цвета.

## ОТСЛЕЖИВАНИЕ ЦВЕТНЫХ ОБЪЕКТОВ С ПОМОЩЬЮ КОДА НА PYTHON

Подготовив основы, мы можем перейти к созданию более интересных поведенческих сценариев.

Разработаем скрипт, который заставит робота следовать за объектом определенного цвета, не подъезжая к нему слишком близко. Так наш робот будет выглядеть еще более «разумным». В этом разделе мы вернемся к цветовым моделям, описанным в главе 9. Мы реализуем цветовое маскирование и фильтрацию, а также используем инструмент OpenCV для очерчивания контуров, с помощью которого робот будет определять самый большой объект на изображении и направляться к нему.

Разработка поведенческого сценария отслеживания объектов определенного цвета состоит из нескольких шагов. Сначала разберемся со структурой такого сценария (см. рис. 13.2).

На рис. 13.2 поток данных начинается с **изображений с камеры**. Затем данные проходят через **обработку видеоданных** и попадают на этап **получения информации об объекте с изображения**. На этом шаге выводится размер объекта (вычисленный на основе радиуса описанной вокруг него окружности) и его положение (середина описанной окружности). Затем кадр помещается в очередь изображений для веб-приложения/браузера.

Информация о размере объекта поступает в **ПИД-регулятор**, в качестве уставки которого выступает ссылка на размер. ПИД регулятор будет корректировать частоту вращения двигателей в зависимости от разницы между ожидаемым и фактическим размерами, тем самым оптимизируя радиус, чтобы он соответствовал ссылке на размер. Таким образом, робот будет поддерживать дистанцию до объекта известного размера. Такая частота вращения становится основной для обоих двигателей.



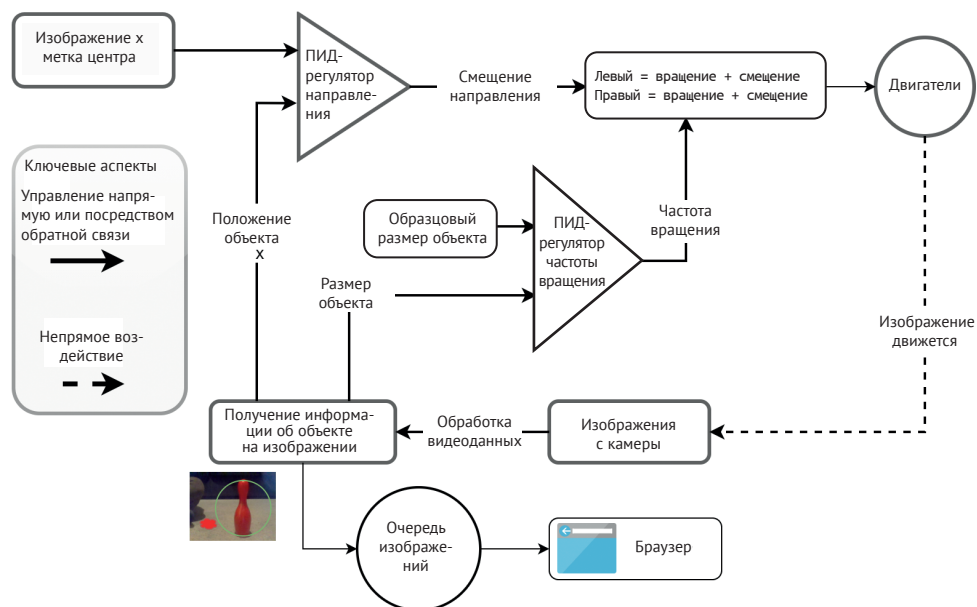


Рис. 13.12. Поведенческий сценарий отслеживания объектов определенного цвета

Положение объекта имеет компоненты  $x$  и  $y$ . В нашем поведенческом сценарии необходимо центрировать объект по горизонтали, поэтому нас интересует координата  $x$ . Координата  $x$  поступает в ПИД-регулятор для управления направлением/курсом. ПИД-регулятор использует исходное положение – центр области просмотра камеры. Регулятор попытается свести разницу между координатами объекта и центра камеры к нулю и выведет соответствующий результат. Частота вращения одного двигателя увеличится, а другого уменьшится, поэтому робот повернется к объекту передней стороной (в качестве эксперимента можно заставить его отвернуться, поменяв двигатели местами).

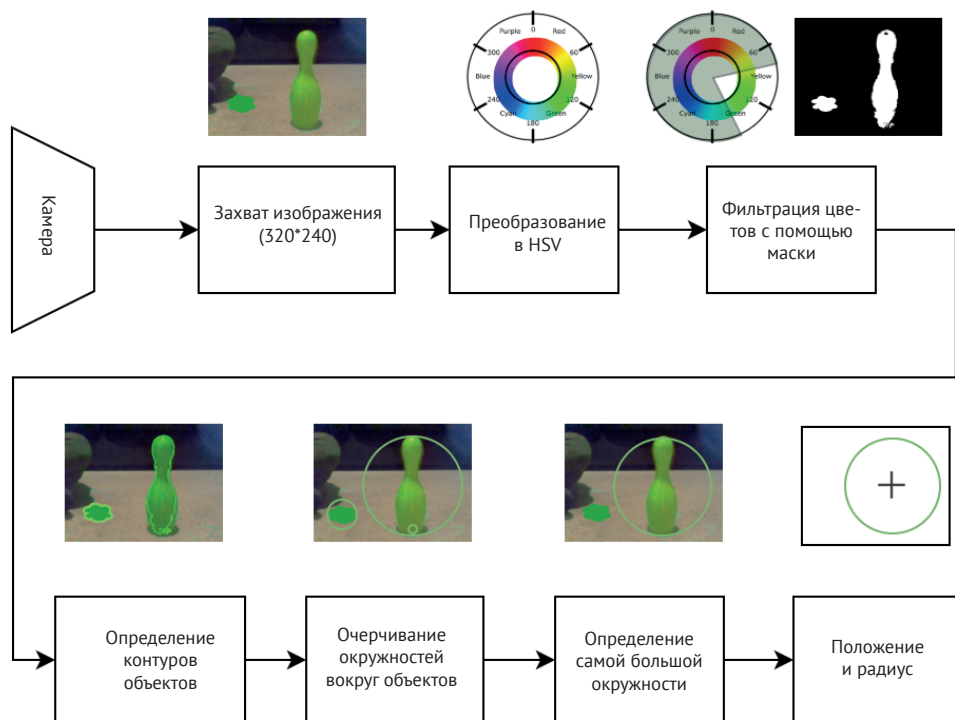
Изображения отправляются в браузер через очередь изображений, используя ядро приложения. На рис. 13.12 не показан один элемент – очередь управления с сообщениями для запуска и остановки двигателей, а также для завершения поведенческого скрипта.

Заключительная часть этой системы и, вероятно, самая интересная – это отслеживание определенного цвета. Отслеживание выполняет блок на схеме с надписью **получить информацию об объекте с изображения**. Рассмотрим, как это работает.

## Преобразование изображения в информацию

Для нашего эксперимента отлично подойдут цветные кегли из детского набора для боулинга. Как правило, они окрашены в яркие базовые цвета. Для примера я буду использовать зеленую кеглю. Чтобы превратить изображение в информацию, которую робот может использовать для принятия решений, необходимо выполнить ряд преобразований.

В этом нам поможет конвейер. На рис. 13.13 показан конвейер обработки изображения.



**Рис. 13.13.** Получение информации о цвете объекта с камеры

Как и в других конвейерах, на рис. 13.13 поток данных начинается с камеры. Для ускорения процесса качество изображения понижается. На этой части рисунка над блоком с текстом показано, как выглядит изображение с камеры.

Затем выходные данные преобразуются в HSV (цветовая модель, которую мы обсуждали в главе 9). Мы используем HSV, поскольку данная модель позволяет фильтровать цвета в определенном диапазоне цветового тона по их светлоте (темные объекты могут сбить систему с толку) и насыщенности (так, почти серые объекты будут игнорироваться). Изображения RGB (или BGR) сложно фильтровать, поскольку получение различных уровней освещенности и насыщенности определенного оттенка (скажем, синего) нецелесообразно. На этой части рисунка над соответствующим текстовым блоком показан цветовой круг HSV.

В OpenCV есть функция `cv2.cvtColor`, которая преобразовывает целые изображения в разные цветовые модели. Обратите внимание, что OpenCV использует диапазон цветового тона 0–179, а не 0–359. Этот диапазон умещается в байт (0–255), но, если вы знаете нужное значение, значение цветового тона можно преобразовать, просто разделив его на 2.

После преобразования в HSV мы фильтруем цвета на изображении с помощью маски, выделяя пиксели определенного диапазона. Если пиксель нахо-

дится внутри диапазона, то он станет белым, а если вне диапазона, то черным. На этой части рисунка вы можете видеть две картинки – слева показан диапазон (незаштрихованная часть цветового круга), а справа – результат наложения маски на изображение. В OpenCV для этого есть специальная функция – `cv2.inRange`. Она дает простой двоичный вывод для нашей системы – изображение после наложения маски.

Затем конвейер использует систему рисования контура, чтобы очертить контуры на нашем изображении с маской. Контур определяет только точки границ объекта. В OpenCV есть функция `cv2.findContours`, которая возвращает список фигур, каждая из которых определяется собственным контуром. На этой части рисунка показаны контуры (взятые из маски), очерченные на необработанном изображении. Обратите внимание, как из-за света и тени контур нижней части кегли стал неровным.

Далее конвейер обработки очерчивает окружности вокруг контуров с помощью функции `cv2.minEnclosingCircle`. Так у нас получается несколько окружностей, имеющих центр с координатами  $x$ ,  $y$  и радиус. На этой части рисунка окружности показаны на необработанном изображении.

На объекте могут быть блики, вокруг которых также будет очерчена окружность. Также маленькие окружности могут появляться вокруг других объектов. Из всех этих окружностей нас интересует только одна – самая большая, поэтому нам нужно перебрать все и оставить только ее. На этой части рисунка (**выделение самой большой окружности**) показано необработанное изображение с одной самой большой окружностью.

Зная координаты и радиус этой окружности, робот будет иметь достаточно информации для следования за объектом. На этой части рисунка показана окружность с перекрестием в центре, которое обозначает положение объекта.

### Важное примечание

Об объектах красного цвета: мы используем объект зеленого цвета, потому что для красного требуется две маски, ввиду чего все становится немного сложнее. Цветовые тона красного пересекают границу между 179 (верхняя граница диапазона) и 0 (нижняя граница), поэтому на такое изображение пришлось бы накладывать маску дважды, а затем объединять результат с помощью операции `or`. Если хотите попробовать наложить маску на красный цвет, воспользуйтесь функцией `cv2.bitwise_or`.

Теперь вы знаете, как работает конвейер и осведомлены о его особенностях. Вы увидели, как он взаимодействует с ПИД-регулятором в рамках описываемого поведенческого сценария. Теперь мы перейдем к разработке кода.

## Усовершенствование ПИД-регулятора

Мы собираемся использовать несколько ПИД-регуляторов. Дифференциальный компонент нам по-прежнему не требуется, но в этом разделе мы решим проблему с увеличением значения интегрального компонента, связанную с тем, что двигателям нужно время, чтобы начать двигаться. Интегральный

компонент представляет собой сумму, которая начинает расти при наличии постоянной ошибки. Эту ошибку можно попытаться исправить, но это может привести к выбросу в процессе регулирования. Причиной выброса станет возрастание значения интегрального компонента после того, как робот начнет реагировать (пусть и медленно). Такой выброс называется **выбегом интегрального компонента**.

Задав предел выбега для ПИД-регулятора, можно предотвратить увеличение суммы.

1. Откройте файл `pid_controller.py` и внесите изменения, выделенные жирным шрифтом в следующем фрагменте кода. Добавьте параметр `windup_limit`, который по умолчанию (при отсутствии ограничения) имеет значение `None`:

```
class PIController(object):
    def __init__(self, proportional_constant=0, integral_
constant=0, windup_limit=None):
        self.proportional_constant = proportional_
constant
        self.integral_constant = integral_constant self.windup_limit = windup_
limit
        self.integral_sum = 0
```

2. Если значение интегрального компонента достигает установленного предела и превышает его, нам необходимо остановить его увеличение. Интегральное значение изменится при наличии одного из следующих условий:

- а) предел выбега не установлен (установлен на `None`);
- б) абсолютное значение суммы ниже предела выбега;
- с) знак ошибки (противоположный) уменьшает значение суммы.

Эти условия позволят не превышать установленный предел.

Давайте посмотрим, как это работает в коде. Следующий фрагмент заменит предыдущий метод `handle_integral`:

```
def handle_integral(self, error):
    if self.windup_limit is None or \
        abs(self.integral_sum) < self.windup_
limit) or \
        ((error > 0) != (self.integral_sum > 0)):
        self.integral_sum += error
    return self.integral_constant * self.integral_sum
```

3. Мы можем запускать (`start`) и останавливать (`stop`) этот поведенческий скрипт на веб-странице. Также нам нужен способ сброса старых значений ПИД-регулятора. Для обнуления интегральной суммы добавим функцию `reset`:

```
def reset(self):
    self.integral_sum = 0
```

Теперь мы можем сбрасывать значения ПИД-регулятора. Также мы установили предел выбега интегрального компонента. Далее перейдем к созданию других компонентов скрипта.

## Создание компонентов поведенческого скрипта

Наш скрипт состоит из двух файлов – шаблона (для передачи в ядро приложения с помощью кнопок управления) и основного кода. Начнем с создания шаблона.

### Создание шаблона элемента управления

Этот шаблон предназначен для потокового приложения и имеет новые элементы управления.

1. Скопируйте шаблон из `templates/control_image_behavior.html` и вставьте в `templates/color_track_behavior.html`.
2. Мы добавим к нему два дополнительных элемента управления – `start` и `stop` (в фрагменте кода ниже выделены жирным шрифтом):

```
<br>
<a href="#" onclick="$.post('/control',
{'command': 'start'});">Start</a>
<a href="#" onclick="$.post('/control',
{'command': 'stop'});">Stop</a><br>
<a href="#" onclick="$.post('/control',
{'command': 'exit'});">Exit</a>
```

Сначала мы запустим программу, когда робот будет стоять на месте. Затем выполним настройку через браузер на смартфоне или компьютере, посмотрим, что обнаружит робот, а затем нажатием на кнопку **Start** (Пуск) заставим его двигаться.

После внесения изменений в шаблон нам необходимо разработать основной код скрипта.

### Разработка кода для поведенческого скрипта

Поместим новый скрипт в файл с именем `color_track_behavior.py`.

1. Как всегда, начнем с импортов. Здесь мы объединяем множество элементов, поэтому импортов будет много, но все они нам уже знакомы:

```
import time
from image_app_core import start_server_process, get_
control_instruction, put_output_image
```

```
import cv2
import numpy as np
```

```
import camera_stream
from pid_controller import PIController
from robot import Robot
```

2. Теперь мы добавляем класс `Behavior`, чтобы робот мог найти объект определенного цвета и приблизиться к нему. Передаем его в объект `robot`:

```
class ColorTrackingBehavior:
    def __init__(self, robot):
        self.robot = robot
```

3. Следующие значения задают настройки цветовой маски и размер объекта:

```

self.low_range = (25, 70, 25)
self.high_range = (80, 255, 255)
self.correct_radius = 120
self.center = 160

```

Для цветового фильтра мы используем значения `low_range` и `high_range` (как показано на рис. 13.13). На изображении с наложенной цветовой маской цвета, лежащие между этими диапазонами HSV, будут отображаться белым. Диапазон нашего цветового тона 25–80, что соответствует диапазону от 50 до 160° на цветовом круге. Диапазон насыщенности составляет 70–255 – если значение ниже, то цвета будут размытыми и серыми. Диапазон яркости составляет от 25 (очень темный) до 255 (полностью светлый).

Значение `correct_radius` задает размер объекта и выступает в качестве настройки расстояния. Значение `center` должно быть равно половине горизонтального разрешения изображения.

- Последняя переменная – это `running`. Если мы хотим, чтобы робот двигался, для нее будет установлено значение `True`. Если для нее устанавливается значение `False`, обработка продолжится, но двигатели и ПИД-регуляторы приостановят работу:

```
self.running = False
```

- Следующий фрагмент кода предназначен для обработки любых управляющих команд из веб-приложения:

```

def process_control(self):
    instruction = get_control_instruction()
    if instruction:
        command = instruction['command']
        if command == "start":
            self.running = True
        elif command == "stop":
            self.running = False
        if command == "exit":
            print("Stopping")
            exit()

```

Этот код обслуживает кнопки `start`, `stop` и `exit`. Для запуска или остановки движения робота здесь используется переменная `running`.

- Сейчас у нас есть код для обнаружения объекта с изображения, который реализует конвейер, показанный на рис. 13.13. Мы немного переработаем эту функцию:

```

def find_object(self, original_frame):
    """Нахождение самой большой описанной окружности среди всех
    контуров в изображении с наложенной маской.
    Возвращает изображение с наложенной маской, координаты и радиус
    объекта"""

```

Код достаточно сложный, поэтому используем **строки документации**, в которых поясняем, что этот код делает и что он возвращает.

- Затем метод преобразует кадр в HSV, поэтому его можно отфильтровать с помощью `inRange` и оставить только пиксели `masked`:

```

frame_hsv = cv2.cvtColor(original_frame, cv2.
COLOR_BGR2HSV)
masked = cv2.inRange(frame_hsv, self.low_range,
self.high_range)

```

8. Теперь, когда у нас есть изображение с наложенной маской, мы можем очертить контуры (точки границ):

```

contours, _ = cv2.findContours(masked, cv2.RETR_
LIST, cv2.CHAIN_APPROX_SIMPLE)

```

При нахождении контуров сначала необходимо определить изображение, на котором их нужно найти. Затем указываем, как извлекаются контуры; мы указали простой список, используя RETR\_LIST. OpenCV поддерживает более детализованные типы, но для их обработки требуется больше времени.

Последний параметр – это метод для определения контуров. Мы используем метод CHAIN\_APPROX\_SIMPLE, который упростит контур до простого набора точек (например, четыре точки для прямоугольника). В возвращенных значениях мы видим символ `_`. Метод может вернуть иерархию, но она нам не нужна. Символ `_` означает, что такие значения нужно игнорировать.

9. Следующий шаг – поиск всех окружностей для каждого контура. Для этого мы используем небольшой цикл. Метод `minEnclosingCircle` получает наименьшую окружность, которая охватывает все точки контура:

```

circles = [cv2.minEnclosingCircle(cnt) for cnt in
contours]

```

cv2 возвращает данные о каждой окружности, т. е. ее радиус и координаты (как раз то, что нам нужно).

10. Нам нужна только самая большая окружность. Отфильтруем остальные:

```

largest = (0, 0), 0
for (x, y), radius in circles:
    if radius > largest[1]:
        largest = (int(x), int(y)), int(radius)

```

Мы сохраняем значение `largest` равным 0, а затем перебираем окружности. Если радиус текущей окружности больше, чем радиус последней сохраненной окружности, то заменяем сохраненную на текущую. Здесь мы преобразуем значения в `int`, поскольку `minEnclosingCircle` генерирует дробные числа с плавающей запятой.

11. Завершаем метод, возвращая изображение с наложенной маской, самые большие координаты и самый большой радиус:

```

return masked, largest[0], largest[1]

```

12. Следующий метод объединит исходный и обработанный кадры. В результате в очереди вывода в веб-приложение мы увидим два изображения одинакового масштаба, соединенных по горизонтали:

```

def make_display(self, frame, processed):
    display_frame = np.concatenate((frame,

```



```
processed), axis=1)
    encoded_bytes = camera_stream.get_encoded_bytes_
    for_frame(display_frame)
    put_output_image(encoded_bytes)
```

Данный метод использует функцию `np.concatenate` для соединения двух изображений, эквивалентных массивам NumPy. Если вы хотите, чтобы изображения соединялись по вертикали, задайте параметру `axis` значение 0.

13. Следующий метод обрабатывает фрейм данных с помощью предыдущих функций, находя объекты и настраивая отображение. Затем он возвращает информацию об объекте:

```
def process_frame(self, frame):
    masked, coordinates, radius = self.find_
    object(frame)
    processed = cv2.cvtColor(masked, cv2.COLOR_
    GRAY2BGR)
    cv2.circle(frame, coordinates, radius, [255, 0,
    0])
    self.make_display(frame, processed)
    return coordinates, radius
```

Обратите внимание, что для замены изображения с наложенной маской на трехканальное изображение мы используем функцию `cvtColor` – чтобы объединить исходный и обработанный кадры, нам нужно чтобы они использовали одну и ту же цветовую модель. Чтобы описать окружность вокруг отслеживаемого объекта на исходном изображении, используем функцию `cv2.circle`. Так мы сможем видеть отслеживаемый объект в веб-приложении.

14. Следующий метод – это наш поведенческий скрипт, преобразующий координаты и радиус в движение робота. При запуске скрипта механизм поворота и наклона может не быть направлен прямо вперед. Нам нужно привести его в нужное положение, установив оба сервопривода на 0, а затем запустить камеру:

```
def run(self):
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
    camera = camera_stream.setup_camera()
```

15. Пока сервоприводы приводят механизм в нужное положение, а камера входит в рабочий режим, мы можем подготовить ПИД-регуляторы частоты вращения (на основе радиуса) и направления (на основе расстояния от середины по горизонтали):

```
speed_pid = PIDController(proportional_
constant=0.8,
    integral_constant=0.1, windup_limit=100)
direction_pid = PIDController(proportional_
constant=0.25,
    integral_constant=0.05, windup_limit=400)
```

Эти значения я получил путем длительной настройки. Вам тоже может понадобится настроить их. Подробнее об этом вы можете узнать в разделе «*Настройка параметров ПИД-регулятора*».

16. Теперь мы немного подождем, пока камера и механизм поворота примут нужное положение, а затем выключим сервоприводы, когда они достигнут центрального положения:

```
time.sleep(0.1)
self.robot.servos.stop_all()
```

17. Посредством оператора `print` мы оповещаем пользователя и выводим несколько заголовков отладки:

```
print("Setup Complete")
print('Radius, Radius error, speed value,
direction error, direction value')
```

18. Затем можем войти в основной цикл. Сначала получаем обработанные данные из кадра. Обратите внимание, что при распаковке `coordinates` в `x` и `y` мы используем скобки:

```
for frame in camera_stream.start_stream(camera):
    (x, y), radius = self.process_frame(frame)
```

19. На этом этапе мы должны проверить управляющие сообщения. Затем проверяем, должен ли робот двигаться, уточняя наличие достаточно большого объекта, чтобы его можно было отслеживать. Если такой объект есть, начинаем:

```
self.process_control()
if self.running and radius > 20:
```

20. Теперь мы знаем, что робот должен двигаться, поэтому необходимо рассчитать значения ошибок для ПИД-регуляторов. Мы получаем ошибку размера и передаем ее в ПИД-регулятор частоты вращения, чтобы получить значения частоты вращения:

```
radius_error = self.correct_radius -
radius
speed_value = speed_pid.get_value(radius_
error)
```

21. С помощью координаты центра и текущего объекта (`x`) вычисляем ошибку направления и передаем ее ПИД-регулятор направления:

```
direction_error = self.center - x
direction_value = direction_pid.get_
value(direction_error)
```

22. Теперь можем выполнить отладку. Выводим на печать отладочное сообщение, соответствующее заголовкам, которые мы видели ранее:

```
print(f"[radius], {radius_error}, {speed_
value:.2f}, {direction_error}, {direction_value:.2f}")
```

23. На основе значений частоты вращения и направления вычисляем частоту вращения левого и правого двигателя:

```

        self.robot.set_left(speed_value -
direction_value)
        self.robot.set_right(speed_value +
direction_value)

```

24. Мы разобрались, что делать, когда двигатели работают. Если двигатели не работают или поблизости нет объекта, который можно отслеживать, то двигатели нужно отключить. После нажатия кнопки **Stop** (Стоп) нужно сбросить значения ПИД-регуляторов, чтобы они не накапливались:

```

else:
    self.robot.stop_motors()
    if not self.running:
        speed_pid.reset()
        direction_pid.reset()

```

25. Мы завершили функцию и класс `ColorTrackingBehavior`. Теперь осталось настроить поведенческий скрипт и ядро веб-приложения, а затем запустить их:

```

print("Setting up")
behavior = ColorTrackingBehavior(Robot())
process = start_server_process('color_track_behavior.
html')
try:
    behavior.run()
finally:
    process.terminate()

```

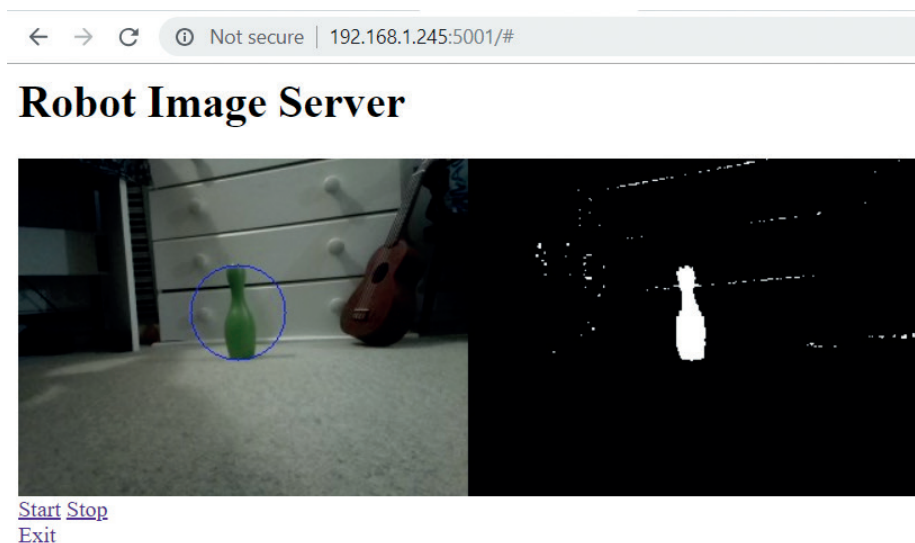
Поведенческий скрипт готов к запуску. Вы узнали, как преобразовать изображение, наложить на него маску для определенного цвета и распознать нужные объекты на таком изображении, а затем найти самый большой из них. Также вы научились преобразовывать результаты обработки видеоданных в движения робота путем передачи данных в ПИД-регуляторы и использования их выходных данных для управления двигателями. Попробуем скрипт в действии!

## Запуск поведенческого скрипта

Я уверен, что вы хотите увидеть, как работает скрипт, и исправить возможные проблемы. Займемся этим.

1. Чтобы запустить скрипт, вам нужно будет загрузить файл `color_track_behavior.py`, файл `pid_controller.py` с внесенными изменениями и шаблон `templates/color_track_behavior.html`. Скорее всего, вы уже загрузили `robot.py` и другие вспомогательные файлы.
2. Запустите приложение с помощью `python3 color_track_behavior.py`. Веб-сервер запустится, а затем последует пауза.
3. Далее зайдите в браузер и подключитесь к <http://myrobot.local:5001>. Здесь вы увидите поток изображений с камеры робота.

Как показано на рис. 13.14, вы увидите объект и очерченную вокруг него окружность, а также ссылки для управления роботом.



**Рис. 13.14.** Скриншот веб-приложения для отслеживания объектов определенного цвета

На рис. 13.14 показан скриншот нашего сервера приложений, на котором запущен код для отслеживания объектов определенного цвета. Под адресной строкой и заголовком находится вывод с двумя изображениями. Слева показан кадр с камеры. На нем (ближе к центру) мы видим зеленую кеглю из детского набора для боулинга, вокруг которой очерчена синяя окружность, показывающая, что робот отслеживает самый большой объект. Справа показан кадр с наложенной маской, который позволяет увидеть, какие аспекты изображения совпадают, и при необходимости настроить значения маски. Ниже находятся ссылки **Start**, **Stop** и **Exit** для запуска, остановки двигателей и выхода из программы соответственно.

4. Чтобы заставить робота двигаться, нажмите на веб-странице на кнопку **Start**.

Когда робот начнет двигаться, на консоли (PuTTY) вы увидите вывод отладки ПИД-регулятора.

5. Чтобы остановить движение робота, нажмите кнопку **Stop**, а чтобы выйти из программы – кнопку **Exit**.

Движения робота не сразу будут правильными. Он может не доезжать до объекта или же, наоборот, приближаться к нему слишком близко. Чтобы это исправить, необходимо настроить параметры ПИД-регуляторов, как описано в следующем разделе.

## Настройка параметров ПИД-регулятора

Я начинаю с пропорциональной константы 0,1 и повышаю ее, используя nano для быстрого редактирования на Pi. Я повышаю значение до тех пор, пока робот не начинает промахиваться (т. е. проходить мимо цели, а затем отъезжать слишком далеко назад). Затем я уменьшаю пропорциональное значение вдвое.

Здесь может возникнуть постоянная ошибка, поэтому, чтобы компенсировать ее, я начинаю повышать интегральную константу примерно на 0,01. Настройка значений ПИД-регулятора – это длительный процесс. Начните с приближения объекта к постоянному центру и настраивайте значение `direction_pid` до получения нужного результата, а затем вернитесь к `speed_pid`.

### Важное примечание

Не пытайтесь настроить все значения сразу – лучше делайте это последовательно.

Более подробно о настройке ПИД-регуляторов рассказывается в материалах, представленных в соответствующем пункте в разделе «Дополнительные материалы».

## Устранение неполадок

Отслеживание цвета – это достаточно сложная задача, поэтому что-то могло пойти не так:

- если двигатели останавливаются или замедляются, замените батарейки;
- при возникновении синтаксических ошибок, внимательно проверьте код;
- проверьте, чтобы примеры веб-приложения взаимодействовали с камерой, и устраните все проблемы, связанные с этим;
- позаботьтесь об освещении, поскольку маска может не улавливать плохо освещенные объекты;
- следите, чтобы маска отображала именно те объекты, которые вам нужны;
- с помощью веб-приложения проверьте, находится ли нужный объект в поле зрения, а также посмотрите, как этот объект выглядит на изображении с маской (он должен быть белым). Если он выглядит не так, как нужно, то вам нужно настроить верхний и нижний диапазоны HSV. Проблемы могут возникнуть с цветовым тоном, поскольку насыщенность и яркость меняются в довольно широких пределах;
- если робот начинает совершать колебательные движения, настройте значения ПИД-регулятора направления (немного уменьшите пропорциональный компонент);
- если робот недостаточно сильно поворачивает, можно немного увеличить значение пропорционального компонента;
- если робот стоит на месте, но не смотрит на обнаруженный объект, то увеличьте значение интегрального компонента ПИД-регулятора направления примерно на 0,01. Если при перемещении вперед и назад возникают одни и те же проблемы, попробуйте применить те же параметры конфигурирования.

В этом разделе вы узнали, как отслеживать объекты ярких цветов с помощью камеры. Эту технику можно использовать для обнаружения объектов в помещении, а промышленные роботы используют ее для обнаружения, например, спелых фруктов. Наблюдать за этим довольно интересно. Однако некоторые объекты обладают не только цветом, но и другими, более сложными признаками (например, лицо). В следующем разделе вы узнаете больше об обнаружении сложных объектов с помощью каскадов признаков.

## ОТСЛЕЖИВАНИЕ ЛИЦ С ПОМОЩЬЮ КОДА НА PUTHON

Обнаружение лиц (или других объектов) на основе признаков сделает нашего робота еще более «разумным». При обнаружении лиц наш робот будет направлять механизм поворота и наклона к ближайшему (самому большому) лицу.

Наиболее распространенной является техника использования **каскадов Хаара**, разработанная Полом Виолой (Paul Viola) и Майклом Джонсом (Michael Jones). Ее часто называют *методом Виолы–Джонса*. Сущность метода заключается в поиске объекта с использованием каскада совпадающих признаков. В этом разделе мы рассмотрим данный метод подробнее, а затем с его помощью реализуем поведенческий сценарий отслеживания лиц. Используя различные файлы каскадных моделей, можно обнаруживать не только лица, но и другие объекты.

### Поиск объекта на изображении

Мы будем использовать алгоритм, реализованный в OpenCV, как единую функцию. Так он становится очень полезным и простым в использовании. Этот подход обеспечивает простой способ обнаружения объектов. Более продвинутые и сложные методы предполагают использование машинного обучения, но многие системы уже используют каскады Хаара, в их числе приложения камеры на смартфонах. Для этого метода наш код преобразует цвета изображения в оттенки серого (по шкале от черного к серому и белому). Каждый пиксель будет иметь число, означающее интенсивность света.

Сначала поговорим о способе представления этих изображений – об изображениях в интегральном представлении.

### Преобразование изображений в интегральное представление

Функция разделена на два этапа. Первый заключается в создании **интегрального изображения** или **таблицы суммированных площадей**, как показано на рис. 13.15.

Изображение									Интегральное изображение								
9	9	5	5	5	5	9	9		9	18	23	28	33	38	47	56	
9	5	1	1	1	1	5	9		18	32	38	44	50	56	70	88	
5	1	0	0	0	0	1	5		23	38	44	50	56	62	77	100	
5	1	7	1	1	7	1	5		28	44	57	64	71	84	100	128	
5	1	1	2	2	1	1	5		33	50	64	73	82	96	113	146	
5	1	1	5	5	1	1	5		38	56	71	85	99	114	132	170	
5	1	3	5	5	3	1	5		43	62	80	99	118	136	155	198	
5	1	1	1	1	1	1	5		48	68	87	107	127	146	166	214	
5	1	5	1	1	5	1	5		53	74	98	119	140	164	185	238	
5	1	1	6	6	1	1	5		58	80	105	132	159	184	206	264	
5	2	1	1	1	1	2	5		63	87	113	141	169	195	219	282	
9	5	2	1	1	2	5	9		72	101	129	158	187	215	244	316	
9	9	5	5	5	5	9	9		81	119	152	186	220	253	291	372	

Рис. 13.15. Интегральное изображение и таблица суммированных площадей

На рис. 13.15 слева показано изображение *улыбающегося лица* с пронумерованными пикселями, представляющими оттенки серого разной интенсивности (чем больше число, тем светлее оттенок). Каждый оттенок имеет номер.

На рис. 13.15 справа показано интегральное изображение. В нем каждый пиксель является суммой или **интегралом** предыдущих пикселей. Каждый пиксель равен сумме значений соответствующих пикселей на исходном изображении, расположенных слева и сверху от него. Координата 2,2 (обведена) является последней в сетке 3 на 3. Ячейка здесь имеет значение 44, что равняется сумме пикселей в выделенном поле ( $9 + 9 + 5 + 9 + 5 + 1 + 5 + 1 + 0$ ).

При суммировании значений пикселей в коде интегральный процесс может использовать ссылку и получить доступ к предыдущим суммам. Новая сумма является результатом сложения пикселей слева и сверху. Например, для координаты 8,8 (обведена), которая находится намного ниже, мы также могли бы сложить все числа, но намного проще использовать уже имеющиеся результаты. Для этого берем значение пикселя (1), добавляем значения ячейки, расположенной выше (166), и ячейки, расположенной слева (164). В эту сумму значения промежуточные пиксели включены дважды, поэтому нам нужно их вычесть. Отнимаем значение ячейки, расположенной сверху слева (146). В результате мы получаем следующую сумму:  $1 + 164 + 166 - 146 = 185$ . Компьютер справляется с такими вычислениями довольно быстро.

В результате мы получаем массив чисел, равный по размеру исходному изображению. Каждая координата представляет собой сумму интенсивности цвета всех пикселей между 0,0 и текущей координатой.

Код может использовать интегральное изображение для быстрого поиска суммы интенсивностей пикселей любого прямоугольника любого размера. Вы можете начать с пикселя, расположенного снизу справа, затем из его значения вычесть значение верхнего правого. В результате останется сумма значений пикселей, находящихся ниже верхнего правого пикселя. Затем вычитаем значение нижнего левого пикселя. Теперь у нас почти получилась сумма пикселей внутри прямоугольника, но мы дважды удалили секции выше верхнего левого пикселя. Чтобы исправить это, прибавляем значение верхнего левого пикселя:

*площадь\_прямоугольника = нижний\_правый - верхний\_правый - нижний\_левый + верхний\_левый*

Это уравнение работает для прямоугольников как размером 2 на 2, так и 300 на 200. Узнать больше вы можете в статье о методе Виолы–Джонса в разделе «Дополнительные материалы». Хорошая новость заключается в том, что этот код уже входит в классификатор OpenCV. На стадии работы с каскадами с помощью интегральных изображений код выполняет один мощный трюк (о нем мы поговорим далее).

## Сканирование базовых признаков

Следующая часть задачи – сканирование изображения на наличие признаков. Признаки крайне просты (например, поиск отличий между двумя прямоугольниками), поэтому сканирование происходит достаточно быстро. На рис. 13.16 показаны базовые признаки.



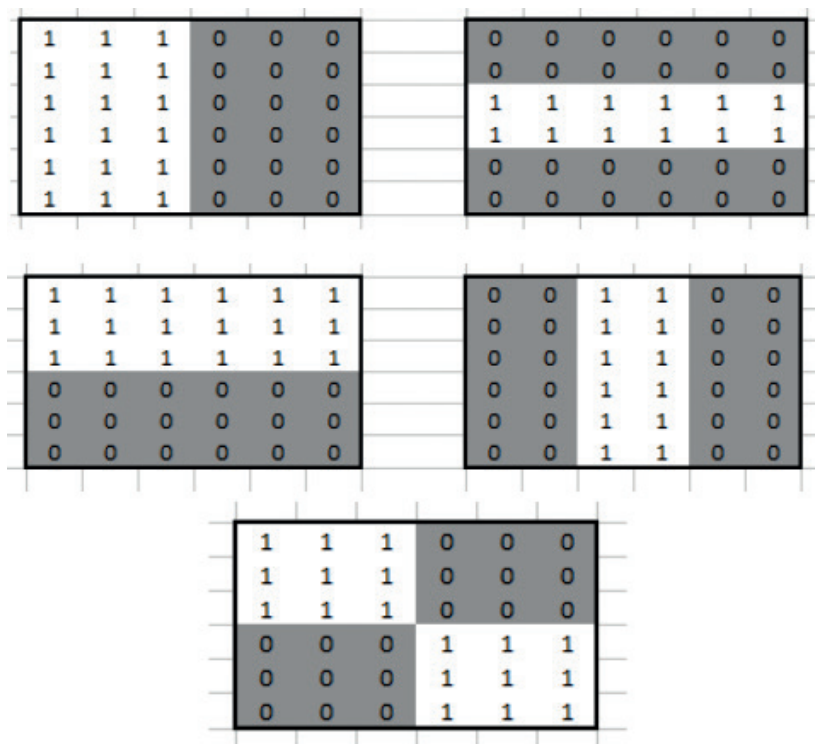


Рис. 13.16. Типы простых прямоугольных признаков

На рис. 13.16 в верхнем левом углу показан признак лево/право, где пиксели слева имеют значение **1**, а справа **0** (они затемнены). Это признак вертикального контраста. В правой верхней части рисунка показан признак горизонтальной полосы, где сначала идет два ряда пикселей со значением **0**, далее два ряда пикселей со значением **1**, а затем снова два ряда пикселей со значением **0**. В признаке, показанном на рисунке в середине слева, сверху идет три ряда пикселей со значением **1**, а ниже три ряда пикселей со значением **0**. Это признак горизонтального контраста. В признаке, показанном на рисунке в середине справа, есть два столбца пикселей со значением **0**, затем два столбца со значением **1** и снова два столбца со значением **0**. Это признак вертикальной полосы.

В признаке, показанном на рисунке снизу, есть две строки по три ряда. В первой строке сначала идет три столбца пикселей со значением **1**, а затем три столбца пикселей со значением **0**. Во второй строке сначала идет три столбца пикселей со значением **0**, а затем три со значением **1**. В результате получается рисунок, похожий на шахматную доску – это признак диагонального контраста.

Алгоритм применяет к изображению признаки (такие как на рис. 13.16) последовательно и в определенном порядке, а затем каждое совпадение *каскадом* переходит к следующей попытке сопоставить другой признак. Существуют файлы, в которых содержится описание объектов как набора признаков. Есть

каскады для обнаружения лиц, включающие в себя 16 000 признаков. Применение к изображению каждого признака заняло бы слишком много времени, поэтому они применяются группами, начиная с одного. Если признак из одной группы не подошел к какой-то части изображения, с ней не будут сопоставляться другие признаки этой группы. Вместо этого к ней применяются признаки из другой группы. Группы включают взвешивание и применение признаков под разными углами.

Если к части изображения подходят все признаки, то она считается совпадающей. Далее нужно найти каскад признаков, который будет идентифицировать объект. К счастью, в OpenCV есть файл, предназначенный для распознавания лиц, и мы уже установили его на Raspberry Pi.

Вся операция применения суммированной области, а затем использования каскадного файла для поиска потенциальных совпадений состоит из двух функций OpenCV:

- `cv2.CascadeClassifier(cascade_filename)` открывает каскадный файл, в котором описаны признаки для проверки. Файл нужно загрузить только один раз, после чего можно использовать его для всех кадров. Это конструктор, который возвращает объект `CascadeClassifier`;
- `CascadeClassifier.detectMultiScale(image)` применяет проверку классификатора к изображению.

Теперь вы разобрались в основах самой распространенной техники распознавания лиц (и объектов). Давайте перейдем к обсуждению плана поведенческого сценария отслеживания лиц. В нем мы объединим технику обработки видеоданных с помощью каскадного классификатора и уже разработанные нами сценарии.

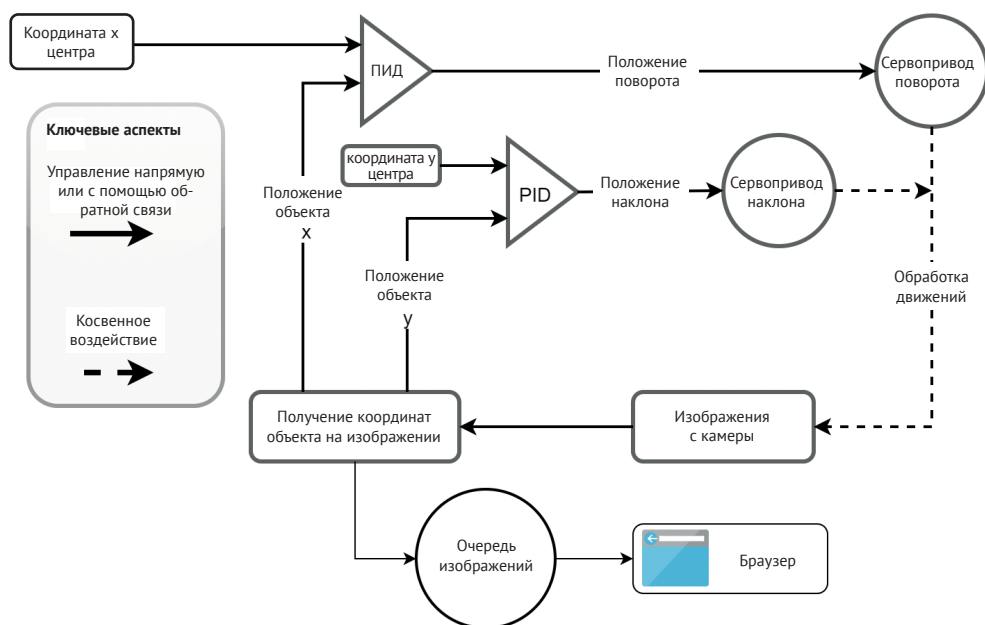
## План поведенческого сценария отслеживания лиц

Для отслеживания лиц можем использовать код, похожий на поведенческий скрипт отслеживания цветных объектов. Мы настроим нашего робота так, чтобы, используя механизмы поворота и наклона, он следовал за самым большим лицом, которое видит камера. На рис. 13.17 показана блок-схема сценария.

Поток данных на рис. 13.17 похож на то, что вы видели ранее. Изображения поступают с той же камеры и попадают в уже знакомую очередь. На этот раз в процессе обработки видеоданных основным является шаг **получения координат объекта на изображении**, который выводит координаты объекта –  $x$  и  $y$ . Чтобы получить положение поворота (которое затем используется сервоприводом поворота), мы передаем значение  $x$  на ПИД-регулятор с центром  $x$ . Чтобы получить положение наклона (для сервопривода наклона) мы передаем координату  $y$  на ПИД-регулятор с центром  $y$ . Сервоприводы приводят камеру в нужное положение, создавая петлю обратной связи для отслеживания перемещений изображения.

Различия заключаются в том, что в этом сценарии мы передаем ПИД-регуляторам другие данные, и в том, что каждый ПИД-регулятор управляет отдельным сервоприводом.

Наш план готов, можем перейти к созданию кода.



## Создание кода для сценария отслеживания лиц

Код для этого сценария покажется вам знакомым, поскольку, в сущности, он представляет собой адаптированный код, созданный ранее. Возможно, с помощью рефакторинга можно получить более обобщенный код, но пока проще работать с копией. Мы поместим новый код в файл `face_track_behaviour.py`. Нам даже не нужно создавать новый шаблон, поскольку подойдет старый (шаблон кода для отслеживания цветных объектов).

1. Сначала выполним почти такие же импорты, как для `color_track_behavior`:

```
import time
```

```
from image_app_core import start_server_process, get_
control_instruction, put_output_image
```

```
import cv2
import os
```

```
import camera_stream
from pid_controller import PIDController
from robot import Robot
```

2. Функция `init` для класса поведения немного отличается. Она начинается с загрузки каскадов Хаара. В том же каталоге есть много других файлов с каскадами, с помощью которых вы можете попробовать отслеживать другие объекты. Для проверки наличия файла в каталоге используется

оператор `assert`. Если OpenCV не найдет файл, то она вернет зашифрованные ошибки в `detectMultiscale`:

```
class FaceTrackBehavior:
    def __init__(self, robot):
        self.robot = robot
        cascade_path = "/usr/local/lib/python3.7/dist-
packages/cv2/data/haarcascade_frontalface_default.xml"
        assert os.path.exists(cascade_path), f"File
{cascade_path} not found"
        self.cascade = cv2.CascadeClassifier(cascade_
path)
```

3. Параметры настройки включают центральное положение и минимальный размер лица. Также я вынес ПИД-регуляторы в отдельный класс, поэтому их значения можно настроить здесь, а сбросить в обработчике управления (можете добавить сброс значений и в предыдущий скрипт):

```
self.center_x = 160
self.center_y = 120
self.min_size = 20
self.pan_pid = PIDController(proportional_
constant=0.1, integral_constant=0.03)
self.tilt_pid = PIDController(proportional_
constant=-0.1, integral_constant=-0.03)
```

4. Конструктор по-прежнему отслеживает, запускает ли этот скрипт двигателя:

```
self.running = False
```

5. Код управления процессом здесь отличается. При получении инструкции `stop` он останавливает двигатели и сбрасывает значения ПИД-регуляторов:

```
def process_control(self):
    instruction = get_control_instruction()
    if instruction:
        command = instruction['command']
        if command == "start":
            self.running = True
        elif command == "stop":
            self.running = False
            self.pan_pid.reset()
            self.tilt_pid.reset()
            self.robot.servos.stop_all()
        elif command == "exit":
            print("Stopping")
            exit()
```

6. В этом скрипте по-прежнему есть метод `find_object`, который принимает исходное изображение. Сначала, чтобы сократить объем данных для поиска, цвета изображения преобразуются в оттенки серого:

```
def find_object(self, original_frame):
    gray_img = cv2.cvtColor(original_frame, cv2.
COLOR_BGR2GRAY)
```

7. Затем мы используем изображение в оттенках серого с каскадным методом `detectMultiScale` и получаем список совпадений:

```
objects = self.cascade.detectMultiScale(gray_img)
```

Метод `detectMultiScale` создает интегральное изображение и применяет алгоритм каскадов Хаара. Он вернет несколько объектов в виде прямоугольников со значениями `x`, `y`, ширины и высоты.

8. Чтобы найти прямоугольник с большей площадью, мы можем использовать цикл, аналогичный тому, что используется в скрипте отслеживания цвета. Сначала нужно настроить хранилище в структуре данных, содержащей площадь, для текущего самого большого прямоугольника, а затем настроить подсписок, содержащий значения `x`, `y`, ширины и высоты:

```
largest = 0, (0, 0, 0, 0)  
for (x, y, w, h) in objects:  
    item_area = w * h  
    if item_area > largest[0]:  
        largest = item_area, (x, y, w, h)
```

9. Возвращаем положение и размеры самого большого прямоугольника:

```
return largest[1]
```

10. Здесь метод `make_display`, ввиду наличия только одного изображения, проще, чем в предыдущем поведенческом скрипте. Однако он по-прежнему кодирует изображение:

```
def make_display(self, display_frame):  
    encoded_bytes = camera_stream.get_encoded_bytes_  
for_frame(display_frame)  
    put_output_image(encoded_bytes)
```

11. Метод `process_frame` находит объект, а затем рисует прямоугольник на кадре для вывода. Функция `cv2.rectangle` принимает координаты двух точек: начальной `x, y` и конечной `x, y`, а также значение цвета. Чтобы получить конечные координаты, нам нужно добавить ширину и высоту:

```
def process_frame(self, frame):  
    x, y, w, h = self.find_object(frame)  
    cv2.rectangle(frame, (x, y), (x + w, y + w),  
[255, 0, 0])  
    self.make_display(frame)  
    return x, y, w, h
```

12. Следующая функция – `run`. Сначала настраиваем камеру и выдерживаем паузу для ее запуска:

```
def run(self):  
    camera = camera_stream.setup_camera()  
    time.sleep(0.1)  
    print("Setup Complete")
```

13. Как и в скрипте отслеживания цветных объектов, мы начинаем главный цикл с обработки кадра и проверки управляющих команд:

```
for frame in camera_stream.start_stream(camera):
    (x, y, w, h) = self.process_frame(frame)
    self.process_control()
```

14. Нам необходимо, чтобы движение начиналось только в том случае, если робот запущен и обнаружил достаточно большой объект (размер определяется по высоте, поскольку в этом измерении лица обычно больше):

```
if self.running and h > self.min_size:
```

15. Когда мы знаем, что робот запущен, мы отправляем выходные значения ПИД-регуляторов на сервоприводы поворота и наклона. Обратите внимание: чтобы найти середину объекта, мы берем координату и добавляем к ней половину ширины или высоты этого объекта:

```
pan_error = self.center_x - (x + (w / 2))
pan_value = self.pan_pid.get_value(pan_
error)
self.robot.set_pan(int(pan_value))
tilt_error = self.center_y - (y + (h / 2))
tilt_value = self.tilt_pid.get_
value(tilt_error)
self.robot.set_tilt(int(tilt_value))
```

16. Чтобы посмотреть, что здесь происходит, используем отладочный оператор print:

```
print(f"x: {x}, y: {y}, pan_error:
{pan_error}, tilt_error: {tilt_error}, pan_value: {pan_
value:.2f}, tilt_value: {tilt_value:.2f}")
```

17. Наконец, нам нужно добавить код для настройки и запуска поведенческого скрипта. Здесь мы по-прежнему используем старый шаблон:

```
print("Setting up")
behavior = FaceTrackBehavior(Robot())
process = start_server_process('color_track_behavior.
html')
try:
    behavior.run()
finally:
    process.terminate()
```

Когда код готов, включая функции настройки, мы можем протестировать его и посмотреть, как работает поведенческий скрипт.

## Запуск поведенческого скрипта отслеживания лиц

Для запуска у вас должны быть загружены файлы сценария отслеживания цветных объектов.

1. Загрузите файл `face_track_behavior.py`.
2. Запустите отслеживание командой `$ python3 face_track_behavior.py`.
3. Введите в браузере <http://myrobot.local:5001>. Вы должны увидеть один кадр с прямоугольным контуром вокруг самого большого лица.
4. Чтобы заставить робота двигаться, нажмите кнопку **Start**.

Сервоприводы механизма поворота и наклона начнут двигаться, чтобы попытаться поместить ваше лицо в центр кадра. Это будет означать, что камера направлена прямо на вас. Если вы начнете двигать головой, камера будет (медленно) следовать за вашими движениями. Если позади вас будет стоять другой человек, робот не увидит его, но, если вы закроете рукой половину своего лица, он перестанет узнавать вас и повернется другому человеку.

## Устранение неполадок

Начните с шагов по устранению неполадок для предыдущего сценария – в большинстве случаев это помогает. В ином случае обратитесь к советам ниже:

- если приложению не удастся найти файл каскада Хаара, проверьте расположение файлов. У них менялось расположение в разных версиях OpenCV, и это может произойти снова. Проверьте код на наличие опечаток. Если их нет, то попробуйте следующую команду:

```
$ find /usr/ -iname "haarcas*"
```

Она покажет расположение файлов на Raspberry Pi;

- если камера не может обнаружить лица на снимке, скорее всего, ей не хватает света;
- алгоритм предназначен только для обнаружения лиц, которые смотрят в прямо в камеру. Если закрыть лицо или его часть, алгоритм не воспримет его. Его могут сбить даже очки или шляпа;
- лица, попавшие в кадр лишь частично, скорее всего, будут проигнорированы. Слишком маленькие лица или лица, которые находятся слишком далеко, отфильтровываются. Уменьшение минимального параметра позволяет захватывать больше объектов, но при этом возникает риск ложных срабатываний, поскольку алгоритм может реагировать на мелкие объекты, похожие на лица;
- проверьте, чтобы отступы в коде совпадали (в Python это важно).

Итак, вы создали код, который будет обнаруживать и отслеживать лица, находящиеся в поле зрения камеры. Без сомнения, выполнение роботом такого сценария будет выглядеть впечатляюще. Подведем итог тому, что вы узнали в этой главе.

## Выводы

В этой главе вы узнали, как настроить модуль камеры Raspberry Pi. Затем научились видеть с его помощью то, что видит робот (взгляд робота на мир).

Вы создали веб-приложение (доступное в веб-браузере как на смартфоне, так и на компьютере), в котором отображаются данные камеры. Затем вы разработали скрипты для отслеживания с помощью камеры объектов определенных цветов и обнаружения лиц. Вы узнали, как улучшить эти сценарии, и получили общее представление о том, как работает компьютерное зрение.

В следующей главе мы расширим возможности обработки видеоданных и научим робота следовать по линиям, основываясь на данных камеры. Также поговорим о некоторых других способах ее использования.



## ЗАДАНИЕ

Созданный нами код весьма занятный, но мы можем расширить его возможности. Я предлагаю несколько заданий, которые позволят улучшить ваш код и углубиться в тему:

- используйте конвейер управления, чтобы дать пользователю возможность настраивать цветовые фильтры, правильный радиус и значения ПИД-регуляторов на веб-странице. Может быть, начальные значения ПИД-регуляторов должны быть близки к другим настраиваемым значениям?
- код программы настройки весьма обширный. Можно ли поместить его в функцию/метод?
- можно ли использовать очереди для отправки отладочных данных на веб-страницу, вместо того чтобы выводить их на печать в консоли? Возможно ли представить данные в виде графика?
- поле зрения камеры Pi довольно узкое. Широкоугольный объектив значительно расширит его и позволит роботу видеть больше;
- камера плохо работает в темноте. У робота есть светодиодная линейка, но она дает мало света. Попробуйте добавить еще один светодиод, который будет выступать как фонарь подсветки для камеры;
- попробуйте отслеживать разные объекты, используя другие каскадные файлы из папки `/usr/share/opencv/haarcascades` на Raspberry Pi;
- попробуйте поменять местами признаки из двух сценариев, чтобы использовать серводвигатели для отслеживания цветных объектов и лиц;
- объедините механизм поворота и наклона с ведущими колесами, чтобы робот мог ездить за отслеживаемым лицом, при этом центрируя поворот и удерживая объект в поле зрения. Задействуйте свое творческое мышление и подумайте, какие значения для ПИД-регуляторов подойдут.

Благодаря этим идеям вы сможете лучше разобраться в этих типах обработки видеоданных.

## ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

Обработка видеоданных – это обширная тема. Ниже представлена лишь малая часть источников, в которых вы можете узнать больше об использовании камеры для подобных задач:

- официальное руководство для камеры Raspberry Pi вы можете найти по адресу <https://magpi.raspberrypi.com/books/camera-guide>. Отличный ресурс для знакомства с камерой. Также здесь содержится множество примеров практических проектов;
- чтобы изучить использование камеры Raspberry Pi подробнее, я рекомендую обратиться к документации PiCamera, доступной по адресу <https://picamera.readthedocs.io/>;
- узнать больше о других методах отслеживания объектов вы можете на веб-сайте PyImageSearch по адресу <https://www.pyimagesearch.com/>;

- OpenCV и обработка видеоданных – это сложная тема, которая в этой книге освещена кратко. Я рекомендую прочесть книгу «*OpenCV 3 Computer Vision with Python Cookbook*» авторов Алексея Спизевого и Александра Рыбникова от издательства *Packt Publishing*. Больше информации о ней вы можете найти по адресу <https://www.packtpub.com/product/opencv-3-computer-vision-with-python-cookbook/9781788474443>;
- потоковая передача видео через Flask – это ловкий трюк. Более подробно он рассматривается в статье «*Video Streaming with Flask*», доступной по адресу <https://blog.miguelgrinberg.com/post/video-streaming-with-flask>;
- чтобы узнать больше об удобных способах использования Flask для управления вашим роботом со смартфона или ноутбука, обратитесь к <https://flaskbook.com>;
- узнать больше о настройке ПИД-регуляторов, о которой мы говорили мы в главе 11 и в этой главе, можно в статье «*Robots For Robotists | PID Control*» по адресу <http://robotsforrobotists.com/pid-control/>. В ней много математики, но зато есть отличный раздел о ручной настройке ПИД-регулятора;
- в статье 2001 года Пола Виолы (Paul Viola) и Майкла Джонса (Michael Jones) под названием «*Rapid Object Detection Using a Boosted Cascade of Simple Features*», которую вы можете найти по адресу <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>, подробно описывается используемая нами техника обнаружения объектов с помощью каскадов Хаара;
- в видеоролике об обнаружении лиц под названием «*Detecting Faces (Viola Jones Algorithm)*», который вы можете найти на YouTube по адресу <https://www.youtube.com/watch?v=uEJ71VlUmMQ>, подробно рассказывается о комбинации используемых техник;
- документация OpenCV по каскадной классификации доступна по адресу [https://docs.opencv.org/2.4/modules/objdetect/doc/cascade\\_classification.html](https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html). Здесь вы можете найти информацию о библиотечных функциях, используемых в поведенческих сценариях отслеживания лиц;
- по адресу [https://docs.opencv.org/3.3.0/d7/d8b/tutorial\\_py\\_face\\_detection.html](https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html) вы можете найти официальное руководство по отслеживанию лиц (для версии OpenCV 3.0) под названием «*OpenCV: Face Detection using Haar Cascades*».

# Глава 14

## Код на Python

### для отслеживания

### линий с помощью камеры

В предыдущей главе вы узнали, как использовать камеру для отслеживания объектов. В этой главе мы расширим код для камеры и разработаем поведенческий скрипт для обнаружения линий.

Вы узнаете, где могут применяться такие скрипты и почему они полезны. Также мы поговорим о некоторых других подходах, позволяющих различным роботам двигаться по заданному пути, и рассмотрим их плюсы и минусы. Затем вы узнаете, как создать простой маршрут с линией, по которой будет следовать робот.

Мы рассмотрим несколько алгоритмов и выберем самый простой. Затем создадим диаграмму потока данных и посмотрим, как работает этот алгоритм. Далее проведем тест производительности на основе образцов изображений. Параллельно мы поговорим о других подходах в компьютерном зрении и способах извлечения данных.

Мы расширим код для ПИД-регуляторов, добавим в наш скрипт алгоритм отслеживания линий, а затем посмотрим, как робот выполняет его. В завершение главы я представлю несколько идей для дальнейшей работы в этом направлении.

В этой главе мы рассмотрим следующие темы:

- введение в отслеживание линий;
- создание тестового маршрута;
- конвейерную обработку данных компьютерного зрения для отслеживания линий;
- тест алгоритма компьютерного зрения на основе образцов изображений;
- отслеживание линий с помощью ПИД-алгоритма;
- нахождение линии при ее потере.

## ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам потребуются:

- робот и код из главы 13;
- белая или черная изолента;
- листы бумаги или картона формата A2 (цвета контрастного к цвету изоленты);
- ножницы;
- хорошее освещение.

Полный код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter14>. Посмотреть видеоролик «Code in Action» на YouTube можно по адресу <https://bit.ly/3sLLzbQ>.

## ВВЕДЕНИЕ В ОТСЛЕЖИВАНИЕ ЛИНИЙ

Прежде чем перейти к созданию кода, необходимо узнать больше о том, как работает отслеживание линий. В этом разделе вы узнаете, где и как используются такие системы, а также какие методы применяются для их реализации.

### Что такое отслеживание линий?

Некоторым роботам для выполнения задач требуется перемещаться строго по определенному пути. В таком случае роботу проще ориентироваться по линии, чем создавать карты целых комнат или зданий.

Говоря проще, отслеживание линий позволяет роботу автономно следовать по размеченному пути. Маркеры пути могут быть визуальными, например синяя лента на полу или белая полоса на темной дороге. Во время движения робот отслеживает участок линии впереди, корректируя свой курс.

На соревнованиях для роботов одним из типичных испытаний является гонка по линиям, где скорость важнее точности.

### Использование в промышленности

Наиболее широко отслеживание линий применяется в промышленности. **Автоматизированные транспортные средства (AGV – Automated Guided Vehicles)** выполняют ряд задач, для которых требуется строгое следование заданному маршруту, например некоторые из них должны двигаться между рядами упакованных продуктов. Роботы на заводах должны перемещаться только по свободным маршрутам вне рабочих зон. Линиями может быть размечен путь от стеллажа до погрузочной площадки или от зарядной станции робота до его рабочей зоны.

На рис. 14.1 показан робот IntelliCart, который следует по пути, размеченному синей лентой. Однако в промышленности чаще используют магнитные дорожки, расположенные под полом.



**Рис. 14.1.** IntelliCart – промышленный робот, отслеживающий линии, от компании Mukeshhrs (фото в открытом доступе)

Маршрут может включать точки выбора, от которых линии отходят в разные стороны. В зависимости от выполняемой задачи роботу могут потребоваться подсказки, чтобы понять, что он находится в такой точке. Для полностью автоматизированных систем инженеры могут задавать повторяющиеся маршруты.

С помощью таких маршрутов устанавливаются границы безопасности и определяются зоны, где люди и роботы могут взаимодействовать, а где нет. Так роботы не будут работать за пределами хорошо знакомых им зон.

## Типы алгоритмов отслеживания линий

Существует несколько основных типов алгоритмов отслеживания линий и связанных с ними систем.

Сегодня наиболее распространенной и простой системой является **следование по визуальным линиям**. В ней робот ориентируется по нарисованным или размеченным лентой линиям. Такую систему очень легко реализовать, но на точность могут влиять загрязнения на поверхности и условия освещения. Способы обнаружения линий делятся на две категории:

- **обнаружение с помощью оптических датчиков.** В этом случае к нижней части робота (ближе к предполагаемому месту размещения линии) прикрепляются небольшие датчики. Они выводят двоичный прерывающийся или аналоговый сигнал. Для освещения поверхности такие датчики, как правило, оснащаются собственным небольшим источником света. Такие датчики компактные и недорогие, но для их подключения потребуется задействовать дополнительные контакты ввода/вывода;

- **обнаружение с помощью камеры.** Если у робота уже есть камера, такой подход сэкономит место и избавит от сложностей установки и подключения, поскольку он не задействует дополнительные контакты ввода/вывода. Однако для обработки данных камеры роботу потребуются алгоритмы компьютерного зрения и сложное программное обеспечение.

**Следование по магнитным линиям** используется в случаях, когда важно, чтобы на линию не влияли окружающие условия. Некоторые варианты таких систем позволяют направлять робота по нескольким маршрутам. Существуют следующие варианты:

- по полу прокладывается магнитная лента, которую робот обнаруживает посредством датчиков Холла (см. главу 12). Робот оснащается рядом таких датчиков, благодаря чему может определять направление линии и следовать ему. Такие маршруты корректировать легче, чем нарисованные или размеченные лентой, но об магнитную ленту можно споткнуться;
- тот же эффект можно получить, проложив по полу (или под ним) провод, по которому будет идти ток. Используя различные схемы с множеством проводов, можно направлять робота по разным маршрутам;
- если разместить магнитную дорожку под полом, то об нее никто не споткнется. Однако в таком случае для людей нужно обеспечить предупреждающие надписи, чтобы они не заходили в рабочую зону роботов.

Итак, вы познакомились с двумя основными типами алгоритмов отслеживания линий. Теперь поговорим о других практических способах определения пути роботом.

- **Маяки.** Робот может ориентироваться по ультразвуковым, светоизлучающим или радиоизлучающим маякам, установленным вокруг. Они могут выступать в качестве отражателей лазерного или другого излучения.
- **Визуальные символы.** На стенах или столбах можно разместить QR-коды или другие визуальные маркеры, содержащие данные о точном положении.

Теперь вы знаете, как робот может следовать по видимым или скрытым линиям (например, провода и магнитные датчики). Мы будем использовать простой подход – видимые линии.

Для подключения простых оптических датчиков потребуется внести изменения в электрическую схему робота, но если у нас уже есть камера, то почему бы не использовать ее?

В этой главе мы реализуем систему, в которой робот будет отслеживать видимую линию с помощью камеры. Код будет более сложным, но аппаратная часть значительно упростится.

Теперь, когда вы получили представление о различных типах алгоритмов отслеживания линий и о том, где они применяются, давайте создадим тестовый маршрут, по которому будет следовать наш робот.

## Создание тестового маршрута

Начнем с изготовления одного участка линии, по которой будет следовать робот. Этот маршрут мы будем использовать для проверки алгоритма обнаруже-



ния линии. Позже мы усложним маршрут и заставим робота проехать по нему. Маршрут, который мы создадим в этом разделе, весьма прост, и его можно легко расширить или изменить, что позволяет экспериментировать с формами линий и смотреть, как робот справляется с их отслеживанием.

Также вы можете поэкспериментировать с различными цветами и контрастностью линий и поверхности.

## Материалы для создания тестового маршрута

На рис. 14.2 показаны необходимые материалы.



**Рис. 14.2.** Материалы, необходимые для создания тестового маршрута

Здесь мы видим черную изоленту и большой лист белой бумаги. В этом разделе вам понадобятся:

- чистые листы белой бумаги или картона формата A2;
- непрозрачная черная изолента или малярный скотч;
- ножницы.

Вместо бумаги можно использовать доски или панели, окрашенные в белый цвет.

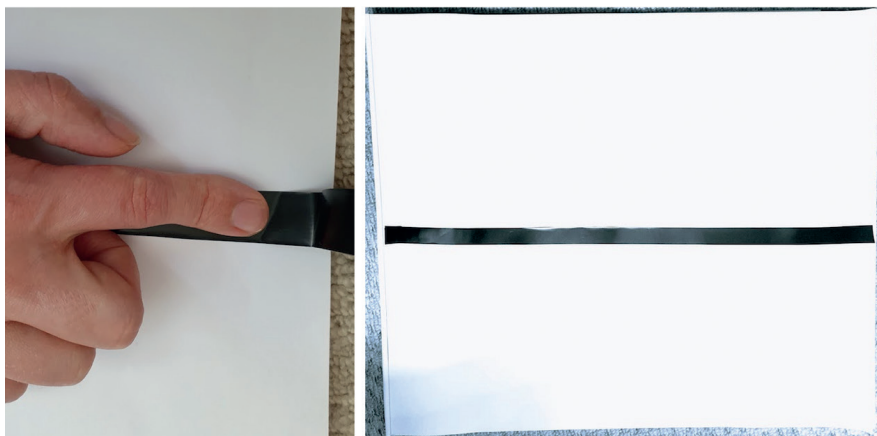
Можно использовать темную или черную бумагу и белую ленту. Главное, чтобы лента была непрозрачной и контрастировала с фоном.

## Создание линии

Расстелите лист бумаги и приклейте посередине (по горизонтали) полоску ленты.

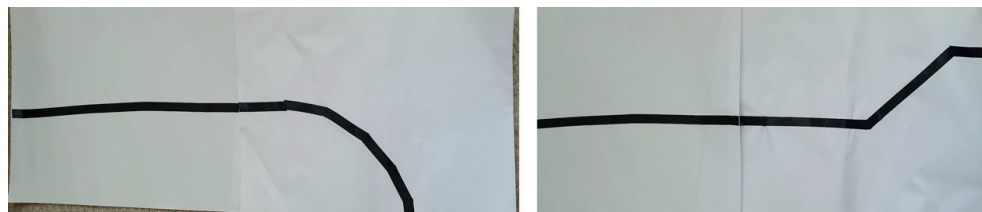
На рис. 14.3 показана бумага с полоской ленты, которую я разглаживаю пальцем. Разгладить ее нужно обязательно. Ничего страшного, если она будет наклеена не идеально прямолинейно, поскольку весь смысл этой системы заключается в том, чтобы робот мог отслеживать любые линии.





**Рис. 14.3.** Разглаживаем ленту на бумаге

На другом листе формируем изогнутую линию, как показано на рис. 14.4. Она будет примыкать к первой.



**Рис. 14.14.** Линии различных форм, которые будут примыкать к основной

Как показано на рис. 14.4, формы линий могут быть любыми. Вы можете соединить их прямыми линиями и сделать целые секции, как на игрушечной железной дороге. Позже, на этапе разработки кода, с этим будет весьма интересно экспериментировать.

Теперь, когда тестовый маршрут готов, перейдем к теме обнаружения линий с помощью видеоустройства.

## **Конвейерная обработка данных компьютерного зрения для следования по линиям**

Как и в предыдущих задачах компьютерного зрения, мы будем визуализировать данные в виде конвейера. Прежде чем мы это сделаем, необходимо обсудить существующие методы отслеживания линий посредством компьютерного зрения.

### **Алгоритмы отслеживания линий с помощью камеры**

Нам нужно выбрать один из самых простых алгоритмов, но, разумеется, существуют и альтернативные варианты, которые позволяют роботу принимать

более сложные решения и лучше предугадывать повороты при следовании по изогнутым линиям.

Вот несколько методов, которые мы могли бы использовать:

- **обнаружение границ.** Алгоритм обнаружения границ, например детектор границ Кэнни, обнаруживает резкие изменения (переходы) на изображении и распознает их как границы линии. OpenCV имеет встроенную систему обнаружения границ. Система может обнаруживать переходы от темного к светлому и наоборот. Она неплохо справляется с обнаружением границ с менее резкими переходами;
- **вычисление разностей значений пикселей.** Этот алгоритм похож на обнаружение границ, но он выполняется только на определенной строке пикселей изображения. На границах пиксели будут резко различаться, поэтому алгоритм ищет разницу между их значениями в каждой строке. Такой подход проще и экономичнее в вычислительном отношении, чем алгоритм Кэнни. Он может обнаружить границы, идущие в любом направлении, но требует, чтобы они сильно контрастировали с фоном;
- **поиск перепадов яркости и обнаружение линии как области абсолютной яркости.** Такой подход очень простой и недорогой, но не всегда дает хорошие результаты. Он плохо справляется с инверсией, но не требует резких контрастов, поскольку не отслеживает границы.

Используя один из трех предыдущих методов, робот может обнаружить линию в какой-либо области изображения и нацелиться на нее. Это означает, что он не сможет предвидеть изменения курса. Такой подход является самым простым. Выбранной областью может быть одна строка в нижней части экрана.

Или же вы можете использовать перечисленные выше методы, чтобы обнаружить линию на изображении с камеры и заблаговременно построить для нее траекторию. Такой подход сложнее, но позволит роботу лучше справляться с крутыми поворотами.

Стоит отметить, что мы могли бы создать более эффективный и сложный алгоритм на основе необработанных данных в формате YUV с камеры Pi. Однако мы остановимся на более простом варианте. В дальнейшем вы можете использовать более быстрые и точные методы.

Еще одним ограничением в нашей системе является узкий угол обзора камеры. Расширить его можно с помощью линз, тогда робот будет терять линию не так часто.

Мы будем использовать метод вычисления разностей, поскольку он достаточно простой и при этом работает с линиями разных цветов. С его помощью мы обработает всего один ряд пикселей, что значительно упростит математическую часть.

## Конвейер обработки изображения

Мы можем представить обработку данных в виде конвейера. Прежде чем сделать это, разберемся, что такое дискретные различия. Яркость каждого пикселя выражается числом от 0 (черный) до 255 (белый). Чтобы вычислить разность, необходимо вычесть значение текущего пикселя из значения пикселя справа от него.

На рис. 14.5 показано шесть комбинаций пикселей разных оттенков.

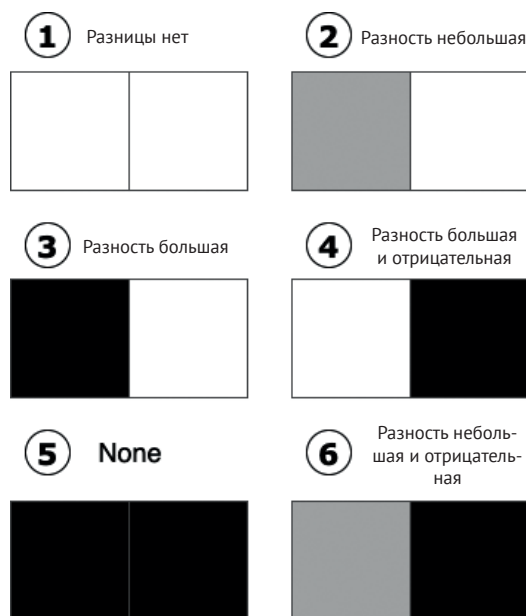


Рис. 14.5. Дискретные различия между пикселями

1. Два белых пикселя. Между ними разницы нет.
2. За серым пикселем следует белый. Значение разности небольшое.
3. За черным пикселем следует белый пиксель. Значение разности большое.
4. За белым пикселем следует черный. Значение разности большое и отрицательное.
5. Два черных пикселя. Между ними разницы нет.
6. За серым пикселем следует черный. Значение разности небольшое и отрицательное.

На контрастной границе линии разность значений пикселей будет самая большая (результат может быть как положительный, так и отрицательный). Наш код будет искать именно такие результаты.

На рис. 14.6 показан процесс обнаружения линии. Все начинается с **камеры**, затем происходит **захват изображения** (разрешение снимка 320 на 240). Затем цвета на изображении **преобразуются в оттенки серого** – нас интересует только их яркость.

На изображении может появиться шум, поэтому нам нужно **снизить его резкость (размытие)**. Этот шаг не строго обязателен и требуется только тогда, когда изображение зашумлено.

Далее алгоритм берет изображение и вырезает из него подходящий ряд пикселей. Ряд должен располагаться не слишком высоко, поскольку тогда он может не захватить линию, а также на этой части снимка могут быть посторонние объекты (в зависимости от положения камеры). Также ряд не должен располагаться слишком низко, поскольку тогда эта область будет

находиться слишком близко к роботу, и он не успеет среагировать. На этой части рисунка в качестве примера показано изображение и вырезанный из него подходящий ряд.

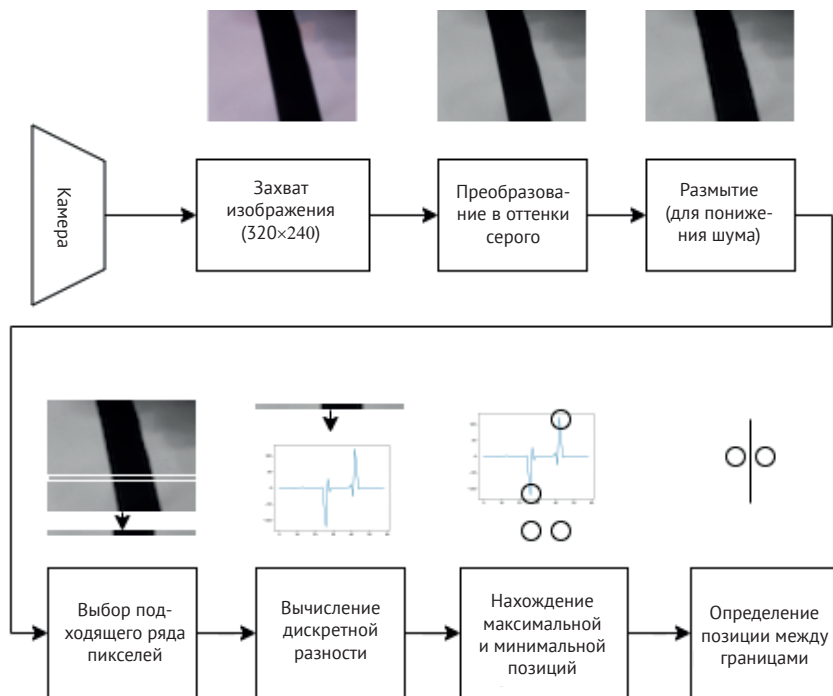


Рис. 14.6. Процесс обработки изображения для обнаружения линии

Затем мы преобразуем строку в набор чисел, **вычисляем дискретную разность между ними**. На этой части рисунка над соответствующим текстовым полем показан график, на котором возникает отрицательный пик, когда за светло-серым пикселем следует черный; когда за черным пикселем следует светло-серый, возникает большой положительный пик. Обратите внимание, что на большей части графика линия проходит ближе к нулю, что свидетельствует об отсутствии разницы между пикселями.

Следующий шаг – **нахождение максимальной и минимальной позиций** и определение, где эти точки находятся в ряду пикселей. Нам нужны позиции/индексы самой высокой (выше 0) и самой низкой (ниже 0) точек. Так мы примерно определяем, где проходят границы линии.

Далее мы **определяем позицию между** этими границами – центр линии. Для этого складываем значения и делим результат на 2. Так мы находим позицию линии по оси координат  $x$  относительно центра изображения с камеры.

Итак, вы узнали, как проходит конвейерная обработка данных. Пришло время сделать несколько снимков, создать тестовый код и проверить, как работает алгоритм.

## ПРОВЕРКА АЛГОРИТМА КОМПЬЮТЕРНОГО ЗРЕНИЯ НА ОСНОВЕ ОБРАЗЦОВ ИЗОБРАЖЕНИЙ

В этом разделе вы узнаете, как и зачем использовать тестовые изображения. Мы создадим первый фрагмент кода для нашего скрипта и опробуем его на тестовых изображениях с камеры нашего робота. С помощью теста мы подготовимся к использованию реального скрипта.

### Зачем нужны тестовые изображения?

До сих пор мы использовали компьютерное зрение только в рамках определенных сценариев. Но что, если мы хотим запустить код обработки видеоданных изолированно?

Возможно, вы захотите посмотреть на код в действии, устранить какие-либо неполадки, попробовать сделать его быстрее и рассчитать время. Для этого нужно запускать конкретный код отдельно от управляющих систем робота.

Здесь целесообразно использовать тестовые изображения. Вместо того чтобы каждый раз запускать камеру и настраивать освещение, вы можете протестировать уже готовые снимки и сравнить их с ожидаемым результатом.

Чтобы протестировать точность и быстродействие, нам нужно использовать одно и то же изображение (или набор изображений) 100 раз и получить согласованные результаты и значимые показатели производительности. Использовать новые данные каждый раз нецелесообразно, поскольку это может привести к неожиданным результатам. Но все же мы можем добавить другие изображения, чтобы посмотреть, как алгоритм работает с ними.

Теперь, когда вы знаете зачем нужны тесты, попробуем сделать несколько тестовых снимков.

### Получение тестовых изображений

В прошлой главе мы делали снимки с помощью команды `raspistill`. Повторим то же самое и в этой главе. Сначала нам нужно привести камеру в другое положение – направить ее вниз на линию.

В этом разделе нам потребуются настройки из главы 13 и код из главы 9.

Включите питание двигателя на Raspberry Pi, затем войдите через `ssh` и выполните следующие шаги.

1. Введите в терминале команду `python 3`:

```
pi@myrobot:~ $ python3
Python 3.7.3 (default, Dec 20 2019, 18:57:59)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

2. Теперь нам нужно импортировать объект `robot` и обеспечить взаимодействие с ним:

```
>>> import robot
```

3. Создаем объект `robot`:

```
>>> r = robot.Robot()
```

4. Настраиваем сервопривод поворота, чтобы камера заняла срединное положение:

```
>>> r.set_pan(0)
```

Сервопривод поворота поместит камеру в центр.

5. Затем настраиваем сервопривод наклона, чтобы камера смотрела вниз на линию:

```
>>> r.set_tilt(90)
```

Сервопривод направит камеру прямо вниз. Он должен двигаться без усилий и рывков.

6. Теперь можно выйти из Python (и отключить двигатели) используя комбинацию клавиш **Ctrl+D**.

Камера направлена вниз. Теперь вы можете выключить двигатели с помощью переключателя и переставить робота на начало тестового маршрута. Постарайтесь расположить робота так, чтобы камера находилась прямо над линией.

7. В терминале ssh введите команду для тестового снимка:

```
$ raspistill -o line1.jpg
```

Теперь вы можете загрузить снимок на свой компьютер с помощью FileZilla (об этом мы говорили в предыдущих главах). На рис. 14.7 показано тестовое изображение, используемое в предыдущих примерах.



Рис. 14.7. Тестовый снимок линии

На рис. 14.7 показано одно из моих тестовых изображений. Обратите внимание, что здесь линия начинается примерно в середине изображения, но затем отклоняется (она и не должна идти идеально ровно). Из-за условий освещения на снимке появляются тени. На них стоит обратить внимание, поскольку они могут запутать систему.

Сделайте несколько снимков под разными углами относительно робота и немного левее или немного правее камеры.

Все готово, можем приступить к созданию тестового кода!

## Создание кода на Python для обнаружения границ линии

Мы готовы приступить к созданию кода, используя тестовые снимки и описанный ранее конвейер. Для большей наглядности мы визуализируем работу алгоритма.

### Совет

Для задач компьютерного зрения лучше использовать самое низкое возможное разрешение. С каждым дополнительным пикселем данных становится все больше, что увеличивает объем необходимой памяти и усложняет обработку. Изображение с разрешением 320 на 200 содержит 76 800 пикселей. Камера Raspberry Pi может делать снимки с разрешением 1920 на 1080, что равняется 2 073 600 пикселям – в 27 раз больше данных! Нам нужно, чтобы алгоритм работал быстро, поэтому выбираем низкое разрешение.

Код в этом разделе будет работать на Raspberry Pi, но вы также можете запустить его на ПК с установленными Python 3, NumPy, Matplotlib и Python OpenCV.

1. Создайте файл с именем `test_line_find.py`.
2. Нам нужно импортировать NumPy для численной обработки изображения, OpenCV для управления изображением и Matplotlib для графического отображения результатов:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

3. Теперь загружаем изображение. OpenCV может загружать изображения в формате jpg, но, если в процессе загрузки возникает проблема, создается пустое изображение. Нам нужно проверить, что изображение загрузилось:

```
image = cv2.imread("line1.jpg")
assert image is not None, "Unable to read file"
```

Я указал, что изображение называется `line1.jpg` и находится в том же каталоге, из которого мы будем запускать этот файл.

4. По умолчанию изображение будет иметь высокое разрешение. Нам необходимо его снизить:

```
resized = cv2.resize(image, (320, 240))
```

5. Нам нужно изображение в оттенках серого, поскольку цвета нас не интересуют:

```
gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
```

6. Теперь выбираем ряд пикселей. Мы будем использовать ряд 180 – это приемлемо для изображения высотой 240 пикселей. Изображения хранятся таким образом, что ряд 0 находится сверху. Обратите внимание, что мы указываем NumPy преобразовать ряд в тип `int32`:



```
row = gray[180].astype(np.int32)
```

Выполняем преобразование в тип `int32` со знаком (плюс или минус), чтобы разности могли быть отрицательными.

7. Далее получаем список разностей для каждого пикселя в ряду. С помощью NumPy сделать это достаточно просто:

```
diff = np.diff(row)
```

8. На основе списка `diff` мы построим график. По оси  $x$  будут указаны номера пикселей. Создадим диапазон NumPy от 0 до текущего количества пикселей:

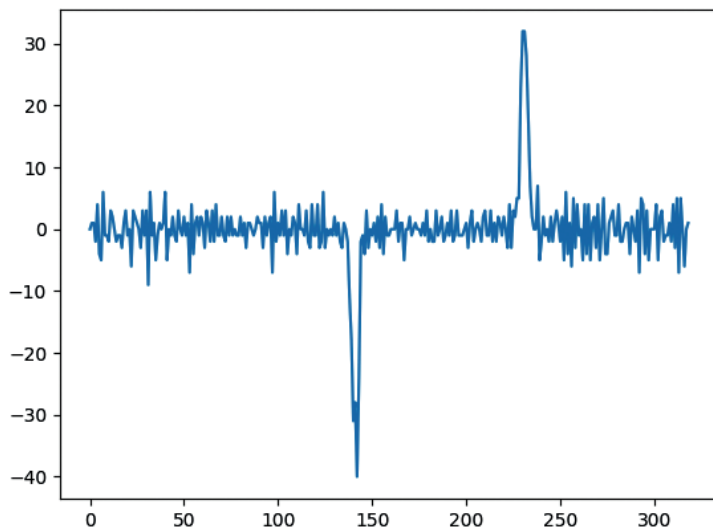
```
x = np.arange(len(diff))
```

9. Создаем график зависимости переменной `diff` и индекса пикселя ( $x$ ) и сохраняем результат:

```
plt.plot(x, diff)
plt.savefig("not_blurred.png")
```

Я назвал этот файл `not_blurred` (без размытия), поскольку у нас отсутствует шаг понижения резкости изображения. Далее на графике мы увидим разницу.

На основе данных моего тестового изображения получается график, показанный на рис. 14.8.



**Рис. 14.8.** График разностей без понижения резкости изображения

На рис. 14.8 показан график, где по оси  $x$  располагаются номера столбцов, а по оси  $y$  значения разности. Линия сильно зашумлена. Мы видим два выделяющихся пика – один ниже нуля около столбца 145, а другой выше нуля около 240. Здесь шум не сильно мешает, поскольку пики и так отчетливо видны.

1. Попробуем добавить размытие и посмотрим, как изменится график. Внесите в код изменения, показанные в следующем фрагменте (выделены жирным шрифтом):

```
gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
blurred = cv2.blur(gray, (5, 5))
row = blurred[180].astype(np.int32)
diff = np.diff(row)
```

В этом коде мы понижаем резкость фрагментов изображения размером 5 на 5.

2. Чтобы мы могли увидеть два разных графика, изменим имя выходного файла:

```
plt.savefig("blurred.png")
```

Небольшое размытие должно снизить шум, не повлияв при этом на выделяющиеся пики. График показан на рис. 14.9.

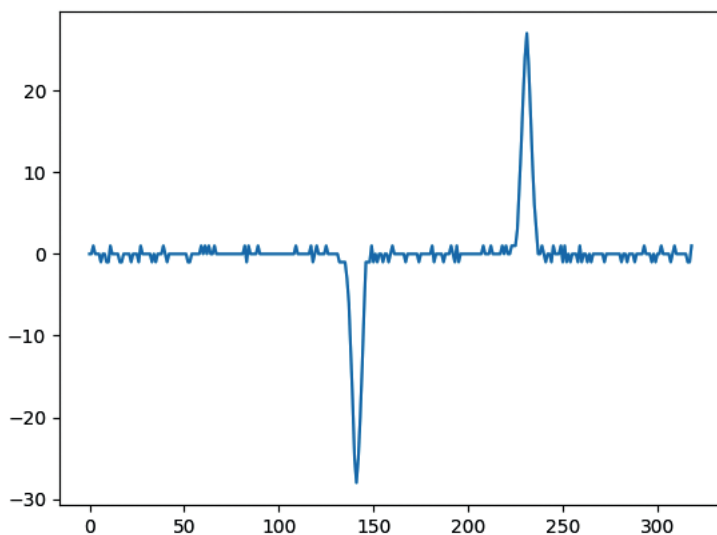


Рис. 14.9. График разностей после размытия

График на рис. 14.9 аналогичен графику на рис. 14.8 – он имеет те же оси и пики. Однако на линии гораздо меньше шума – это говорит о том, что уменьшение резкости делает разницу более четкой. Влияет ли это на результат? Глядя на положение и размер пиков, становится понятно, что не так сильно. Поэтому мы можем исключить этот шаг из финального скрипта и за счет этого немного увеличим скорость алгоритма. Каждая операция будет выполняться быстро.

Теперь, когда у нас есть два пика, найдем с их помощью положение линии.

## Определение положения линии по границам

Пики являются маркерами границ нашей линии. Чтобы найти середину, мы всегда складываем левую и правую координаты, а затем делим результат на 2.

1. Сначала мы должны определить координаты. Создадим код, который будет запрашивать максимальное и минимальное положение. Мы поместим его между кодом анализа изображения и кодом выдачи графика:

```
diff = np.diff(row)
max_d = np.amax(diff, 0)
min_d = np.amin(diff, 0)
```

Код находит максимальное и минимальное значения массива. Я назвал их `min_d` и `max_d`, где `d` – это разность. Обратите внимание, что их нельзя назвать `min` и `max`, так как эти имена уже принадлежат функциям в Python.

2. Мы нашли значения, но не положения. Теперь нам нужно найти индекс положений. Для получения индексов из массивов в NumPy существует функция `np.where`:

```
highest = np.where(diff == max_d)[0][0]
lowest = np.where(diff == min_d)[0][0]
```

Функция NumPy `where` возвращает массив ответов для каждого измерения – поэтому, хотя `diff` и является одномерным массивом, мы все равно получим несколько списков. Первый `[0]` выбирает список результатов первого измерения, а второй `[0]` выбирает первый элемент в результатах. Наличие нескольких результатов свидетельствует о том, что было найдено более одного пика, но мы предполагаем, что на данный момент есть только один.

3. Чтобы найти середину, нам нужно сложить значения и разделить на 2:

```
middle = (highest + lowest) // 2
```

4. Теперь мы должны отобразить результат. Можно сделать это на графике, построив три линии. С помощью Matplotlib мы можем указать цвет и стиль графика. Начнем со средней линии:

```
plt.plot([middle, middle], [max_d, min_d], "r-")
```

Matplotlib ожидает серии данных, поэтому линия определяется как пара координат  $X$  и пара координат  $Y$ . В качестве координат  $Y$  мы используем `max_d` и `min_d`, поэтому линия проводится от самого высокого пика к самому низкому. Спецификатор стиля `r-` сообщает, что линия должна быть непрерывной и иметь красный цвет.

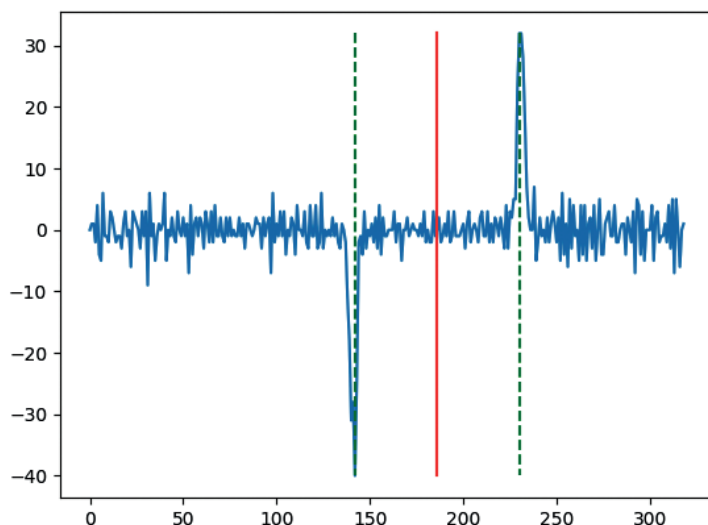
5. Делаем то же самое для положений `highest` и `lowest`. Здесь строим пунктирную зеленую линию с помощью спецификатора `g--`:

```
plt.plot([lowest, lowest], [max_d, min_d], "g--")
plt.plot([highest, highest], [max_d, min_d], "g--")
```

6. Когда мы строили график по изображению с размытием, мы поменяли имя выходного графика. Здесь делаем то же самое:

```
plt.savefig("located_lines.png")
```

При запуске кода мы получаем график, как на рис. 14.10.



**Рис. 14.10.** График с линией для минимального значения, линией для максимального значения и средней линией

График на рис. 14.10 показывает, что мы нашли среднюю линию и два четких пика. Код выглядит пригодным для нашего робота.

Но что бывает, когда все не так однозначно?

## Тест снимков без четкой линии

Давайте посмотрим, как наш код работает с другими изображениями. Мы узнаем, как ведет себя робот в таких условиях и устраним возможные неполадки.

Что, если разместить линию на неравномерной поверхности, например на ковре? А что насчет снимка бумаги или ковра без линии?

На рис. 14.11 показано несколько графиков, которые расскажут вам больше о нашей системе.

В верхней части рисунка показаны три исходных изображения. Ниже следуют три графика, которые показывают результаты нахождения разностей и середины линии на изображениях без размытия. На нижней части показаны три графика, полученных от изображений с размытием.

При обработке зашумленных изображений (что выходит за рамки функционала нашего кода) понижение их резкости позволяет различить нахождение линии и нахождение случайного артефакта. Однако на втором графике мы видим, что один из артефактов отображается на графике как большой отрицательный пик, похожий на тот, который появляется при обнаружении линии. В этом случае должно помочь большее размытие по оси Y.

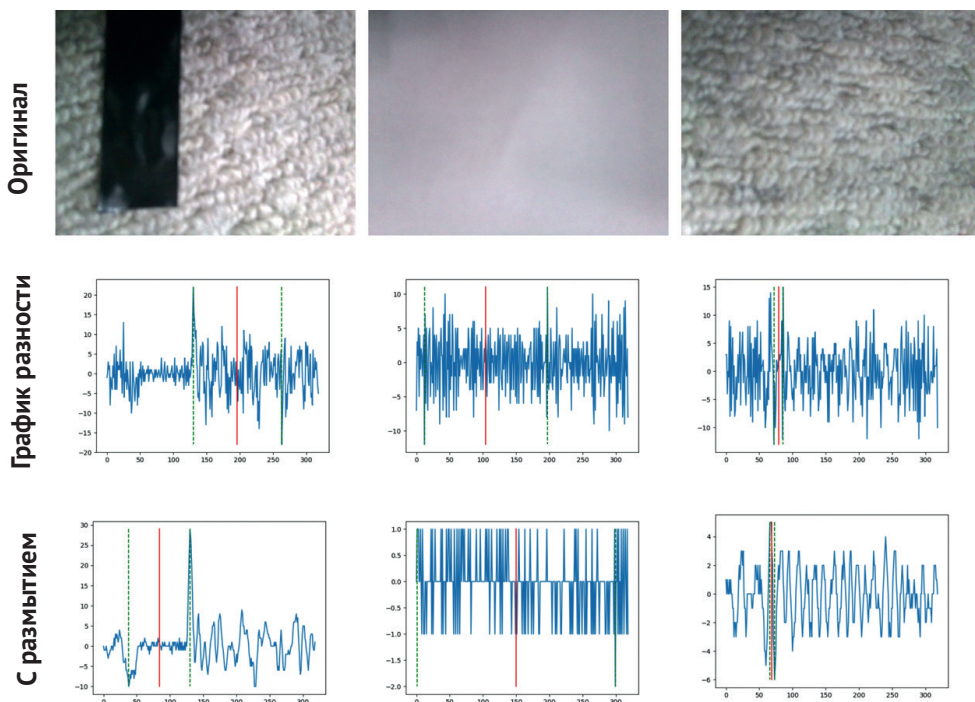


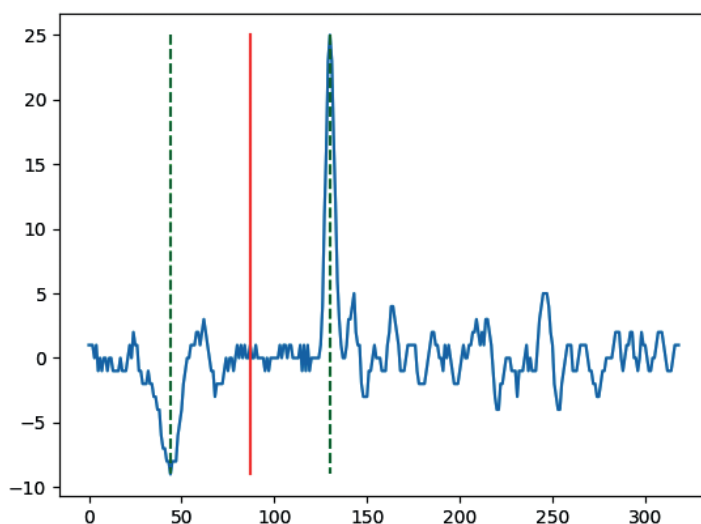
Рис. 14.11. Графики разностей для более зашумленных снимков

При внимательном рассмотрении вы увидите, что масштаб этих графиков неодинаков. На графике, соответствующем снимку чистого листа бумаги, без размытия появляются пики  $+10/-10$ , а с размытием  $+1/-1$ . Так если разница настолько мала, нужно ли вообще искать пики? То же самое происходит и на графиках для снимков поверхности ковра.

Мы можем внести некоторые изменения в систему, чтобы она не считала подобные артефакты линиями. Самый простой способ – добавить условие, которое отфильтровывает значения минимального положения выше  $-5$  и максимального – ниже  $10$ . Цифра  $-5$  не случайна, поскольку иначе система отфильтровала бы линию на первом графике. В этом случае может помочь увеличение области размытия.

Понижать резкость нужно в зависимости от зашумленности снимка. Для снимка линии с хорошим освещением размытие, скорее всего, не понадобится.

На рис. 14.12 показан график для снимка линии на ковре с понижением резкости, установленным на  $(5, 40)$ , с дальнейшим понижением резкости между рядами и дополнительной фильтрацией шума.



**Рис. 14.12.** График от снимка линии на ковре с большим понижением резкости

На графике, показанном на рис. 14.12, значительно меньше шума. Размытие сгладило побочные пики, но при этом остались пики настоящей линии. Такой метод замедляет работу алгоритма, поэтому его лучше использовать только для сильно зашумленных снимков.

Как видите, с помощью тестовых изображений мы узнали много нового о системе. Вы можете оптимизировать алгоритм для разных сценариев, используя одни и те же изображения, но пробуя другие параметры и экспериментируя с конвейером. При работе с компьютерным зрением эксперименты – это очень полезный подход.

Теперь, когда мы опробовали наш код обработки видеоданных на тестовых изображениях, пришло время применить его в поведенческом скрипте!

## Следование по линиям с помощью ПИД-алгоритма

В этом разделе мы объединим обработку видеоданных, управляющие контуры ПИД и потоковую передачу данных камеры, описанную в главе 13 (начнем с кода из этой главы).

Вам понадобятся следующие файлы:

- `pid_controller.py`;
- `robot.py`;
- `servos.py`;
- `camera_stream.py`;
- `image_app_core.py`;
- `leds_led_shim.py`;
- `encoder_counter.py`;
- папка с шаблонами.

Для визуализации результатов мы будем использовать тот же шаблон, но в выходной фрейм добавим быстрый способ рендеринга графиков `diff` в `OpenCV`. Библиотека `Matplotlib` не подойдет, поскольку она слишком медленная.

## Создание блок-схемы поведенческого сценария

Прежде чем перейти к разработке скрипта, создадим схему потока данных, которая позволит увидеть, что происходит с данными после обработки.

Схема выглядит знакомо, поскольку она очень похожа на те, которые мы создавали в главе 13. Взгляните на рис. 14.13.

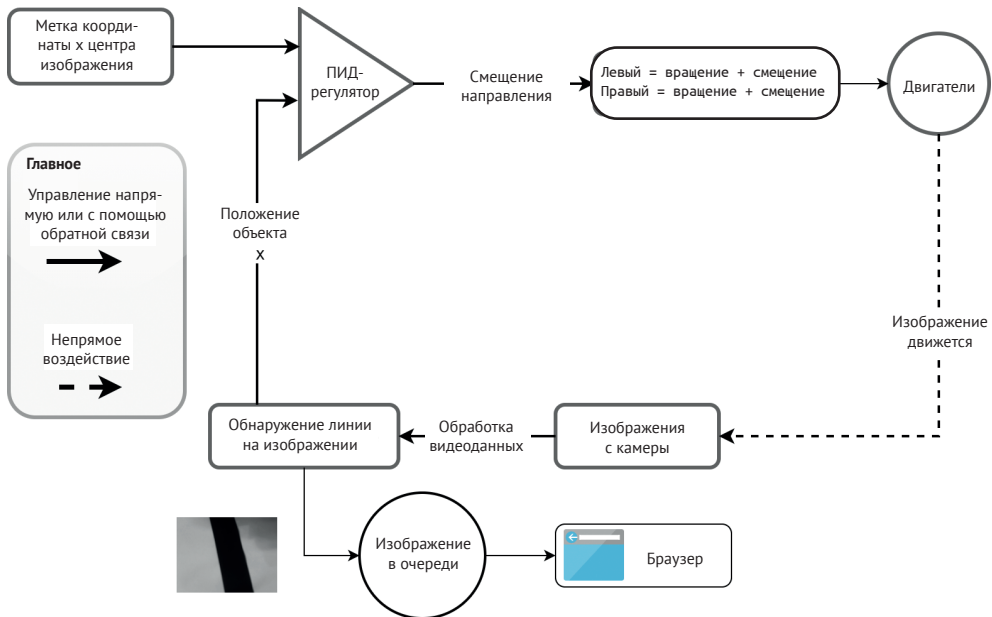


Рис. 14.13. Блок-схема поведенческого сценария следования по линиям

На рис. 14.13 мы видим, что изображение с камеры проходит через этап **обнаружения линии на изображении**. Этот шаг выводит  $X$ -позицию объекта (сердину линии), а затем эти данные переходят в ПИД-регулятор. Обратите внимание, что данные изображения передаются из этапа обнаружения линии еще и в очередь изображений, чтобы вы могли видеть их в браузере.

ПИД-регулятор также принимает метку средней точки, по которой центрируется камера. Он использует ошибку для расчета смещения и на основе этого управляет двигателями.

На схеме показано, что двигатели имеют обратную связь с камерой, поскольку их вращение может косвенно повлиять на изображение, и мы увидим другую линию.

Сначала необходимо усовершенствовать наш ПИД-регулятор.



## Добавляем время в ПИД-регулятор

Поведенческие сценарии нашего робота включают в себя обработку кадров и отправку ПИД-регуляторам данных об ошибках рассогласования всякий раз, когда процесс завершает цикл. В цикле могут происходить самые разные операции, поэтому время его выполнения варьируется. Когда мы формируем интегральное значение, мы добавляем данные так, как если бы время было постоянным. Чтобы получить более точную картину, необходимо умножить значение интеграла на значение времени.

1. Откройте файл `pid_controller.py`.
2. В методе `handle_integral` измените параметры, чтобы они принимали `delta_time`:

```
def handle_integral(self, error, delta_time):
```

3. Добавьте этот параметр в интегральный компонент:

```
self.integral_sum += error * delta_time
```

4. Обычно мы обновляем значения ПИД с помощью метода `get_value`, но, поскольку у нас уже есть код, использующий его, необходимо внести в него изменения. Для этого мы добавим параметр `delta_time` со значением по умолчанию 1:

```
def get_value(self, error, delta_time=1):
```

5. Когда метод `get_value` вызывает `handle_integral`, он всегда должен передавать новый параметр `delta_time`:

```
p = self.handle_proportional(error)
i = self.handle_integral(error, delta_time)
logger.debug(f"P: {p}, I: {i:.2f}")
return p + i
```

Мы внесли небольшие изменения, благодаря которым сможем учитывать разницу во времени между обновлениями кода ПИД-регуляторов.

Теперь можем использовать это в поведенческом скрипте.

## Разработка начального кода для поведенческого скрипта

Соберем все имеющиеся элементы и на их основе создадим поведенческий скрипт для отслеживания линий.

1. Создайте файл с именем `line_follow_behavior.py`.
2. Начните с импортов для `image_app_core`, `NumPy`, `OpenCV`, потока камеры, ПИД-регулятора и объекта робота. У нас есть библиотека `time`, поэтому позже мы сможем вычислить дельту (разность) времени:

```
import time
from image_app_core import start_server_process, get_
control_instruction, put_output_image
import cv2
import numpy as np
import camera_stream
from pid_controller import PIDController
from robot import Robot
```

3. Создадим класс нашего поведенческого скрипта. Конструктор, как и прежде, принимает объект робота:

```
class LineFollowingBehavior:
    def __init__(self, robot):
        self.robot = robot
```

4. Для отслеживания нашего сценария необходимо добавить переменные в конструктор. Задаем ряд пикселей, в котором мы будем искать разности, и граничное значение (все, что ниже него, не будет считаться линией):

```
self.check_row = 180
self.diff_threshold = 10
```

5. Как в скрипте для камеры, у нас есть уставка ПИД для центра (переменная, указывающая, должны ли двигатели работать) и скорость движения вперед:

```
self.center = 160
self.running = False
self.speed = 6
```

6. У нас будет несколько интересных визуальных элементов. Сохраним нужные цвета – зеленый для перекрестия, красный для средней линии и светло-синий для графика. Мы используем модель BGR, а OpenCV ожидает следующие значения:

```
self.crosshair_color = [0, 255, 0]
self.line_middle_color = [128, 128, 255]
self.graph_color = [255, 128, 128]
```

Конструктор для нашего скрипта готов.

7. Теперь нам нужен элемент управления, который будет определять действия запуска и остановки системы. Этот код аналогичен тому, который мы использовали в других скриптах для камеры, поэтому он покажется вам знакомым:

```
def process_control(self):
    instruction = get_control_instruction()
    if instruction:
        command = instruction['command']
        if command == "start":
            self.running = True
        elif command == "stop":
            self.running = False
        if command == "exit":
            print("Stopping")
            exit()
```

8. Далее мы создаем метод `run`, который будет выполнять основной цикл ПИД-регуляторов и управлять роботом. Сервоприводу наклона мы задаем значение 90, а сервоприводу поворота 0 – так механизм будет смотреть прямо вниз. Также настраиваем камеру:

```
def run(self):
    self.robot.set_pan(0)
    self.robot.set_tilt(90)
    camera = camera_stream.setup_camera()
```

9. Теперь настраиваем ПИД-регулятор направления. Возможно, вам нужно будет скорректировать значения, представленные в этом фрагменте кода. Значение пропорционального компонента здесь небольшое, поскольку значение ошибки направления может быть достаточно высоким по сравнению со значением частоты вращения двигателя:

```
direction_pid = PIDController( proportional_
constant=0.4, integral_constant=0.01, windup_limit=400)
```

10. Устанавливаем небольшую паузу, во время которой камера инициализируется, а сервоприводы приведут ее в нужное положение:

```
time.sleep(1)
self.robot.servos.stop_all()
print("Setup Complete")
```

После того как сервоприводы придут в нужное положение, мы отключаем их в целях экономии энергии.

11. Мы собираемся отслеживать время, поэтому нам нужно сохранить последнее значение. Время измеряется в секундах, а значение представляет собой число с плавающей запятой:

```
last_time = time.time()
```

12. Мы запускаем цикл камеры и передаем кадр в метод `process_frame` (мы создадим его позже). Также обрабатываем управляющую команду:

```
for frame in camera_stream.start_stream(camera):
    x, magnitude = self.process_frame(frame)
    self.process_control()
```

На этапе обработки кадра мы получаем значения `x` и `magnitude` (величина – разница между самым высоким и самым низким значениями разностей). Промежуток между пиками помогает определить, действительно ли это линия, а не просто шум.

13. Чтобы начать движение, нужно узнать, работает ли робот, и проверить, превышает ли найденная нами величина граничное значение:

```
if self.running and magnitude > self.diff_
threshold:
```

14. Если она больше, то запускаем поведенческий скрипт для ПИД:

```
direction_error = self.center - x
new_time = time.time()
dt = new_time - last_time
direction_value = direction_pid.get_
value(direction_error, delta_time=dt)
last_time = new_time
```

Мы вычисляем значение ошибки направления, вычитая из уставки для центра значение  $x$ . Затем получаем новое значение времени и вычисляем разницу между этим значением и исходным –  $dt$ . Ошибка направления и дельта времени передаются в ПИД-регулятор и получают новые значения. Сейчас мы готовы к следующему вычислению: `last_time` получает значение `new_time`.

15. Записываем новое значение в журнал и изменяем направление робота. Устанавливаем частоту вращения двигателя на базовое значение, а затем прибавляем/вычитаем выходное значение ПИД-регулятора двигателей:

```
print(f"Error: {direction_error},
Value:{direction_value:2f}, t: {new_time}")
self.robot.set_left(self.speed -
direction_value)
self.robot.set_right(self.speed +
direction_value)
```

16. Теперь мы увидели, что происходит, когда робот обнаруживает линию. Но что, если он не может обнаружить ее? Оператор `else` останавливает работу двигателей и сбрасывает значения ПИД-регулятора, чтобы они не накапливались:

```
else:
    self.robot.stop_motors()
    if not self.running:
        direction_pid.reset()
    last_time = time.time()
```

Обратите внимание, что мы по-прежнему поддерживаем актуальное значение времени. В противном случае возник бы большой разрыв между временем остановки и временем запуска, из-за чего в ПИД-регуляторе начали бы накапливаться странные значения.

17. Далее мы указываем, что происходит, когда мы обрабатываем кадр. Добавим метод `process_frame`:

```
def process_frame(self, frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.blur(gray, (5, 5))
    row = blur[self.check_row].astype(np.int32)
    diff = np.diff(row)
    max_d = np.amax(diff, 0)
    min_d = np.amin(diff, 0)
```

Этот код покажется вам знакомым, поскольку это тестовый код, который мы создали ранее.

18. Нам нужно проверить, чтобы значения на графике были как выше нуля, так и ниже, чтобы мы нашли два разных положения. Максимум должен быть строго выше нуля, а минимум ниже. Если получился другой результат, главный цикл не воспримет это как линию:

```
if max_d < 0 or min_d > 0:
    return 0, 0
```

19. Мы найдем положения в ряду, как делали это раньше, а также среднюю точку:

```
highest = np.where(diff == max_d)[0][0]
lowest = np.where(diff == min_d)[0][0]
middle = (highest + lowest) // 2
```

20. Чтобы мы могли использовать это значение для определения наличия положительного совпадения, рассчитаем величину разницы между минимумом и максимумом (так мы определим, что это именно то, что нужно):

```
mag = max_d - min_d
```

21. Далее нам нужно настроить отображение данных для пользователя. Как и в предыдущих скриптах для камеры, мы вызываем метод `make_display` и передаем ему переменные, на основе которых он строит график:

```
self.make_display(frame, middle, lowest, highest,
diff)
```

22. Возвращаем среднюю точку и величину:

```
return middle, mag
```

23. Этот код будет управлять движением нашего робота, но, если мы не увидим детали процесса, нам будет сложно его настроить. Для отображения создадим метод `make_display`:

```
def make_display(self, frame, middle, lowest,
highest, diff):
```

Мы передаем ему следующие параметры: исходный кадр (`frame`)), среднее положение линии (`middle`), положение разности `lowest`, положение разности `highest` и разность в строке `diff`.

24. Первое, что нам нужно отобразить, – это метка центра. Определим выбранный ряд и поместим в центр перекрестие:

```
cv2.line(frame, (self.center - 4, self.check_
row), (self.center + 4, self.check_row), self.crosshair_
color)
cv2.line(frame, (self.center, self.check_row - 4), (self.center, self.
check_row + 4), self.crosshair_
color)
```

25. Затем линией другого цвета показываем найденную середину:

```
cv2.line(frame, (middle, self.check_row - 8),
(middle, self.check_row + 8), self.line_middle_color)
```

26. Отмечаем значения `lowest` и `highest` другим цветом, чтобы мы могли их различить:

```
cv2.line(frame, (lowest, self.check_row - 4),
(lowest, self.check_row + 4), self.line_middle_color)
cv2.line(frame, (highest, self.check_row - 4),
(highest, self.check_row + 4), self.line_middle_color)
```

27. Теперь нам нужно построить график `diff` в новом пустом фрейме. Создадим пустой фрейм – простой массив NumPy:

```
graph_frame = np.zeros((camera_stream.size[1],
camera_stream.size[0], 3), np.uint8)
```

Размерность массива определяется количеством строк и столбцов, поэтому мы меняем значения размеров области захвата камеры  $X$  и  $Y$ .

28. Затем используем метод для построения простого графика (мы реализуем его позже). Передаем ему следующие параметры: фрейм, в котором будет находиться график, и значения  $Y$ . Значения  $X$  выступают в роли номеров столбцов:

```
self.make_cv2_simple_graph(graph_frame, diff)
```

29. Теперь нам нужно объединить пустой фрейм и фрейм графика (мы уже делали это в коде для определения цвета объекта):

```
display_frame = np.concatenate((frame, graph_
frame), axis=1)
```

30. Кодировать байты и помещаем их в очередь вывода:

```
encoded_bytes = camera_stream.get_encoded_bytes_
for_frame(display_frame)
put_output_image(encoded_bytes)
```

31. Реализуем метод `make_cv2_simple_graph`, который рисует линии между точками  $Y$  вдоль оси  $x$ :

```
def make_cv2_simple_graph(self, frame, data):
```

32. Нам нужно сохранить последнее значение, чтобы код отображал следующее значение относительно него – в результате получается линейный график. Начнем с элемента 0. Также зададим для графика произвольную среднюю точку  $Y$ . Мы уже знаем, что значения `diff` могут быть отрицательными:

```
last = data[0]
graph_middle = 100
```

33. Затем мы должны перечислить данные для построения каждого элемента:

```
for x, item in enumerate(data):
```

34. Теперь мы можем построить линию от позиции последнего элемента  $Y$  до текущей позиции следующего элемента  $X$ . Обратите внимание, что мы смещаем каждый элемент относительно середины графика:

```
cv2.line(frame, (x, last + graph_middle), (x
+ 1, item + graph_middle), self.graph_color)
```

35. Заменяем последний элемент на текущий:

```
last = item
```

Почти готово, можем перейти к построению графика.

36. Наш поведенческий скрипт готов, и для его запуска нужен внешний код. Он тоже похож на примеры, которые мы создавали для других скриптов камеры:

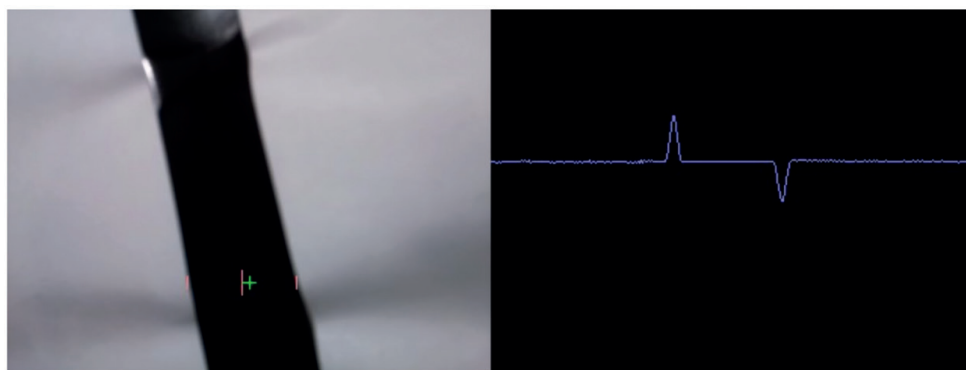
```
print("Setting up")
behavior = LineFollowingBehavior(Robot())
process = start_server_process('color_track_behavior.
html')
try:
    behavior.run()
finally:
    process.terminate()
```

Помните, что мы по-прежнему используем шаблон `color_track_behavior.html`.

Далее загружаем код в память робота, а затем запускаем его и включаем двигатели. Для просмотра веб-интерфейса введите в браузере <http://myrobot.local:5001>.

Результат будет выглядеть так, как показано на рис. 14.14.

## Robot Image Server



[Start](#) [Stop](#)  
[Exit](#)

Рис. 14.14. Скриншот вывода поведенческого скрипта для следования по линиям

На рис. 14.14 мы видим заголовок, а под ним два изображения. Слева показан снимок линии с камеры. Зеленое перекрестие на нем показывает центральную точку. Длинной вертикальной красной чертой отмечена середина линии, а красными чертами покороче – ее границы. Справа показан график изменения разности оттенков пикселей после размытия. На графике есть положительные и отрицательные пики.

Ниже находятся кнопки **Start** (Пуск), **Stop** (Стоп) и **Exit** (Выход).

Обеспечив хорошее освещение поместите робота на линию. Если в браузере вы видите примерно то же самое, что на рис. 14.14 слева, нажмите кнопку **Start** (Пуск), чтобы увидеть код в действии. Робот должен начать шатко следовать по линии.



## Настройка параметров ПИД-регулятора

Нам необходимо, чтобы робот отслеживал изогнутые линии и находил их границы более эффективно. Сейчас он может промахиваться или выполнять повороты с замедлением, и здесь на помощь приходит настройка параметров ПИД-регулятора:

- если вам кажется, что робот поворачивает с замедлением, попробуйте немного увеличить пропорциональную постоянную. И наоборот, если он поворачивает слишком быстро, попробуйте уменьшить ее;
- при возникновении непрерывной ошибки увеличьте интегральную постоянную.

Для настройки ПИД-регуляторов вам, вероятно, потребуется несколько попыток. Запаситесь терпением.

## Устранение неполадок

Если что-то работает не так, как ожидалось, воспользуйтесь следующими советами:

- если сервопривод наклона не обращается вниз при установке его значения на 90°, возможно, он неправильно откалиброван. Замените параметр значения `deflect_90_in_ms` на объект `Servos` и пошагово увеличивайте его значение на 0,1, пока не получите 90°;
- если линия на снимке получается нечеткой, проверьте освещение и поверхность – она должна быть ровной (как, например, лист бумаги), и линия должна выделяться на ее фоне;
- если робот не может обнаружить линию, попробуйте увеличивать вертикальное размытие с шагом 5;
- если робот не успевает повернуть по линии, попробуйте уменьшить частоту вращения двигателей с шагом 10;
- если картинка на камере пошатывается по горизонтали, удалите строку `self.robot.servos.stop_all()` из кода `line_follow_behavior`. Будьте осторожны, поскольку этот шаг повысит энергопотребление;
- если робот обнаруживает много побочных линий, увеличьте вертикальное размытие. Также можете начать увеличивать граничное значение с шагом 1 или 2, чтобы добиться более выразительного контраста яркости;
- просмотрите код и убедитесь, что не упустили каких-либо фрагментов из этой главы или из главы 13.

## НАХОЖДЕНИЕ ЛИНИИ ПРИ ЕЕ ПОТЕРЕ

Важно учитывать, что будет делать робот при потере линии. Возвращаясь к примерам использования таких роботов в промышленности, мы понимаем, что от этого может зависеть безопасность.

Наш робот в такой ситуации просто останавливается. Когда вы поставите его обратно на линию, он сразу же начнет двигаться. Такое поведение у небольших роботов – норма, но у более крупных оно становится опасным.

Мы можем заставить нашего робота делать повороты, пока он снова не найдет линию. Он может потерять линию, если не доедет или, наоборот, пересечет ее границы, а потом не сможет их найти или если доедет до ее конца. Такое поведение приемлемо только для не очень серьезных соревнований роботов.

Вам нужно тщательно продумать такие моменты и понять, где робот будет использоваться. Роботы для соревнований и промышленные роботы, как правило, оснащаются несколькими датчиками, направленными под разными углами, либо датчиком с широким углом обзора, поэтому у них гораздо меньше шансов потерять линию, чем у нашего робота. Кроме того, если крупный робот начнет крутиться, даже медленно, это может быть опасно. По этой причине мы реализуем простую дополнительную функцию для безопасности.

Если робот теряет линию, он не просто останавливает двигатели, а присваивает признаку состояния значение `false`, поэтому его нужно запустить вручную.

1. Откройте файл `line_follow_behavior.py`.
2. Перейдите к методу `run` и найдите оператор `else`.
3. Внесите изменения в его содержимое:

```
else:
    self.robot.stop_motors()
    self.running = False
    direction_pid.reset()
    last_time = time.time()
```

Здесь мы внесли два небольших изменения. Вместо того чтобы сбрасывать значения ПИД-регуляторов, когда `running` имеет значение `False`, теперь мы каждый раз устанавливаем для `running` значение `False`. Также мы сбрасываем значения ПИД-регуляторов.

Сохраните и запустите код на роботе и подождите, когда он потеряет линию (пока не отклонится от курса или не достигнет конца). Робот должен остановиться и подождать, пока вы нажмете кнопку запуска, прежде чем снова начать движение. Чтобы робот поехал, его нужно поставить обратно на линию и нажать кнопку запуска.

Теперь мы знаем, как робот ведет себя при потере линии.

## Выводы

В этой главе вы узнали, как обнаруживать линии с помощью камеры и визуализировать получаемые данные. Вы увидели, как применить эти данные в поведенческом скрипте для следования по линии. Вы познакомились с OpenCV и узнали хитрый способ помещать графики на выходные кадры камеры. Также мы обсудили настройку ПИД-регулятора и сделали движения робота более точными. В завершение вы узнали, как обеспечить остановку робота при потере линии.

В следующей главе мы увидим, как общаться с нашим роботом с помощью голосового агента Mycroft. Мы добавим к Raspberry Pi микрофон и динамики, а затем установим программное обеспечение для распознавания речи. Так мы сможем отправлять голосовые команды Raspberry Pi, который будет отправлять их роботу, а Mycroft будет отвечать.

## ЗАДАНИЕ

Сейчас все работает, но мы можем усовершенствовать систему и сделать ее более интересной:

- в методе `make_display` используйте `cv2.putText` для отрисовки значений (например, данных ПИД-регулятора) в кадрах;
- попробуйте записать в отдельный файл данные ПИД-регулятора и ошибки в зависимости от времени, а затем, используя Matplotlib, загрузите их в другой файл Python и посмотрите, что получилось. В ретроспективе это изменение позволит яснее увидеть недостаточную/избыточную поворачиваемость;
- внесите некоторые изменения в код управления двигателями, чтобы они работали быстрее, когда линия находится ближе к середине, и медленнее, когда она находится дальше;
- вы можете реализовать проверку двух рядов пикселей и нахождение угла между ними. Так вы узнаете, как далеко линия находится от середины и в какую сторону она направлена. Эти данные можно использовать для управления роботом на следующих участках пути.

Эти задания подскажут вам несколько интересных способов поэкспериментировать с новыми знаниями и умениями, приобретенными в этой главе.

## Дополнительные материалы

Узнать больше об отслеживании линий вы можете в следующих материалах:

- ознакомиться с материалом об альтернативном подходе к обработке строк на языке Go на Raspberry Pi от легенды Pi Wars Брайана Старки (Brian Starkey) можно по адресу <https://blog.usedbytes.com/2019/02/autonomous-challenge-blast-off/>;
- информацию о роботе, который тоже умеет следовать по линии, можно найти по адресу <https://www.raspberrypi.org/blog/an-image-processing-robot-for-robocup-junior/>. Здесь реализуется подход, похожий на наш, но более сложный.

# Глава 15

## Голосовая связь с роботом посредством Mycroft

Способность робота принимать голосовые команды и давать ответы уже давно является признаком «разумности». Некоторые устройства вокруг нас, например те, которые используют Alexa и Google Assistant, умеют это делать и имеют соответствующие инструменты. Путем интеграции этих инструментов и кода мы можем создать эффективную систему голосового помощника. Mycroft – это голосовая система на основе Python с открытым исходным кодом<sup>14</sup>. Мы подключим к Raspberry Pi динамики и микрофоны и запустим на нем Mycroft, чтобы управлять роботом с помощью голосовых команд.

В этой главе вы познакомитесь с Mycroft и узнаете, как добавить плату динамика/микрофона к Raspberry Pi. Затем мы установим и настроим Mycroft на Pi.

Также опробуем другие возможности Flask и разработаем с его помощью API (Application Programming Interface – программный интерфейс приложения) с большим количеством точек управления.

В завершение главы вы разработаете собственный код навыка для голосового помощника. В дальнейшем благодаря полученным знаниям вы сможете доработать его.

В этой главе мы рассмотрим следующие темы:

- введение в Mycroft – основные понятия;
- ограничения возможностей голосового управления;
- установку платы динамика/микрофона на робота;
- настройку и конфигурацию Raspberry Pi для запуска Mycroft;
- разработку API управления с помощью Flask;
- создание кода навыка для голосового помощника.

---

<sup>14</sup> К сожалению, на момент подготовки к изданию перевода этой книги Mycroft не поддерживал команды на русском языке. Возможно, найдутся энтузиасты, которые обучат голосовую модель для поддержки русского языка. Пожалуйста, сообщите об этом в издательство, и мы незамедлительно обновим информацию в книге. А пока читатели могут потренироваться в произношении несложных фраз на английском языке, общаясь с роботом. Это не только увлекательно, но и полезно. – *Прим. ред.*

## ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе вам понадобится следующее:

- еще один микрокомпьютер Raspberry Pi 4 (модель B);
- SD-карта (как минимум на 8 ГБ);
- ПК, способный записывать данные на SD-карту (с утилитой balenaEtcher);
- плата расширения ReSpeaker 2-Mics Pi HAT;
- миниатюрный магнитный динамик для Raspberry Pi с разъемом JST или мини-джек (3,5 мм);
- кабель Micro-HDMI–HDMI (понадобится на этапе устранения неполадок);
- блок питания Micro-USB;
- робот, созданный в предыдущих главах (его мы будем заставлять двигаться).

Полный код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter15>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/2N5bXqr>.

## ВВЕДЕНИЕ В MYCROFT – ОСНОВНЫЕ ПОНЯТИЯ

**Mycroft** – это программный комплекс **голосового помощника**. Он принимает голосовые команды и выполняет соответствующие действия. Mycroft создан на Python, и его исходный код находится в открытом доступе. По большей части обработка голосовых сигналов происходит в облаке. После обработки команд Mycroft отвечает пользователю голосом.

Mycroft снабжен онлайн-документацией и имеет сообщество пользователей. Помимо Mycroft, в дальнейшем вы можете поэкспериментировать с такими помощниками, как Jasper, Melissa-AI и Google Assistant.

В следующих подразделах поговорим об основных понятиях Mycroft.

### Преобразование речи в текст

**Преобразование речи в текст** (Speech to text – STT) – это технология, на основе которой работают системы, преобразующие звук (человеческую речь) в текст (набор слов на экране), пригодный для обработки компьютером.

Такие системы могут работать локально или в облаке (на более мощных машинах).

### Слово для пробуждения

У голосовых помощников есть **слова для пробуждения** – это может быть слово или фраза, которое произносится перед голосовой командой, например «Привет, Siri», «Привет, Google» и «Alexa». Для Mycroft по умолчанию используется слово «Mycroft» или фраза «Hey Mycroft», но вы можете сменить ее.

До пробуждения голосовой помощник будет игнорировать все фразы, кроме слова для пробуждения. Оно распознается на устройстве локально. После пробуждения помощник отбирает звуки и распознает их с помощью системы преобразования речи в текст.

## Реплика

**Реплики** – это фразы, которые говорит пользователь. У голосовых помощников есть словари (определяемые пользователем), которые помогают им сопоставить голосовую команду и действие. У Mycroft есть словарь, который вызывает обработчик намерений.

Словарь Mycroft – это файл, в котором хранятся списки взаимозаменяемых (схожих по значению) фраз.

Одним из примеров реплики является команда «*Hey Mycroft, what is the weather?*» («Эй, Майкрофт, какая погода?»).

## Намерение

**Намерение** – это задача, выполняемая голосовым помощником (например, узнать, какая сегодня погода). Для нашего робота мы также создадим намерения. Намерение является частью навыка. Оно определяет код обработчика для задачи и диалог для ответа.

В рамках навыка просмотра погоды реплика «*What is the weather?*» («Какая погода?») вызывает намерение узнать текущую погоду в соответствии с местоположением, а затем сообщить информацию пользователю. В качестве примера для нашего робота мы можем *попросить его протестировать светодиоды*, что вызовет намерение запустить поведенческий скрипт радужного свечения (из главы 9).

## Диалог

В терминологии Mycroft **диалог** – это фраза, которую голосовой помощник говорит пользователю, например: «*OK, the robot has been started*» («ОК, робот запущен») или «*Today, the weather is clear*» («Сегодня ясная погода»).

У навыков есть свои диалоги. Файлы диалогов включают в себя наборы взаимозаменяемых слов или фраз (которые могут быть на разных языках).

## Словарь

После преобразования реплик пользователя в текст помощник ищет соответствие в **словаре**. Словари, как и диалоги, являются частью намерения – с их помощью робот сопоставляет реплику с действием. Файлы словарей содержат синонимичные фразы, которые могут быть объединены в языковые наборы, что позволяет создавать словари на разных языках.

Фразы «*What is the weather?*» («Какая погода?»), «*Is it sunny?*» («Сегодня солнечно?»), «*Do I need an umbrella?*» («Нужно ли взять зонт?») или «*Will it rain?*» («Пойдет ли дождь?») будут считаться синонимичными. Вы можете использовать разные словари; например, чтобы *спросить робота о чем-либо* – один словарь, а чтобы *заставить его двигаться вперед* – другой.

## Навык

**Навыки** – это контейнеры, в которых хранятся все словари с репликами, диалоги и намерения. Навык будильника может включать такие намерения, как установка, удаление или изменение будильника и список будильников. Он мо-

жет содержать диалог, сообщающий пользователю, что будильник установлен, или диалог для подтверждения установки.

Позже в этой главе мы создадим навык MyRobot, который будет содержать измерения «начать движение» и «остановиться».

Вы познакомились с основными понятиями и компонентами Mycroft. Далее поговорим о том, как реализовать это с аппаратной точки зрения. Куда установить динамики и микрофоны?

## ОГРАНИЧЕНИЯ ГОЛОСОВОГО УПРАВЛЕНИЯ

Прежде чем перейти к аппаратной части, мы должны рассмотреть ряд вопросов. Где лучше установить динамики и микрофоны? Как будет выполняться обработка данных: локально или в облаке?

Вот факторы, которые нужно учитывать:

- **шум:** робот, оснащенный двигателями, достаточно сильно шумит, поэтому микрофоны необходимо расположить вдали от них;
- **питание:** голосовой помощник работает всегда и в любой момент готов начать принимать команды. У нашего робота уже есть несколько датчиков, которым тоже необходимо питание. Здесь нужно учитывать как батарейное питание, так и энергопотребление центрального процессора;
- **размер и физическое размещение:** динамик и плата расширения для голосовой связи добавят и без того загруженному роботу габаритов и усложнят его электрическую схему.

В большом роботе микрофон и динамик можно разместить на высокой стойке с еще одним Raspberry Pi. Однако для нашего небольшого и простого робота такой вариант не подходит. У нас плата голосового помощника будет взаимодействовать с роботом на расстоянии. Ее мы установим на второй Raspberry Pi.

Мы реализуем систему обработки речи в облаке. Локальная система была бы быстрее и безопаснее, но на момент написания этой книги готового программного обеспечения такого типа для Raspberry Pi еще не существует. Mycroft позволяет добавлять новые навыки и имеет ПО для обработки голоса, которое можно подключить дополнительно, поэтому в дальнейшем его можно будет запустить локально.

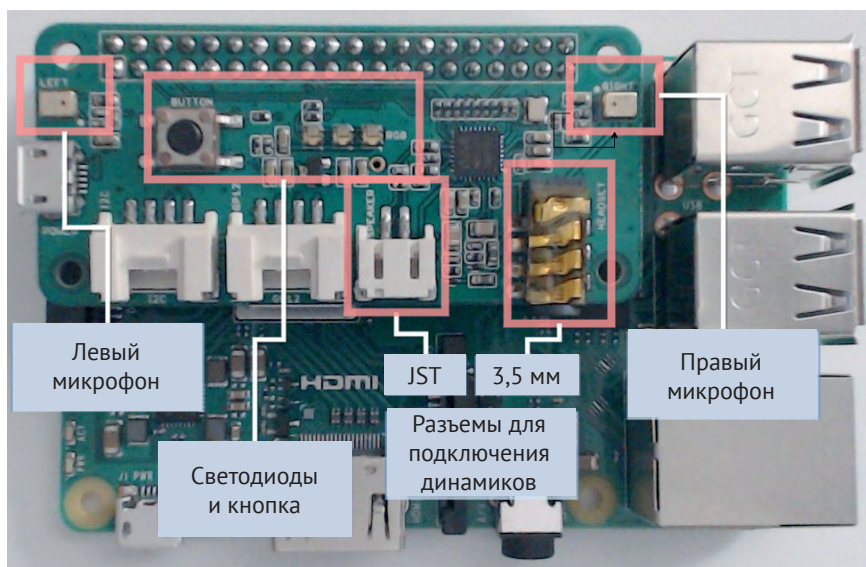
Теперь, когда мы решили, что будем использовать Mycroft и второй Raspberry Pi, пришло время приступить к сборке.

## Аудиовход и выход для RASPBERRY PI

Прежде чем мы сможем использовать обработку голоса / голосового помощника, нам нужно подключить к Raspberry Pi динамики и микрофоны. Для этого понадобятся дополнительные устройства: микрофонный массив (для более эффективного распознавания речи) и плата расширения ReSpeaker 2-Mics Pi HAT (ее можно приобрести во многих магазинах).

На рис. 15.1 показано, как выглядит плата.





**Рис. 15.1.** Плата расширения ReSpeaker 2-Mics Pi HAT

На рис. 15.1 показана плата расширения ReSpeaker 2-Mics Pi HAT, установленная на Raspberry Pi. Слева находится левый микрофон (отмечен подписью). У платы их два (крошечные прямоугольные металлические компоненты, расположенные слева и справа). Следующая отметка на рисунке – три RGB-светодиода и кнопка, подключенная к GPIO. Правее находятся два разъема для подключения динамиков – 3,5 мм и JST. Чтобы услышать звук с выхода этой платы, нужно подключить динамик. Последняя отметка на рисунке – правый микрофон.

Я выбрал плату расширения ReSpeaker 2-Mic Pi HAT, поскольку она экономична и при этом отлично подходит для первых проектов с распознаванием речи. Обычные USB-микрофоны не подойдут. Существуют другие устройства, которые лучше поддерживаются Mycroft, но их нельзя установить на Pi как плату расширения. Выбранная плата является неплохим компромиссом – простая и экономичная, но с возможностью дополнительной настройки программного обеспечения. Давайте посмотрим, как установить ее на нашего робота.

## Физическая установка

Плата ReSpeaker 2-Mics HAT будет подключаться прямо к разъемам Raspberry Pi 4, а сама панель нависать над Pi.

Динамики будут иметь либо крошечный двухконтактный разъем (JST), который подходит к единственному двухконтактному разъему на плате, либо разъем мини-джек (3,5 мм). На рис. 15.2 показан динамик, подключенный к плате.

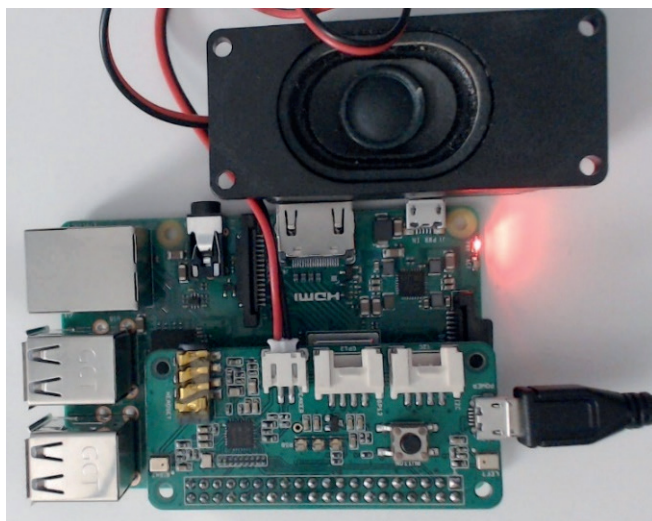


Рис. 15.2. Конфигурация голосового помощника Mycroft с ReSpeaker

На рис. 15.2 показана моя конфигурация Mycroft с ReSpeaker 2-Mics Pi HAT, размещенная на столе. Она включена, и на Raspberry Pi мы видим огонек светодиодного индикатора. Также я подключил динамик.

Можете поместить конструкцию в корпус Raspberry Pi или самодельный корпус, но главное – оставить микрофоны открытыми.

Также вам понадобится SD карта и источник питания.

#### Важное примечание

В следующих нескольких разделах я рекомендую использовать питание от сети. Пока не подключайте его и не включайте.

Итак, мы подготовили аппаратную часть с динамиками и микрофонами. В следующем разделе мы настроим Raspbian и программное обеспечение голосового помощника.

## Установка программного обеспечения голосового помощника на Raspberry Pi

У Mycroft есть дистрибутив для Raspbian. Запишем ПО на SD-карту.

1. Перейдите на веб-сайт Mycroft (<https://mycroft-ai.gitbook.io/docs/using-mycroft-ai/get-mycroft/picroft>) и загрузите образ *Picroft* – он основан на Raspbian Buster. Выберите **stable disk image** (стабильный образ диска).
2. Вставьте SD-карту в ваш компьютер. Воспользуйтесь инструкцией, представленной в разделе «Запись образа ОС Raspberry Pi на SD-карту» в главе 3. Вместо Raspbian выберите образ Picroft.

3. Проверьте, работает ли образ без монитора, через SSH и W-Fi. Для этого включите SSH и Wi-Fi, как мы делали в разделе «*Настройка Wi-Fi и SSH на Raspberry Pi*» в главе 4.

Теперь, когда SD-карта готова, проверим ее. Вставьте карту в Raspberry Pi (для голосового помощника) и подключите питание через разъем Micro-USB на ReSpeaker 2-Mics Pi HAT (не на Pi).

### Важное примечание

Подавать питание нужно не через Pi, а через ReSpeaker 2-Mics Pi HAT. Этой плате требуется прямое питание для вывода звука на динамик. В документации к ней говорится, что если подать питание через Pi, то вы не услышите звук из динамика. Подробнее см. [https://wiki.seeedstudio.com/ReSpeaker\\_2\\_Mics\\_Pi\\_HAT/#hardware-overview](https://wiki.seeedstudio.com/ReSpeaker_2_Mics_Pi_HAT/#hardware-overview).

Имя хоста начинается с `picraft.local`. Имя пользователя: `pi`, пароль: `mycroft`. Проверьте подключение Wi-Fi и SSH (для взаимодействия через PuTTY). Теперь вы можете запустить Raspberry Pi и приступить к настройке Mycroft.

## Установка программного обеспечения ReSpeaker

Когда вы войдете в систему, Mycroft выведет руководство по установке. Воспользуйтесь следующей инструкцией.

1. Когда Mycroft спросит, хотите ли вы выполнить пошаговую настройку, введите `Y`, чтобы ответить «да». При установке загрузится множество обновлений (это займет от 30 мин до часа).
2. Далее Mycroft попросит указать устройство аудиовыхода:

### HARDWARE SETUP

How do you want Mycroft to output audio:

- 1) Speakers via 3.5mm output (aka 'audio jack' or 'headphone jack')
- 2) HDMI audio (e.g. a TV or monitor with built-in speakers)
- 3) USB audio (e.g. a USB soundcard or USB mic/speaker combo)
- 4) Google AIY Voice HAT and microphone board (Voice Kit v1)
- 5) ReSpeaker Mic Array v2.0 (speaker plugged in to Mic board)

Choice [1-5]:

В списке нет варианта ReSpeaker 2-Mics Pi HAT. Введите `3`, чтобы выбрать USB-динамики (это установит базовые настройки по умолчанию).

3. Нажмите **Ctrl+C**, чтобы выйти из режима настройки, и вернитесь к приглашению `$` в командной строке.
4. Для успешного запуска нам потребуется обновить программное обеспечение на SD-карте. В командной строке введите команду `apt update -y && sudo apt upgrade -y`. Обновление займет некоторое время.
5. Чтобы обновления вступили в силу, перезагрузите Pi (с помощью команды `sudo reboot`). После перезагрузки Pi войдите в систему по `ssh`. Вы снова окажетесь в меню пошаговой настройки. Нажмите **Ctrl+C** еще раз.

6. Установите аудио драйверы для ReSpeaker 2-Mics Pi HAT с помощью следующих команд:

```
$ git clone https://github.com/waveshare/WM8960-Audio-HAT.git
$ cd WM8960-Audio-HAT
$ sudo ./install.sh
```

Выполнение первой команды (клонирование репозитория Git) займет 1-2 мин. Вторая команда укажет, что плата использует звуковой чип WM8960. Третья команда (выполнение скрипта установки) займет 20–30 мин.

7. Перезагрузите Pi еще раз. Для выхода из меню пошаговой настройки нажмите **Ctrl+C**.

Прежде чем двигаться дальше, необходимо проверить звук.

8. Чтобы получить список устройств воспроизведения, введите команду `aplay -l`. Ее вывод выглядит следующим образом:

```
card 1: wm8960soundcard [wm8960-soundcard], device 0: bcm2835-i2s-wm8960-hifi wm8960-hifi-0 [bcm2835-i2s-wm8960-hifi wm8960-hifi-0]
```

Здесь мы видим нашу карту.

9. Теперь проверяем, будет ли карта воспроизводить звук. Чтобы воспроизвести аудиофайл, используйте команду `aplay -Dplayback /usr/share/sounds/alsa/ Front_Left.wav`.

Эта команда указывает устройство с именем `playback` с флагом `-D`, а затем файл для воспроизведения. Устройство `playback` – это обработчик ALSA (Advanced Linux Sound Architecture – Продвинутая звуковая архитектура Linux), установленный по умолчанию. Он отвечает за микширование и позволяет избежать несоответствия битрейта и номера канала. В директории `/usr/share/sounds/alsa` вы можете найти другие тестовые аудиофайлы.

10. Далее проверим записывающие устройства с помощью команды `arecord -l`. В ее выводе мы видим, что наша карта обнаружена:

```
card 1: wm8960soundcard [wm8960-soundcard], device 0: bcm2835-i2s-wm8960-hifi wm8960-hifi-0 [bcm2835-i2s-wm8960-hifi wm8960-hifi-0]
```

Карта готова. Теперь нужно указать системе Muscroft выбрать ее для использования.

## Устранение неполадок

Если звук не работает, воспользуйтесь следующими советами.

1. Сначала введите команду `sudo poweroff`, чтобы выключить Raspberry Pi. На выключенном Pi проверьте все подключения. Плата должна быть полностью подключена к разъему GPIO. Посмотрите, правильно ли подключены динамики к ReSpeaker 2-Mics Pi HAT.
2. После того как снова включите Pi, проверьте, какой разъем питания вы используете (это должен быть разъем на ReSpeaker 2-Mics Pi HAT, а не на Raspberry Pi).

3. Если вместо динамика вы используете наушники, попробуйте. увеличить громкость. Введите команду `alsamixer`, выберите звуковую карту `WM8960` и увеличьте громкость. Проверьте, появился ли звук.
4. Проверьте, не пропустили ли вы шаг обновления (с командами `apt update` и `apt upgrade`). Без обновления драйверы не будут работать. Перезагрузите компьютер и попробуйте установить драйверы заново.
5. Если при установке драйверов не выполняется команда клонирования Git-репозитория, проверьте вызванный адрес.
6. При попытке воспроизведения звука, помните, что флаг `-D` чувствителен к регистру. Строчная `d` здесь не подойдет.

Если советы не помогли, перейдите по адресу <https://github.com/waveshare/WM8960-Audio-HAT>. На этом веб-сайте вы можете ознакомиться с документацией или задать вопрос.

Теперь перейдем к настройке взаимодействия звуковой карты и Mycroft.

## Настройка взаимодействия звуковой карты и Mycroft

На этом этапе вам нужно связать Mycroft и звуковую карту. Для этого отредактируйте файл конфигурации Mycroft.

1. С помощью команды `sudo nano /etc/mycroft/mycroft.conf` откройте файл конфигурации Mycroft как пользователь `root`.
2. В файле содержатся строки, описывающие различные аспекты Mycroft. Нас интересуют только две из них:

```
"play_wav_cmdline": "aplay -Dhw:0,0 %1",
"play_mp3_cmdline": "mpg123 -a hw:0,0 %1",
```

Первая строка указывает, что Mycroft будет воспроизводить звуковые файлы с помощью команды `aplay` на аппаратном обеспечении устройства `0,0` (разъем для наушников на Pi) – записывается как `hw:0,0`. Это не наше устройство. Вторая строка указывает, что Mycroft будет воспроизводить файлы `mp3` с помощью команды `mpg123` на том же устройстве. Если звук будет идти напрямую в аппаратное устройство, то будет воспроизводиться файл в формате по умолчанию, поэтому воспроизводимый звук должен проходить через микшер. Исправим это.

3. В обеих строках замените `hw:0,0` на `playback`. Они должны выглядеть так:

```
"play_wav_cmdline": "aplay -Dplayback %1",
"play_mp3_cmdline": "mpg123 -a playback %1",
```

4. Нажмите **Ctrl+X** чтобы перезаписать файл и выйти. Когда Mycroft спросит хотите ли вы перезаписать файл, введите `Y`, чтобы ответить «да».
5. Перезагрузите Pi еще раз. После включения не выходите из режима управления.
6. Mycroft спросит хотите ли вы протестировать устройство. Нажмите `T`, чтобы проверить динамик. Это может занять несколько секунд, и в результате вы услышите, как Mycroft говорит с вами. Если звук слишком тихий, попробуйте ввести цифру `9` и проверьте еще раз. Захватывающий момент! Нажмите `D` для завершения проверки.

7. Затем он спросит, хотите ли вы протестировать микрофон. Выберите **4** для **Other USB Microphone** (Другой USB-микрофон) и проверьте звук. Установщик попросит вас сказать что-нибудь в микрофон, а затем воспроизведет ваш голос. Если все в порядке, нажмите **1**.
8. Mycroft спросит, хотите ли вы использовать рекомендации (если да – выберите **1**). Далее он перейдет к настройкам пароля. Я рекомендую поменять пароль, установленный по умолчанию, на что-нибудь уникальное.
9. Mycroft запустится, и вы увидите большой блок фиолетового текста.

Итак, вы настроили и запустили Mycroft. Он может записывать и воспроизводить ваш голос. Еще вы слышали, как он произносит тестовое слово. Пришло время посмотреть, на что он способен.

## Начало использования Mycroft

Сначала мы познакомимся с системой Mycroft, а затем попробуем с ним поговорить. Начнем с интерфейса отладки – клиента Mycroft, который показывает пользователю, что происходит в системе.

### Клиент Mycroft

Когда вы подключитесь к Mycroft, то увидите экран, показанный на рис. 15.3.

```

Log Output:                                0-10 of 10
===== mycroft-core 20.8.0 =====
Establishing Mycroft Messagebus connection...
Connected to Messagebus!
mycroft.session:get:74 | New Session Start: 2c486244-2370-4032-bea2-1f53c81384fa
~~~~0 | 745 | __main__:handle_wakeword:67 | Wakeword Detected: hey mycroft
~~~~nd/start_listening.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Stereo
~~~~24 | INFO | 745 | __main__:handle_record_begin:37 | Begin Recording...
~~~~53.807 | INFO | 745 | __main__:handle_record_end:45 | End Recording...
~~~~45 | __main__:handle_utterance:72 | Utterance: ['what is the weather today']
12:01:54.954 | INFO | 739 | WeatherSkill | Forecast for now
^--- NEWEST ---^

History ===== Log Output Legend ===== Mic Level ===
what is the weather today          DEBUG output
>> It's currently a clear sky and 25 skills.log, other
degrees.                          voice.log
>> Today's forecast is for a high of
27 and a low of 15.

----- 744.00

Input (':' for command, Ctrl+C to quit) =====399 *
>

```

Рис. 15.3. Клиентский интерфейс Mycroft

Скриншот, показанный на рис. 15.3, – это клиент Mycroft. Он позволяет вам видеть, что происходит в системе (для того чтобы Mycroft принимал голосовые команды, подключаться к нему не обязательно). В правом верхнем углу показано, сколько сообщений есть в системе и сколько вы можете видеть. Вы видите сообщение **0** из **10**, а всего их **10**.



В основной части в середине показаны сообщения. *Красные и фиолетовые* – сообщения от системы и плагинов Mycroft. Если вы видите много *фиолетовых* сообщений, это значит, что устанавливаются плагины и обновления, поэтому вам придется немного подождать. *Зеленые* сообщения показывают, что Mycroft взаимодействует с пользователем. Такие сообщения появляются, когда он обнаруживает слово для пробуждения, начинает или заканчивает запись. Также они показывают, как Mycroft обработал вашу команду. Просмотр сообщений помогает в ситуациях, когда Mycroft не отвечает. Так вы можете проверить, принял ли он слово для пробуждения и соответствует ли принятое им высказывание тому, что вы хотели сказать.

Ниже слева находится история. В ней содержатся обработанные команды (выделено *синим* цветом). Диалог, который произносит Mycroft, выделен *желтым* цветом. Из динамика вы будете слышать текст *желтого* цвета, но его воспроизведение может занять некоторое время, если Mycroft занят. Правее показана легенда, где цвета текста сопоставляются с файлом журнала. Еще правее находится измеритель уровня громкости микрофона, и, если Mycroft не занят и вы молчите, он будет перемещаться вверх и вниз, улавливая шум в помещении. Обратите внимание, что посторонние шумы могут негативно влиять на работу голосового помощника.

В нижней части экрана находится область ввода команд для Mycroft.

Подождите, пока система завершит все установки (обычно 30–40 мин). Если система не отвечает – она не зависла, а устанавливает и компилирует дополнительные компоненты.

Mycroft сообщит вам, что необходимо выполнить синхронизацию с помощью `mycroft.ai`. Зарегистрируйте устройство, используя вывод команды (вы можете сделать это во время установки). Для этого вам нужно создать учетную запись Mycroft (или войти, если это второе устройство / вторая попытка). Препятствие чем продолжить, дождитесь завершения процесса.

Когда вы синхронизируете Mycroft и он завершит установку, можно начинать взаимодействовать с ним.

## Взаимодействие с Mycroft

Теперь вы можете поговорить со своим голосовым помощником.

1. Сначала, чтобы привлечь его внимание, скажите «*Hey Mycroft*». Если помощник готов (и еще не занят), он издаст звуковой сигнал. При разговоре вы должны стоять примерно в метре от микрофонов на Raspberry Pi. Он может ответить «*Please wait a moment while I finish booting up*» («*Пожалуйста, подождите, пока я закончу загрузку*»). Подождите минуту и попробуйте еще раз.
2. Если вы услышали звуковой сигнал, то можете попросить помощника что-нибудь сделать. Можете сказать «*Say hello*» («*Поздоровайся*»), и Mycroft должен ответить «*Hello*» («*Привет*») примерно через 10 с. Постарайтесь говорить четко, проговаривая каждый слог.

Теперь можно повеселиться! Вы можете сократить «*Hey Mycroft*» до просто «*Mycroft*». Также попробуйте следующие команды:



- «Hey Mycroft, what is the weather?» По этой команде помощник будет сообщать вам погоду, используя соответствующий навык. Погода может быть определена неверно из-за местоположения. Настроить его можно на веб-сайте mycroft.ai;
- «Mycroft, what is 23 times 76» («Mycroft, сколько будем 23 умножить на 76?»). Эта команда задействует навык использования Wolfram для решения математических задач;
- «Mycroft, wiki banana» («Mycroft, банан – Wiki») Эта команда запустит навык поиска в Wikipedia, и Mycroft расскажет вам, что он нашел по этому запросу.

Для тренировки разговоров с Mycroft попробуйте все эти команды. Если он ответит «*I don't understand*» («Я не понимаю»), загляните в клиент и посмотрите, что услышал помощник. При необходимости потренируйте произношение команд.

Теперь мы можем создать навык для подключения Mycroft к нашему роботу, но сначала устраним возможные неполадки.

## Устранение неполадок

Если Mycroft не отвечает или не распознает речь, воспользуйтесь следующими советами:

- вы должны находиться достаточно близко / говорить достаточно громко. Чтобы проверить громкость, пойдите в клиент Mycroft и посмотрите, поднимается ли измеритель уровня громкости микрофона выше пунктирной линии;
- проверьте сетевое подключение на Raspberry Pi. Mycroft работает только с доступом к интернету. Узнать больше вы можете в документации для работы с прокси. При слабом интернет-соединении Mycroft может не загрузиться. Исправьте проблему и перезагрузите Pi;
- подключение монитора во время загрузки Pi может привести к возникновению ошибок;
- у Mycroft есть собственная система устранения неполадок. Подробнее о ней вы можете узнать по адресу <https://mycroft.ai/documentation/troubleshooting/> (раздел *Troubleshooting and Known errors*);
- сейчас Mycroft находится в стадии активной разработки, поэтому порядок его установки и запуска может измениться. Всегда используйте драйвер ReSpeaker и новейший образ Picofit.

Теперь, когда Mycroft может распознавать речь и отвечать, необходимо подготовить нашего робота для взаимодействия с ним.

## РАЗРАБОТКА API С ПОМОЩЬЮ FLASK

В этой главе мы реализуем управление роботом с помощью Mycroft. Для этого нам нужно дать нашему роботу возможность получать команды от других систем. **Программный интерфейс приложения (API)** на сервере позволит нам отделить такие системы, чтобы по сети отправлять команды другим системам и получать ответ. Для этого идеально подходит система Flask.

У веб-API есть так называемые конечные точки, к которым другие системы обращаются с запросами и приблизительно сопоставляются с функциями или методами в модуле Python. Позже вы увидите, что мы будем сопоставлять контрольные точки API напрямую с функциями в модуле Python `robot_modes`.

Прежде чем перейти к реализации, давайте посмотрим, как работают API и Mycroft.

## Как работает управления роботом с помощью Mycroft

На рис. 15.4 показано, как пользователь управляет роботом с помощью Mycroft.

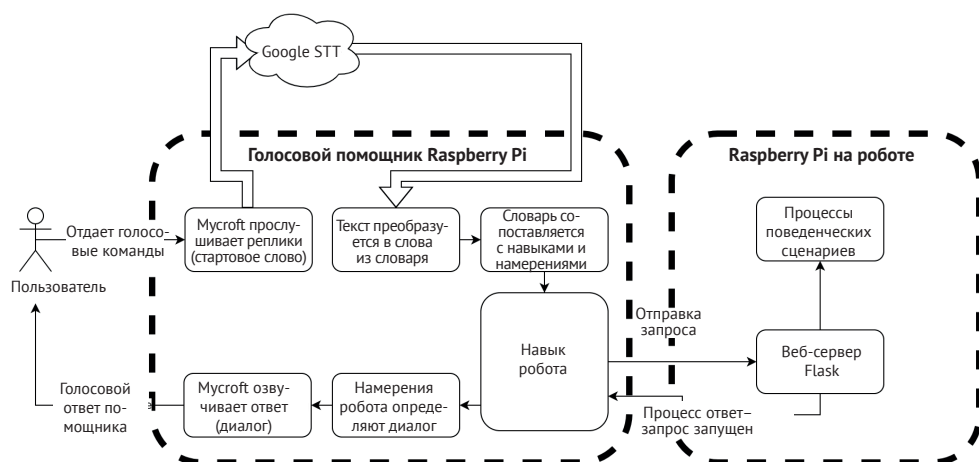


Рис. 15.4. Обзор навыка робота

Поток данных, показанный на рис. 15.4 проходит следующие этапы.

1. В левой части рисунка показан пользователь, отдающий голосовые команды Mycroft.
2. Распознав слово для пробуждения, Mycroft отправляет звук в движок Google STT.
3. Google STT возвращает текст – реплику, которую Mycroft сопоставляет со словарем в навыках/намерениях. Позже мы поговорим об этом подробнее.
4. Это вызывает намерения в навыке робота (его мы создадим позже). Навык отправляет запрос на Raspberry Pi на роботе (показан в правой части рисунка) как запрос на (веб) сервер API управления Flask.
5. Сервер API управления запускает процессы в работе и отправляет ответ о выполнении.
6. Навык робота выбирает диалог, чтобы сообщить о выполнении задачи, а затем отправляет его Mycroft.
7. Mycroft озвучивает этот ответ пользователю.

Теперь переходим к созданию сервера Flask на роботе. Вы уже знакомы с Flask и использовали его для обработки видеоданных (это значит, что нужная библиотека уже установлена).

## Удаленный запуск поведенческого скрипта

Для удаленного запуска мы будем использовать HTTP и веб-сервер. Так мы сможем реализовать разные способы удаленного управления роботом. HTTP отправляет запросы на URL-адрес – сначала идентификатор протокола `http://`; имя хоста сервера `myrobot.local`; путь `/mode/foo`; затем какие-либо дополнительные параметры. Мы будем определять действия робота с помощью пути URL-адреса.

Как и в других системах, мы создаем несколько логических разделов и блоков для обработки различных аспектов:

- код для управления режимами робота, а также для запуска и остановки известных скриптов. Также он может выдавать нам список известных скриптов;
- веб-сервер для обработки запросов по сети.

Начнем с разработки кода для управления режимами.

### Управление режимами робота

Мы можем управлять режимами, запуская и останавливая поведенческие скрипты как подпроцессы. Создадим конфигурацию, с помощью которой сможем сообщать режимы. Она будет сопоставлять имя режима с файлом Python. Обратите внимание, что мы определяем список файлов, а не выводим его. Мы могли бы получить файл, просто взяв название раздела режима/пути, и добавить `.py`, но делать этого не следует по двум причинам:

- такой вариант свяжет нас напрямую с именами скриптов; однако нам необходима возможность менять базовые скрипты для одного и того же имени режима;
- робот – это небезопасная среда, поэтому нельзя допускать произвольного выполнения подпроцессов. Ограничив это, мы немного повысим безопасность.

Перейдем к разработке кода для управления режимами.

1. Создайте файл с именем `robot_modes.py`. В нем будет храниться класс `RobotModes`, который управляет процессами в роботе.
2. Начнем с импортов и первой части определения класса:

```
import subprocess
```

```
class RobotModes(object):
```

3. Далее сопоставляем несколько имен режимов и имен файлов:

```
mode_config = {
    "avoid_behavior": "avoid_with_rainbows.py",
    "circle_head": "circle_pan_tilt_behavior.py",
    "test_rainbow": "test_rainbow.py"
}
```

Имя режима представлено в сокращенном виде (так называемый *слаг*<sup>15</sup>), что делает его понятным как для человека, так и для машины. Обычно

<sup>15</sup> Слаг (англ. slug) – фрагмент текста (обычно из URL), понятный человеку. – Прим. пер.

такие имена намного короче, чем полное описание режима, и состоят только из строчных букв и символов подчеркивания. Исходные имена наших файлов уже похожи на слагги.

4. Помимо фиксированной конфигурации, этот класс управляет запущенными поведенческими сценариями как процессами. Они должны запускаться строго по одному, поэтому нам необходима переменная экземпляра, которая позволит отслеживать текущий процесс и проверять, запущен ли он:

```
def __init__(self):
    self.current_process = None
```

5. Нам нужна возможность проверить, запущен ли какой-либо процесс или уже завершен:

```
def is_running(self):
    return self.current_process and self.current_
process.returncode is None
```

Модуль `subprocesses` позволяет запускать из Python другие процессы и приложения. Так мы проверяем, есть ли у нас текущий процесс, и если да, то запущен ли он. Процессы имеют код возврата, который сообщает об ошибках и завершении. Если процессы все еще активны, вместо кода возврата мы увидим `None`. Так мы можем определить, выполняет ли робот какой-либо процесс в данный момент.

6. Следующая функция запускает процесс. В ее параметры входит имя режима. Функция проверяет, запущен ли процесс, и если нет, то запускает его:

```
def run(self, mode_name):
    if not self.is_running():
        script = self.mode_config[mode_name]
        self.current_process = subprocess.
Popen(["python3", script])
    return True
    return False
```

### Важное примечание

Прежде чем мы запустим новый процесс, необходимо проверить, завершен ли предыдущий скрипт. Если запустить два процесса одновременно, поведение робота будет непредсказуемым. Чтобы этого не произошло, нам нужно быть осторожными.

С помощью `self.mode_config` мы сопоставляем `mode_name` с именем скрипта. Затем посредством `subprocess` запускаем скрипт из Python. Функция `Popen` создает процесс, и код сохраняет дескриптор в `self.current_process`. Если мы запустили режим, метод возвращает `True`, а если он уже был запущен – `False`.

7. Класс должен иметь возможность остановить процесс. Обратите внимание, что класс не будет пытаться остановить процесс, если он не за-

пущен. Для остановки скриптов мы можем использовать сигналы Unix, которые позволят остановить их так, чтобы код `atexit` работал. Отправляем сигнал `SIGINT`, который действует так же, как нажатие комбинации клавиш **Ctrl+C**:

```
def stop(self):
    if self.is_running():
        self.current_process.send_signal( subprocess.
signal.SIGINT)
        self.current_process = None
```

После подачи сигнала устанавливаем для текущего процесса значение `None` (отбрасываем дескриптор).

Теперь у нас есть код для запуска и остановки процессов, который также сопоставляет имена со сценариями. Чтобы голосовой помощник смог его использовать, обернем его в веб-службу,

## Создание сервера API управления с помощью Flask

Ранее мы уже использовали Flask для создания веб-сервера обработки изображений. На этот раз задача гораздо проще.

Мы знаем, что на сервере обработки изображений были кнопки для запуска и остановки. Это означает, что Flask позволяет нам настраивать обработчики ссылок для выполнения задач. Давайте разработаем скрипт для нашего веб-сервиса управления, используя библиотеку Flask и объект `RobotModes`.

Выполните следующие шаги.

1. Создайте скрипт с именем `control_server.py`. Начните с импорта Flask и режимов робота:

```
from flask import Flask
from robot_modes import RobotModes
```

2. Затем создайте приложение Flask, которое будет содержать маршруты и экземпляр объекта `RobotModes`:

```
app = Flask(__name__)
mode_manager = RobotModes()
```

3. Для запуска приложения нам нужен маршрут или контрольная точка API. Она принимает имя режима как часть маршрута:

```
@app.route("/run/<mode_name>", methods=['POST'])
def run(mode_name):
    mode_manager.run(mode_name)
    return "%s running"
```

Возвращаем подтверждение запуска процесса.

4. Для его остановки нам понадобится еще одна контрольная точка API:

```
@app.route("/stop", methods=['POST'])
def stop():
    mode_manager.stop()
    return "stopped"
```

5. Наконец, запускаем сервер:

```
app.run(host="0.0.0.0", debug=True)
```

Теперь с помощью этого приложения мы можем осуществлять голосовое управление.

### Совет

Что такое URL-адрес? Ранее в этой книге мы уже использовали такие адреса при работе с другими веб-сервисами. URL-адрес (Uniform Resource Locator – унифицированный указатель ресурса) определяет, как получить доступ к тому или иному ресурсу. Он начинается со спецификации протокола – в данном случае `http` (поскольку мы используем гипертекстовые данные). Затем идет двоеточие (`:`), две косые черты `//` и имя (или адрес) хоста – сетевой адрес Raspberry Pi, на котором будет находиться ресурс. Поскольку на одном хосте может быть запущено несколько сервисов, мы добавляем номер порта с двоеточием в роли разделителя – `:5000`. После добавляем косую черту `/` и выбираем конкретный ресурс в веб-службе.

Сделаем проверку.

6. Включите робота и скопируйте в его систему файлы `control_server.py` и `robot_modes.py`.
7. Войдите через SSH и запустите сервер управления с помощью команды `python3 control_server.py`. Вы должны увидеть следующее:

```
$ python3 control_server.py
* Serving Flask app "control_server" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production
environment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

8. Теперь создайте еще одно окно `ssh` в Raspberry Pi для Mycroft. Нам нужно проверить, взаимодействуют ли они друг с другом. Чтобы перейти к командной строке Linux (приглашение `$`) нажмите **Ctrl+C** в `pi@picraft.local`.
9. Для тестирования подобных серверов в системах Linux, таких как система Raspberry Pi, часто используется команда `curl`. Она делает запросы к веб-серверам, отправляет/получает данные и отображает результат. Эта команда идеально подходит для проверки API управления, использующих HTTP.

Нам нужно сделать запрос `post`. Введите следующую команду:

```
curl -X POST http://myrobot.local:5000/run/test_rainbow
```

Она должна начать включать и выключать радужное свечение светодиодов, используя код из главы 9. Команда `curl` указывает, что мы выполняем запрос с помощью метода `POST`, затем сообщает URL-адрес с портом, имя хоста робота, команду `run` и имя режима.

10. Остановить скрипт можно с помощью команды `curl -X POST http://myrobot.local:5000/stop`. URL-адрес здесь содержит команду `stop`. В результате радужное свечение должно выключиться.

Обратите внимание, что оба URL-адреса начинаются с `http://myrobot.local:5000/`. В зависимости от имени хоста адрес вашего робота может отличаться. В примерах я показываю базовый URL-адрес для этого сервера управления.

11. Чтобы завершить работу, нажмите **Ctrl+C**.

Теперь вы можете использовать этот сервер при создании поведенческих скриптов для Mycroft, но сначала нужно исправить возможные проблемы.

## Устранение неполадок

Если что-то пошло не так, воспользуйтесь следующими советами:

- при возникновении синтаксических ошибок проверьте код на правильность и попробуйте еще раз;
- проверьте, что робот и устройство, на котором вы тестируете сервер, имеют доступ к интернету;
- при запуске подпроцессов мы используем Python 3. Если вы не ввели 3, результат может быть непредсказуем;
- сервер управления работает на Raspberry Pi 3A+ (на роботе). В командах `curl` указывайте адрес вашего робота;
- проверьте, установили ли вы Flask, как описано в главе 13;
- проверьте, скопировали ли вы в систему робота скрипты сервера управления и режимов робота. Также для проверки у вас должен быть код из главы 9.

После проверки сервера Pi можно выключить. Нам нужно разработать еще один код! В следующем разделе свяжем Pi с Mycroft.

## РАЗРАБОТКА КОДА ДЛЯ ЗАПУСКА MYCROFT НА RASPBERRY PI

Серверная часть робота, предоставляемая системой управления Flask, отлично подходит для создания навыка для Mycroft.

На рис. 15.4 вы видели, что после того, как вы произносите слово для пробуждения и команду, Mycroft передает звук в систему Google STT, которая затем возвращает его в виде текста. Далее Mycroft соотносит текст с содержимым файлов словарей (в соответствии с регионом, в котором вы находитесь) и сопоставляет результат с намерениями в рамках определенного навыка. Затем он вызывает намерение в навыке. Для нашего навыка взаимодействия с роботом мы создадим намерения, которые будут делать сетевые (HTTP) запросы к Flask-серверу управления. Когда Flask-сервер ответит, что он обработал запрос (возможно, поведенческий скрипт будет запущен), навык выберет диалог, чтобы ответить пользователю и подтвердить, что он успешно выполнил запрос или обнаружил проблему.

Мы начнем с простого навыка и намерения, а затем вы сможете расширить их. Для простоты я выбрал тест радужных светодиодов (`test_leds` из главы 9).



Стоит отметить, что из-за времени, затрачиваемого на обработку речи Google, мы не сможем быстро остановить робота голосовой командой. Распознавание голоса занимает некоторое время. Для остановки сервера управления вы можете использовать библиотеку GPIOZero в намерении и обработчик `when_pressed`.

## Разработка намерения

Мы начнем с разработки намерения, а потом займемся словарем. Для создания намерения воспользуемся встроенной в Mycroft библиотекой `adapt`.

1. Создайте папку с именем `my-robot-skill`. В ней мы будем создавать навыки для Mycroft.
2. Создайте в папке основной файл для намерения с именем `__init__.py`. Такое имя файла означает, что Python будет рассматривать всю папку как библиотеку Python, называемую **пакетом**. Начнем с импортов в `my-robot-skill/__init__.py`:

```
from adapt.intent import IntentBuilder
from mycroft import MycroftSkill, intent_handler
from mycroft.util.log import LOG
```

```
import requests
```

Мы импортируем `IntentBuilder` для создания и определения намерений на основе словаря. Класс `MycroftSkill` – это базовый класс для подключения нашего кода к Mycroft. Класс `intent_handler` отмечает, какие части нашего кода являются намерениями, и связывает код с `IntentBuilder`. Импорт `LOG` необходим для записи информации в клиент Mycroft – так мы сможем видеть возникающие проблемы.

Последний импорт, `requests`, – это инструмент для удаленного взаимодействия с управляющим сервером с помощью Python.

3. Далее мы определяем навык из базового класса `MycroftSkill`. Для него нужно установить надкласс и подготовить настройки:

```
class MyRobot(MycroftSkill):
    def __init__(self):
        super().__init__()
        self.base_url = self.settings.get("base_url")
```

Ключевое слово Python `super` вызывает метод из базового класса. Вызываем метод `__init__`, чтобы позволить ему выполнить настройку.

Единственный устанавливаемый здесь параметр – это элемент `base_url` для управляющего сервера на роботе. Он извлекает данные из файла настройки, который мы увидим позже. Я рекомендую разделить конфигурацию и код.

4. Теперь нам необходимо определить намерение. Сделаем это с помощью метода `handle_test_rainbow`, к которому добавим декоратор `@intent_handler`. В Python декорирование оборачивает метод для дальнейшей обработки – в данном случае делает его подходящим для Mycroft:

```
@intent_handler(IntentBuilder(""))
.require("Robot")
```

```
.require("TestRainbow"))
def handle_test_rainbow(self, message):
```

Декоратор `intent_handler` принимает параметры для настройки словаря. Позже мы определим его в файле. Сначала нам нужно провести сопоставление со словарем для `Robot`, а затем для `TestRainbow` (здесь могут подходить несколько фраз).

5. Затем навык должен сделать запрос к роботу с помощью `request.post`:

```
try:
    requests.post(self.base_url + "/run/test_
rainbow")
```

Этот фрагмент кода объявляет URL-адрес в переменной `base_url`, а также инструкцию `run` и режим `test_rainbow`.

6. Нам необходимо, чтобы Mycroft оповестил пользователя о том, что он сообщил роботу:

```
self.speak_dialog('Robot')
self.speak_dialog('TestingRainbow')
```

Метод `speak_dialog` сообщает Mycroft, что он должен выбрать подходящий диалог из соответствующих файлов. Так Mycroft может выбрать что ему сказать.

7. По ряду причин этот запрос может завершиться ошибкой, поэтому в позапрошлом фрагменте кода мы использовали блок `try`. Для обработки запроса и воспроизведения диалога нам нужен блок `except`. Также мы записываем исключение в консоль Mycroft с помощью `LOG`:

```
except:
    self.speak_dialog("UnableToReach")
    LOG.exception("Unable to reach the robot")
```

Мы обрабатываем множество типов ошибок, таких как `Unable to reach the robot`, при этом не проверяя код результата от сервера полностью (мы проверяем только то, связался ли навык голосового помощника с роботом).

8. Затем этот файл должен предоставить функцию `create_skill` вне класса, которую Mycroft ожидает найти в файлах навыков:

```
def create_skill():
    return MyRobot()
```

Код является одной из частей системы, но перед использованием нам нужно выполнить его конфигурацию.

## Файл настроек

Наше намерение началось с загрузки настройки. Мы поместим настройку в `my-robot-skill/settingsmeta.json` и определим базовый URL-адрес для сервера управления.

Обратите внимание, что здесь нужно использовать имя хоста/адрес `Raspberry Pi` вашего робота (они могут отличаться от представленных в приме-

рах). Код в этом файле достаточно объемный, но он позволяет настроить URL-адрес позже, если это необходимо:

```
{
  "skillMetadata": {
    "sections": [
      {
        "name": "Robot",
        "fields": [
          {
            "name": "base_url",
            "type": "text",
            "label": "Base URL for the robot
control server",
            "value": "http://myrobot.local:5000"
          }
        ]
      }
    ]
  }
}
```

Теперь, когда мы установили базовый URL-адрес, нам нужно настроить Mycroft для загрузки навыка.

### Файл требований

Наш навык использует библиотеку requests. Нам необходимо указать Mycroft ожидать ее. В Python это можно сделать с помощью файлов требований. В my-robot-skill/requirements.txt введите следующее:

```
requests
```

Этот файл используется не только для Mycroft, но и для других систем на Python. Он устанавливает необходимые приложения библиотеки.

Теперь нам нужны словари.

### Файлы словарей

Чтобы определить словари, нам нужно создать их файлы. Мы поместим их в папку по следующему пути: my-robot-skill/vocab/<IETF language and locale>. Язык/локаль<sup>16</sup> позволяет определить словарь для таких вариантов как en-us (американский английский) и zh-cn (китайский упрощенный). На момент написания этой книги en-us является наиболее поддерживаемым языком для Mycroft. Сообщество активно работает над расширением поддержки других языков.

Каждое намерение определяется с помощью одной или нескольких частей словаря, соответствующих файлам. Файлы словарей содержат строки, представляющие варианты формулировки предполагаемой реплики. Благодаря этому человек может формулировать их по-разному. Иногда бывает, что голосовой помощник отвечает не совсем то, что нужно пользователю. Чтобы этого

<sup>16</sup> Локаль (англ. locale) – набор данных, включающих правила, характерные для определенного языка и конкретной географической области (региональные настройки). – Прим. пер.

не случилось, в файлы словарей необходимо добавить схожие по значению (но разные по формулировке) фразы.

Для нашего намерения нам нужны файлы словарей – файл синонимов для `robot` и файл синонимов для `TestRainbow`.

1. В папке `my-robot-skill` создайте папку `vocab`, а в ней – еще одну папку `en-us`.
2. В ней создайте файл по следующему пути `my-robot-skill/vocab/en-us/robot.voc` (и с соответствующим именем).
3. Добавьте несколько фраз, с помощью которых можно *попросить робота сделать что-нибудь*:

```
robot
my robot
ask robot to
tell the robot to
```

Теперь, когда обработчик намерений столкнется со словом `robot`, Mycroft будет пытаться сопоставить с ним эти фразы.

4. Создадим словарь для намерения включить радужное свечение. Поместите файл словаря (с соответствующим именем) по следующему пути: `my-robot-skill/vocab/en-us/TestRainbow.voc`:

```
test rainbow
test the leds
deploy rainbows
turn on the lights
```

### Важное примечание

Обратите внимание, что заглавные буквы в имени файла словаря должны соответствовать имени в сборщике намерений. Я воспользовался соглашением о написании неразделяемых элементов словаря с заглавной буквы.

При проверке рано или поздно вы скажете фразу, которой нет в этих списках. В этом случае Mycroft ответит «*Sorry, I don't understand*» («Извините, я не понимаю вас»), и вы сможете добавить новую фразу.

### Файлы диалогов

Теперь мы определим фразы, которыми Mycroft будет нам отвечать. Для нашего намерения требуются три фразы. Мы реализуем структуру, аналогичную файлам словарей, и поместим их по следующему пути: `my-robot-skill/dialog/en-us`. Перейдем к созданию файлов.

1. В папке `my-robot-skill` создайте папку `dialog`, а в ней – `en-us`.
2. В папке по пути `my-robot-skill/dialog/en-us/Robot.dialog` создайте файл. В него добавьте несколько фраз:

```
The Robot
Robot
```

3. В этой же папке создайте файл `TestRainbow.dialog`:

```
is testing rainbows.
is deploying rainbows.
is starting rainbows.
is lighting up.
```

4. Поскольку у нас есть обработчик ошибок, создайте файл `UnableToReach.dialog`:

```
Sorry I cannot reach the robot.
The robot is unreachable.
Have you turned the robot on?
Is the control server running on the robot?
```

Мы определили несколько возможных диалогов, и теперь для ответа Mycroft будет случайным образом выбирать один из них (при этом стараясь не повторяться). Мы создали файлы словарей и файлы диалогов, далее кратко резюмируем, что у нас получилось.

### Папка навыка

Сейчас папка навыка должна выглядеть так, как показано на рис. 15.5.

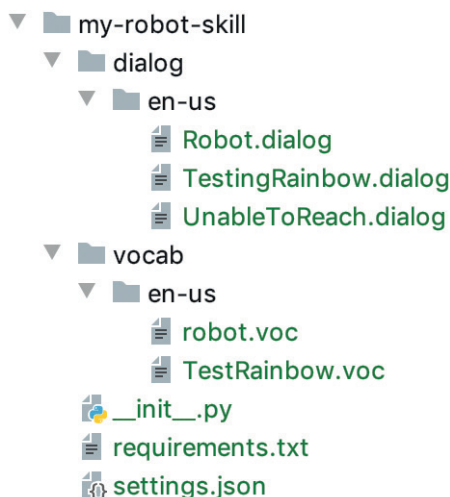


Рис. 15.5. Скриншот папки навыка

На рис. 15.5 мы видим структуру папки навыка `my-robot-skill`. В ней есть папка `dialog`, имеющая вложенную папку `en-us`, в которой хранятся три файла диалогов. Ниже находится папка `vocab`, в которой есть папка `en-us`, содержащая два файла словарей. Под папкой `vocab` находится файл `__init__.py`. Далее идут файл с требованиями для установки Mycroft и файл настроек. Ух ты, мы создали столько всего – оно того стоит!

Теперь нужно загрузить все это в память робота.

1. С помощью SFTP (FileZilla) загрузите папку в Pi для Mycroft и поместите ее в директорию `mycroft/skills`.

2. Mycroft автоматически загрузит этот навык. Во время установки в клиенте вы увидите фиолетовый текст.
3. Если вам нужно обновить код, загрузите новые файлы в это же место, и Mycroft перезапишет их.
4. Все проблемы, связанные с загрузкой или использованием навыка, будут отображаться в клиенте Mycroft. Также эти данные хранятся в `/var/log/mycroft/skills.log` – инструмент Linux `less` позволяет просматривать вывод журнала. Чтобы перейти в конец файла, нажмите **Shift+G**, а чтобы увидеть вывод, введите `/myrobot`. Еще возникающие проблемы можно увидеть в `tail -f /var/log/mycroft/skills.log`. Для остановки нажмите **Ctrl+C**.
5. Включите робота, войдите через `ssh` и запустите сервер управления с помощью команды `python3 control_server.py`.
6. Теперь вы можете протестировать навык. Для этого скажите Mycroft: «*Tell the robot to turn on the lights*» («Скажи роботу включить светодиоды»).
7. Mycroft подаст звуковой сигнал (чтобы показать пользователю, что он готов) и после преобразования речи в текст отправит команду `/run/test_rainbow` на сервер управления. Mycroft ответит пользователю одной из диалоговых фраз, например «*The robot is testing rainbows*» («Робот проверяет радужное свечение»). После этого вы увидите, как включатся светодиоды.

## Устранение неполадок

Если намерение не выполняется, воспользуйтесь следующими советами:

- сначала проверьте синтаксис и отступы в коде на Python;
- Raspberry Pi робота и голосового помощника должны находиться в одной сети. Я обнаружил проблему при использовании расширителя диапазона Wi-Fi, в таком случае вместо `myrobot.local` нужно использовать IP-адрес. Настроить это можно в файле `settingsmeta.json`;
- убедитесь, что вы полностью скопировали структуру (словари, диалоги, файлы `settingsmeta.json` и `__init__.py`) в папку `/opt/mycroft/skills` на Raspberry Pi голосового помощника;
- если вы ввели неверные настройки, поменяйте их на веб-странице <https://account.mycroft.ai/skills> (найдите навык `My Robot` и внесите изменения). Сохраните изменения. После этого вам нужно будет перезапустить Mycroft или подождать несколько минут, чтобы они вступили в силу;
- говорите Mycroft только те фразы, которые находятся в его файлах словарей, иначе он не сможет распознать ваши слова;
- при возникновении проблем с распознаванием голоса вы можете вводить команды в консоль Mycroft.

Первое намерение создано и готово к работе! Вы смогли поговорить с голосовым помощником, он передал команды роботу, и тот включил светодиоды. Сейчас единственный способ выключить их – это неудобная команда `curl`. Исправим это, добавив еще одно намерение.

## Создание еще одного намерения

После проделанной работы создание намерения для навыка остановки покажется вам относительно простой задачей. Для этого мы будем использовать другую контрольную точку на сервере управления.

### Словарь и диалог

Для нового намерения нужно добавить словарь и диалог.

1. Создадим словарь `stop` и поместим его в `my-robot-skill/vocab/en-us/stop.voc`:

```
stop
cease
turn off
stand down
```

2. Для того чтобы Mycroft мог сообщить пользователю, что светодиоды выключены, добавим файл диалога в каталог `my-robot-skill/dialog/en-us/stopping.dialog`:

```
is stopping.
will stop.
```

При необходимости вы можете добавить в эти файлы больше фраз.

### Добавляем код намерения

Теперь нам нужно добавить код намерения в наш навык:

1. Поместим следующий код в класс `MyRobot` в каталог `my-robot-skill/__init__.py`:

```
@intent_handler(IntentBuilder(""))
    .require("Robot")
    .require("stop"))
def handle_stop(self, message):
    try:
        requests.post(self.base_url + "/stop")
        self.speak_dialog('Robot')
        self.speak_dialog('stopping')
    except:
        self.speak_dialog("UnableToReach")
        LOG.exception("Unable to reach the robot")
```

Этот код почти идентичен коду намерения для включения радужного свечения со словарем `stop`, именем обработчика (оно может быть любым, но не должно совпадать с именем другого обработчиком) и контрольной точкой URL-адреса.

Для такого кода можно выполнить рефакторинг. Как вы помните, рефакторинг – это изменение внешнего вида кода, не влияющее на его функции. Такой подход облегчает работу с общими/повторяющимися фрагментами кода и делает его удобным для восприятия человеком. Оба намерения содержат один и тот же блок попытка/перехват (`try/catch`) и похожий диалог (с небольшими отличиями).



2. В этот же файл добавьте следующий фрагмент кода:

```
def handle_control(self, end_point, dialog_verb):
    try:
        requests.post(self.base_url + end_point)
        self.speak_dialog('Robot')
        self.speak_dialog(dialog_verb)
    except:
        self.speak_dialog("UnableToReach")
        LOG.exception("Unable to reach the robot")
```

Это код для общего обработчика. Он принимает параметр `end_point` и использует его в своем запросе. Запрос принимает параметр `dialog_verb` и помещает его после `Robot`. Также здесь находятся все остальные диалоги и обработчики ошибок, которые мы видели ранее.

3. Оба намерения теперь стали немного проще. Замените их на следующий фрагмент кода:

```
@intent_handler(IntentBuilder(""))
    .require("Robot")
    .require("TestRainbow"))
def handle_test_rainbow(self, message):
    self.handle_control('/run/test_rainbow',
        'TestingRainbow')

@intent_handler(IntentBuilder(""))
    .require("Robot")
    .require("stop"))
def handle_stop(self, message):
    self.handle_control('/stop', 'stopping')
```

Теперь добавлять новые намерения стало проще, поскольку можно повторно использовать функцию `handle_control`.

### Запуск системы с новым намерением

Теперь нужно загрузить структуру папок заново, так как мы внесли изменения в файлы `vocab`, `dialog` и `__init__`. После этого Microsoft автоматически перезапишет измененный навык (или вы увидите сообщения о соответствующих ошибках), так что он сразу будет готов к использованию.

Скажите: *«Microsoft, tell the robot to stop»* (*«Майкрософт, пусть робот остановится»*).

Вы добавили в систему второе намерение, определив необходимые словари и диалоги. Также вы выполнили рефакторинг кода, устранив повторяющиеся фрагменты. Теперь у вас есть отправная точка для управления роботом с помощью голосового помощника.

## Выводы

В этой главе вы узнали об основных понятиях Microsoft, технологии преобразования речи в текст, словах для пробуждения, намерениях, навыках, репликах, словарях и диалогах. Вы приняли решение, куда установить микрофоны и динамики и должны ли они находиться непосредственно на работе.

Затем вы узнали, как установить динамики и микрофоны на Raspberry Pi, и подготовили программное обеспечение, чтобы Pi мог использовать их. Вы установили Picroft – среду Mycroft для Raspbian (программное обеспечение голосового помощника).

Далее вы поэкспериментировали с Mycroft и посмотрели, как он реагирует на различные голосовые команды и регистрирует их в своей базе.

Затем вы увидели, как подготовить робота к работе с внешними агентами (в данном случае с голосовым помощником), и узнали, как управлять ими с помощью Flask API. Вы создали несколько навыков, которые взаимодействуют с роботом, и в дальнейшем сможете создать еще множество таких навыков.

В следующей главе мы вернемся к IMU из главы 12 и расширим его применение. Мы откалибруем датчики, а затем объединим их и создадим для робота новый сценарий поведения, который заставит его всегда поворачивать на север.

## Задание

Попробуйте выполнить следующие задания, чтобы извлечь больше пользы из этой главы и расширить свой опыт:

- попробуйте установить другие навыки для Mycroft (их можно найти на официальном сайте) и поэкспериментировать с ними. Например: «*Hey Mycroft, install pokemon*» («Эй, Майкрофт, установи Покемонов»);
- система режимов робота имеет один недостаток. Она предполагает, что процесс, который пользователь попросил остановить, действительно останавливается. Должна ли система подождать и проверить код возврата, чтобы убедиться, что процесс остановлен?
- в качестве альтернативного способа реализации режимов робота вы можете обновить все варианты поведенческих сценариев, чтобы завершить работу корректно. После этого их можно будет импортировать, а не запускать в подпроцессах. Насколько это будет сложно?
- если во время проверки взаимодействия Mycroft и робота вы поняли, что в словаре не хватает каких-либо фраз, то можно пополнить его. Добавьте наиболее подходящие, по вашему мнению, фразы. То же самое вы можете сделать и с диалогами;
- добавьте в навык больше намерений, например для обхода стен. Также вы можете добавить намерение для остановки робота, но ввиду долгого времени отклика оно будет работать далеко не идеально;
- как можно использовать RGB-светодиоды на плате ReSpeaker 2-Mics Pi HAT? Подробнее об этом вы можете узнать в проекте [https://github.com/respeaker/mic\\_hat](https://github.com/respeaker/mic_hat).

Все перечисленные идеи открывают простор для дальнейшего исследования рассмотренных концепций. Также в этом вам помогут дополнительные материалы.

## ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

Обратитесь к следующим источникам, чтобы узнать больше:

- в книге «*Raspberry Pi Robotic Projects*» автора *Dr. Richard Grimmett* (*Д-р Ричард Гримметт*), выпущенной издательством *Packt Publishing*, есть глава, посвященная реализации речевого ввода и вывода;
- книга «*Voice User Interface*» *Projects* автора *Henry Lee* (Генри Ли) от *Packt Publishing* полностью посвящена теме голосовых интерфейсов. В ней рассказывается, как создавать чат-боты и приложения с помощью голосовых помощников Alexa и Google Home;
- «*Mycroft AI – Introduction Voice Stack*» – это технический документ от Mycroft AI, в котором вы можете найти подробную информацию о работе стека Mycroft и его компонентов;
- у Mycroft есть большое сообщество, члены которого могут поддерживать и обсуждать технологию по адресу <https://community.mycroft.ai/>. Я рекомендую ознакомиться здесь с информацией об устранении неполадок. Mycroft находится на стадии активной разработки и имеет множество особенностей и новых функций, что создает пространство для обмена навыками;
- Компания Seeed Studio, создатель ReSpeaker 2-Mics Pi HAT, размещает документацию и код для этого устройства (а также для более крупных моделей с четырьмя и шестью микрофонами) по адресу <https://github.com/respeaker/seeed-voicecard>.

# Глава 16

## Погружаемся глубже в работу IMU

В главе 12 вы научились считывать данные датчиков **инерциального измерительного модуля (IMU)**. Теперь вы знаете немного больше об обработке показаний датчиков, а также об использовании математических операций и конвейеров для принятия решений.

В этой главе вы узнаете, как выполнять калибровку датчиков IMU и объединять их показания. На основе полученных знаний вы разработаете поведенческий сценарий, полностью основанный на ориентировании робота в пространстве. Вместе с этим мы рассмотрим различные алгоритмы, позволяющие повысить точность/скорость получения и корректность данных.

В завершение главы ваш робот будет способен определять абсолютное положение и выводить эти данные на экран. Также вы объедините этот сценарий с поведенческими сценариями для **пропорционально-интегрально-дифференцирующих регуляторов (ПИД-регуляторов)**.

В этой главе мы рассмотрим следующие темы:

- разработку кода для виртуального робота;
- определение параметров вращения с помощью гироскопа;
- измерение углов тангажа и крена с помощью акселерометра;
- определение направления с помощью магнитометра;
- приблизительное определение направления робота с помощью магнитометра на практике;
- объединение показаний датчиков для ориентирования робота;
- управление роботом на основе показаний IMU.

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам потребуются:

- робот из главы 14;
- код для робота из главы 14 (его вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter14>);

- код для IMU из главы 12 (его вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter12>);
- пространство без магнитов, в котором будет перемещаться робот;
- магнитный компас.

Полный код из этой главы вы можете найти по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter16>.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/2Lztw00>.

## РАЗРАБОТКА КОДА ДЛЯ ВИРТУАЛЬНОГО РОБОТА

Сначала нам нужно научиться определять ориентацию робота в пространстве. Удобнее всего будет показать это на его трехмерной модели. Эта часть главы основывается на разделе «Системы координат и вращения» из главы 12. В текущем разделе мы создадим простую модель нашего робота в VPython.

### Создание модели робота в VPython

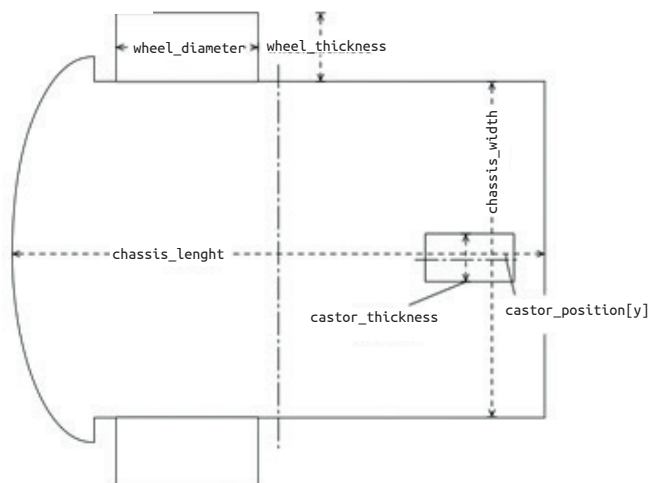
При моделировании робота мы будем использовать фигуры, известные как **графические примитивы**. Они имеют следующие свойства: положение, поворот, размер и цвет. Параметры высоты и ширины соответствуют системе координат VPython (см. рис. 12.14 в главе 12), поэтому положение компонентов должно соответствовать системе координат осей корпуса робота.

Нам нужно измерить параметры робота, как показано на рис. 16.1. Сначала выполните большие измерения, а затем на их основе измерьте более мелкие параметры.

На рис. 16.1 показано, что нужно измерить. Здесь представлено два взгляда (сверху и слева), что позволяет захватить все аспекты. Мы измеряем ширину и высоту основания (шасси) робота – обратите внимание, что для этого мы представили его в виде прямоугольника. Также нужно измерить диаметр и положение ведущих и поворотного колеса. Измерьте или приблизительно рассчитайте параметры для вашего робота. Для нашей задачи приблизительных расчетов будет достаточно. Положение опирается на центр шасси.

Разработаем код для создания базовых форм.

## ВИД СВЕРХУ



## ВИД СЛЕВА

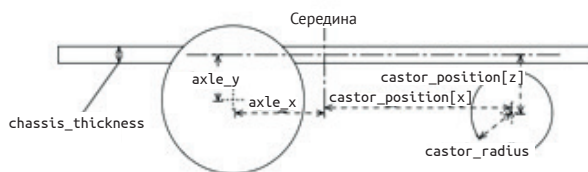


Рис. 16.1. Измерение параметров для виртуального робота

1. Создайте файл с именем `virtual_robot.py`, добавьте импорт `vpython` и представление робота:

```
import vpython as vp
from robot_pose import robot_view
```

2. Мы поместим виртуального робота в функцию, готовую для использования в нескольких разных скриптах:

```
def make_robot():
```

3. Помещаем результаты измерений, показанных на рис. 16.1, в переменные. У меня все измерения представлены в **миллиметрах (мм)**. Код показан в следующем фрагменте:

```
chassis_width = 155
chassis_thickness = 3
chassis_lenght = 200
wheel_thickness = 26
wheel_diameter = 70
axle_x = 30
axle_z = -20
castor_position = vp.vector(-80, -6, -30)
castor_radius = 14
castor_thickness = 12
```

4. Основание представляет собой прямоугольник с положением по умолчанию (0, 0, 0). Код показан в следующем фрагменте:

```
base = vp.box(length=chassis_length,
             height=chassis_thickness,
             width=chassis_width)
```

5. Повернем этот прямоугольник в соответствии с системой координат робота (на 90° вокруг оси x, подняв по оси z):

```
base.rotate(angle=vp.radians(90),
            axis=vp.vector(1, 0, 0))
```

6. Колеса мы представляем в виде двух цилиндров. Расстояние от каждого колеса до середины основания составляет примерно половину ширины шасси. Рассчитаем положение колес по оси y:

```
wheel_dist = chassis_width/2
```

7. Настраиваем положение колес так, чтобы оно совпадало с концами осей двигателя. У левого колеса есть координата y. Параметр -wheel\_dist перемещает ее к левой стороне основания:

```
wheel_l = vp.cylinder(radius=wheel_diameter/2,
                     length=wheel_thickness,
                     pos=vp.vector(axle_x, -wheel_dist, axle_z),
                     axis=vp.vector(0, -1, 0))
```

Ось VPython cylinder показывает направление. Чтобы направить влево, мы устанавливаем координату y равной -1.

8. Теперь настраиваем правое колесо с положительным значением wheel\_dist.; устанавливаем y равной 1, чтобы направить вправо:

```
wheel_r = vp.cylinder(radius=wheel_diameter/2,
                     length=wheel_thickness,
                     pos=vp.vector(axle_x, wheel_dist, axle_z),
                     axis=vp.vector(0, 1, 0))
```

9. Заднее поворотное колесо мы также представляем в виде цилиндра, как показано в следующем фрагменте кода:

```
castor = vp.cylinder(radius=castor_radius,
                    length=castor_thickness,
                    pos=castor_position,
                    axis=vp.vector(0, 1, 0))
```

10. Теперь объединяем все компоненты в единый составной трехмерный объект:

```
return vp.compound([base, wheel_l, wheel_r, castor])
```

11. Для проверки создадим небольшой раздел main. Этот фрагмент кода всегда проверяет, запущен ли он напрямую, так что код не будет запускаться после импорта виртуального робота как библиотеки:

```
if __name__ == "__main__":
```

12. Определим представление робота. Для этого укажем, что мы смотрим на него прямо:

```
robot_view()
```



13. Чтобы показать, как расположены компоненты, добавляем оси:

```
x_arrow = vp.arrow(axis=vp.vector(200, 0, 0),
color=vp.color.red)
y_arrow = vp.arrow(axis=vp.vector(0, 200, 0),
color=vp.color.green)
z_arrow = vp.arrow(axis=vp.vector(0, 0, 200),
color=vp.color.blue)
```

14. Теперь создаем модель робота:

```
make_robot():
```

15. Загрузите и протестируйте этот код с помощью команды `vpython virtual_robot.py`.
16. Чтобы увидеть виртуального робота в браузере, подключитесь к порту робота 9020. Результат будет выглядеть так, как показано на рис. 16.2.

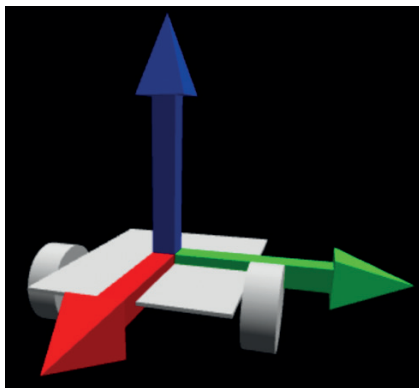


Рис. 16.2. Скриншот трехмерного виртуального робота в VPython

На рис. 16.2 мы видим ось  $x$  красного цвета, направленную вперед, ось  $y$  зеленого цвета, направленную вправо, и ось  $z$  синего цвета, направленную вверх. Эта система координат опирается на правило правой руки. На рисунке мы видим модель виртуального робота с колесами по бокам (вид спереди). Сейчас модель выглядит достаточно просто (серый прямоугольник), но для наших дальнейших экспериментов этого достаточно.

17. Нажмите правую кнопку мыши и потяните в сторону, чтобы посмотреть на робота с другой стороны. Колесиком мыши можно увеличивать или уменьшать масштаб. На рис. 16.3 мы видим заднее поворотное колесо робота.

На рис. 16.3 показан вид виртуальной модели слева.

Когда закончите, закройте вкладку браузера, и нажмите **Ctrl+C**, чтобы завершить программу. Проверим, все ли работает.

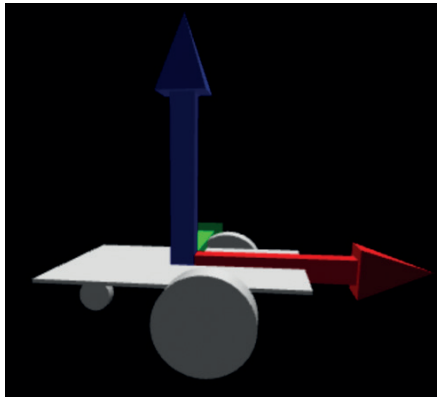


Рис. 16.3. Виртуальная модель робота с другой стороны

### Устранение неполадок

Если что-то пошло не так, воспользуйтесь следующими советами.

1. При возникновении ошибки, сообщающей, что **модуль vpython не найден (no such module vpython)**, проверьте, установили ли вы VPython. Установка описана в разделе «Считывание данных датчика температуры» в главе 12. Чтобы код из текущей главы работал, вам нужен полный код из главы 12.
2. При возникновении ошибки, сообщающей о том, что **не найдена команда vpython (no such command vpython)**, проверьте выполнили ли вы действия, описанные в разделе «Упрощение командной строки VPython» в главе 12. Для отображения результата нужен псевдоним (ярлык) VPython.
3. При возникновении синтаксических ошибок, проверьте, нет ли в коде опечаток.
4. Если результат не отображается в браузере (и вы проверили шаг 1), проверьте, что для вашего робота используется порт 9020 (у меня <http://myrobot.local:9020>).
5. Будьте терпеливы – для запуска VPython может потребоваться несколько минут.

Теперь, когда виртуальный робот готов, мы можем поэкспериментировать. Сначала вернемся к гироскопу и заставим робота на экране повторять движения реального робота.

## ОПРЕДЕЛЕНИЕ ПАРАМЕТРОВ ВРАЩЕНИЯ С ПОМОЩЬЮ ГИРОСКОПА

Мы уже знаем, как получать необработанные данные гироскопа, но, чтобы использовать их более эффективно, необходимо выполнить две операции: откалибровать гироскоп и суммировать его показания (как показано на рис. 16.4).

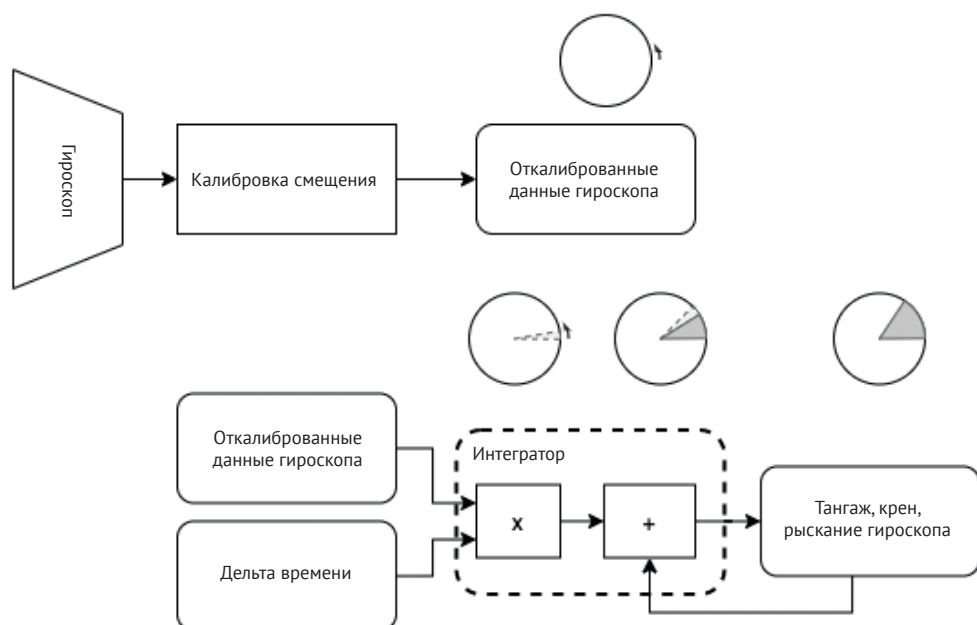


Рис. 16.4. Поток данных гироскопа

На рис. 16.4 показан поток данных гироскопа (позже в этом разделе мы рассмотрим это подробнее). В верхней части рисунка показана первая операция: данные гироскопа проходят этап калибровки смещений (для устранения ошибок). В результате мы получаем откалиброванные данные гироскопа с коэффициентом изменения в градусах в секунду (по оси) – на рисунке этот шаг показан как окружность со стрелочкой. Так гироскоп выполняет относительные измерения.

В нижней части рисунка показана вторая операция, объединяющая дельту времени и данные откалиброванного гироскопа. Нам нужно **суммировать** их, чтобы найти абсолютное значение. Сумматор умножает входное значение на дельту времени и добавляет полученный результат к предыдущему. В этом случае мы умножаем коэффициент изменения данных гироскопа на дельту времени и получаем значение перемещения за определенный временной промежуток – на рисунке это показано символом умножения в рамке. Над этой частью рисунка мы видим окружность с небольшим сектором, выделенным пунктирными линиями. Он показывает, насколько изменилось положение робота.

Затем добавляем последнее значение для этой оси – на рисунке выглядит как плюс в рамке. Над ним – окружность, на которой сегмент, выделенный серым, показывает текущее положение, а сегмент, выделенный пунктирными линиями, – новое измерение. Вместе они составляют общее значение для этой оси, показанное на следующей окружности (большой серый сегмент). Система передает результат измерения углов тангажа, крена или рыскания в следующий цикл.

Прежде чем мы реализуем такую систему, необходимо устранить ошибки гироскопа.

## Калибровка гироскопа

С производства **микроэлектромеханические гироскопы (МЭМС-гироскопы)** могут поставляться с небольшими погрешностями, из-за которых их показания могут различаться. Эти погрешности мешают нам при интеграции.

Мы разработаем код для обнаружения и компенсации таких погрешностей, т. е. выполним **калибровку**. Для этого осуществим следующие шаги.

1. Создайте файл с именем `calibrate_gyro.py`.
2. Нам понадобится импортировать VPython для работы с векторами и библиотеку `time`, чтобы задать небольшую паузу; также нужно настроить IMU, как показано в следующем фрагменте кода:

```
from robot_imu import RobotImu
import time
import vpython as vp
imu = RobotImu()
```

3. Создадим векторы, в которых будут храниться минимальное и максимальное значения гироскопа:

```
gyro_min = vp.vector(0, 0, 0)
gyro_max = vp.vector(0, 0, 0)
```

4. Для цикла нам нужен набор показаний с течением времени:

```
for n in range(500):
    gyro = imu.read_gyroscope()
```

5. Для калибровки нам нужно провести измерения и получить минимальное и максимальное значения для каждой оси. Функция Python `min` возвращает меньшее из двух переданных ей значений:

```
gyro_min.x = min(gyro_min.x, gyro.x)
gyro_min.y = min(gyro_min.y, gyro.y)
gyro_min.z = min(gyro_min.z, gyro.z)
```

6. Используя функцию Python `max` (по такому же принципу), возвращаем максимальные значения:

```
gyro_max.x = max(gyro_max.x, gyro.x)
gyro_max.y = max(gyro_max.y, gyro.y)
gyro_max.z = max(gyro_max.z, gyro.z)
```

7. Средние значения показывают, насколько далеко мы находимся относительно нуля. Чтобы рассчитать их, нужно сложить векторы и разделить сумму на 2:

```
offset = (gyro_min + gyro_max) / 2
```

8. Перед началом нового цикла устанавливаем небольшую паузу:

```
time.sleep(.01)
```

9. Чтобы мы могли использовать эти значения, отправляем их на печать:

```
print(f"Zero offset: {offset}.")
```

10. Код готов к запуску. Загрузите его и запустите с помощью Python 3. Во время выполнения программы робот должен оставаться на ровной и устойчивой поверхности.
11. Последние строки вывода в консоли будут выглядеть примерно так, как показано в следующем фрагменте кода:

```
pi@myrobot:~ $ python3 calibrate_gyro.py
Zero offset: <-0.583969, 0.675573, -0.530534>.
```

Мы вычислили среднее значение изменения показаний гироскопа в условиях, когда робот неподвижен. Мы рассчитали его как смещение для каждой оси. Вычитая этот результат из других измерений, мы компенсируем непрерывные ошибки гироскопа. Поместим этот код в нужное место.

1. Создайте файл `imu_settings.py`.
2. Импортируйте тип данных `vector`, а затем настройте калибровочные показания. Этот код нужно будет запустить только один раз (и еще в случае, если вы замените IMU). Здесь используйте показания ваших датчиков. Запустите следующий код:

```
from vpython import vector
gyro_offsets = vector(-0.583969, 0.675573, -0.530534)
```

3. Теперь нам нужно обновить класс `RobotImu` для обработки смещений. Откройте файл `robot_imu.py`.
4. Класс должен принимать смещения, если мы их передаем, или подставлять ноль, если они отсутствуют. Внесите следующие изменения в метод `__init__` класса `RobotImu` (выделены жирным шрифтом):

```
def __init__(self, gyro_offsets=None):
    self._imu = ICM20948()
    self.gyro_offsets = gyro_offsets or vector(0, 0,
0)
```

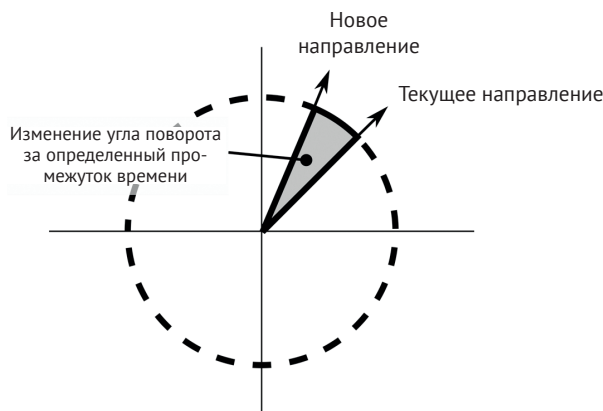
5. Также внесем изменения в метод `read_gyroscope`, чтобы он тоже мог учитывать смещения:

```
def read_gyroscope(self):
    _, _, _, gyro_x, gyro_y, gyro_z = self._imu.read_
accelerometer_gyro_data()
    return vector(x, y, z) - self.gyro_offsets
```

Теперь мы проверим, работает ли наш код. Запустим его и посмотрим, как меняется положение виртуальной модели робота.

## Изменение положения виртуального робота на основе данных гироскопа

Ранее мы говорили, что будем суммировать измерения гироскопа. На рис. 16.5 показано суммирование измерений для одной оси.



**Рис. 16.5.** Суммирование измерений для одной оси гироскопа

На рис. 16.5 пунктирной линией показана окружность поворота по одной оси. Перекрестием отмечен центр окружности. Толстая стрелка, направленная вверх и влево, показывает текущее направление. Затемненный сегмент показывает изменение угла поворота в градусах за определенный промежуток времени. Это изменение мы добавляем к текущему направлению и получаем новое направление, отмеченное на рисунке еще одной толстой стрелкой.

Чтобы получить перемещение, мы умножаем угловую скорость вращения на время. Мы оцениваем (предварительно рассчитываем) перемещение, поскольку у нас нет промежуточных значений.

При работе с ПИД-регуляторами в главе 14 вы узнали, что принцип определения времени относительно последнего измерения очень важен. Часто его называют вычислением дельты (разности) времени.

Мы можем заставить виртуальную модель поворачиваться, основываясь на показаниях гироскопа. Для этого выполните следующие шаги.

1. Создайте файл с именем `visual_gyroscope.py`. Чтобы объединить компоненты, понадобится много импортов, как показано в следующем фрагменте кода:

```
import vpython as vp
from robot_imu import RobotImu
import time
import imu_settings
import virtual_robot
```

2. В этот раз для класса `RobotImu` мы будем использовать созданные ранее настройки:

```
imu = RobotImu(gyro_offsets=imu_settings.gyro_offsets)
```

3. Выполните суммирование по трем осям: `pitch`, `roll` и `yaw`. Для начала установите значения равными нулю:

```
pitch = 0
roll = 0
yaw = 0
```

4. Теперь настройте виртуального робота и его представление:

```
model = virtual_robot.make_robot()
virtual_robot.robot_view()
```

5. Мы будем отслеживать дельту времени, поэтому начнем с последнего значения времени:

```
latest = time.time()
```

6. Затем запустите основной цикл поведенческого скрипта. Поскольку скрипт выполняет визуализацию в VPython, необходимо установить скорость цикла и указать, чтобы он обновлялся:

```
while True:
    vp.rate(1000)
```

7. Далее вычисляем дельту времени (dt), сохраняя новое последнее значение времени:

```
current = time.time()
dt = current - latest
latest = current
```

8. Код считывает данные гироскопа и помещает вектор gyro:

```
gyro = imu.read_gyroscope()
```

9. Как показано в следующем фрагменте кода, суммируем текущую угловую скорость (в градусах в секунду), умноженную на dt (в секундах):

```
roll += gyro.x * dt
pitch += gyro.y * dt
yaw += gyro.z * dt
```

10. Возвращаем виртуальную модель в исходное положение, чтобы подготовить ее к повороту:

```
model.up = vp.vector(0, 1, 0)
model.axis = vp.vector(1, 0, 0)
```

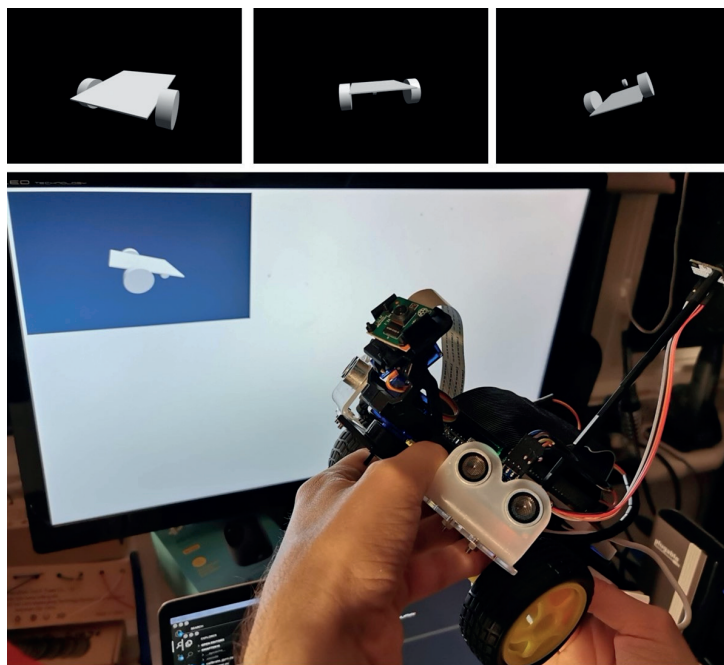
11. Выполняем повороты вокруг каждой оси. Значения нужно преобразовать в радианы:

```
model.rotate(angle=vp.radians(roll), axis=vp.
vector(1, 0, 0))
model.rotate(angle=vp.radians(pitch), axis=vp.
vector(0, 1, 0))
model.rotate(angle=vp.radians(yaw), axis=vp.vector(0, 0, 1))
```

Все повороты выполняются вокруг осей. Поворот вокруг оси x мы задаем с помощью вектора (1, 0, 0).

12. Теперь код готов к запуску. Виртуальная модель будет менять свое положение вместе с реальным роботом! Загрузите код и запустите его с помощью команды `python visual_gyroscope.py`.
13. Как вы уже делали ранее, подождите минуту или около того, а затем введите в браузере `myrobot.local:9020`. На экране вы должны увидеть нечто похожее на то, что показано на рис. 16.6.





**Рис. 16.6.** Поворот реального робота и виртуальной модели

На рис. 16.6 показано, как виртуальный робот поворачивается вместе с реальным. Подвигайте своего робота и постарайтесь привести его примерно в такое же положение, как показано на рисунке.

14. Если вы уже повернули робота, то привести модель в исходное положение не получится ввиду накапливающихся ошибок и дрейфа, вызванного суммированием данных гироскопа.

В этом эксперименте вы могли заметить, что одного лишь гироскопа недостаточно для точного отслеживания параметров вращения. Для этого нам понадобятся другие датчики IMU.

Прежде чем продолжить, устраним возможные неполадки.

### Устранение неполадок

Если что-то пошло не так, воспользуйтесь следующими советами.

1. Для отображения вывода кода в браузере используйте команду `vrpython`.
2. Если реальный робот неподвижен, а виртуальный продолжает поворачиваться, вернитесь к калибровке смещения. Значения гироскопа всегда не идеально точные, но в следующих разделах мы постараемся это исправить.
3. Если модель неконтролируемо поворачивается или «прыгает», проверьте, преобразовали ли вы значения в радианы.
4. Если робот поворачивается не в том направлении (влево/вправо вместо вверх/вниз), проверьте параметры осей вращения.

Теперь, когда все работает, мы можем перейти к акселерометру, который покажет, какие силы действуют на нашего робота.

## ИЗМЕРЕНИЕ УГЛОВ ТАНГАЖА И КРЕНА С ПОМОЩЬЮ АКСЕЛЕРОМЕТРА

В главе 12 вы научились отображать данные акселерометра в виде вектора, а теперь необходимо измерить углы, чтобы использовать его вместе с гироскопом и магнитометром. Чтобы использовать данные акселерометра для вращения, необходимо преобразовать вектор в углы тангажа и крена.

### Определение углов тангажа и крена на основе векторных данных акселерометра

Данные акселерометра представляют собой **декартовы координаты**. Нам нужно преобразовать их в пару углов тангажа и крена, перпендикулярных друг другу. В главе 12 в разделе «Системы координат и вращения» говорится, что крен – это вращение вокруг оси  $x$ , а тангаж – вокруг оси  $y$ .

Мы можем представить это в виде двух плоскостей. При вращении вокруг оси  $x$  мы возьмем вектор в плоскости  $yz$  и найдем его угол. При вращении вокруг оси  $y$  мы сделаем то же самое в плоскости  $xz$ . Взгляните на рис. 16.7.

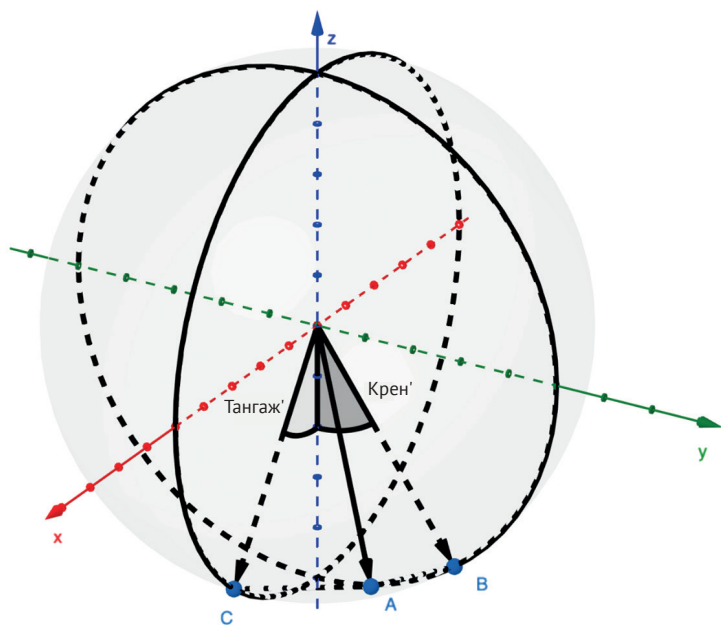


Рис. 16.7. Векторы и углы акселерометра

На рис. 16.7 показаны оси  $x$ ,  $y$  и  $z$ , а также сфера, которую формируют плоскости  $xz$  и  $yz$ .

Вектор **A** – это вектор акселерометра. Используя только компоненты  $xz$ , мы проецируем этот вектор на окружность  $xz$  в точку **C**. Таким образом, угол тангажа – это угол от оси  $z$  до точки **C**. Затем проецируем вектор **A** на окружность  $yz$  в точку **B**. Так, угол крена – это угол от оси  $z$  до точки **B**.

Когда на плоскости есть два компонента (например,  $x$  и  $z$ ), мы можем вычислить угол с помощью функции `atan2` (доступна в большинстве языков программирования). В связи с ориентацией компонентов IMU для угла тангажа нужно выполнить операцию смены знака. Процесс показан на рис. 16.8.

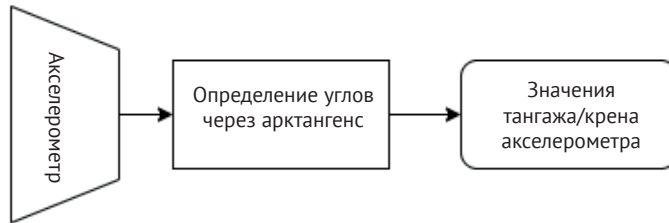


Рис. 16.8. Поток данных акселерометра

На рис. 16.8 показано, как через арктангенс необработанных данных акселерометра мы определяем значения углов тангажа и крена.

Превратим показания акселерометра в углы тангажа и крена, а затем нанесем их на график.

1. Откройте файл `robot_imu.py`.
2. Расширьте импорт, включив в него тригонометрические функции:

```
from vpython import vector, degrees, atan2
```

3. Для выполнения необходимых математических операций после метода `read_accelerometer` добавьте следующий фрагмент кода:

```
def read_accelerometer_pitch_and_roll(self):
    accel = self.read_accelerometer()
    pitch = degrees(-atan2(accel.x, accel.z))
    roll = degrees(atan2(accel.y, accel.z))
    return pitch, roll
```

4. Мы отобразим углы на графике, который также покажет нам, в чем недостаток использования акселерометра без других датчиков. Создайте файл `plot_pitch_and_roll.py`.
5. Начнем с импортов:

```
import vpython as vp
import time
from robot_imu import RobotImu
imu = RobotImu()
```

6. Создаем графики:

```
vp.graph(xmin=0, xmax=60, scroll=True)
graph_pitch = vp.gcurve(color=vp.color.red)
graph_roll = vp.gcurve(color=vp.color.green)
```

7. Чтобы график опирался на время, задаем время начала:

```
start = time.time()
while True:
    vp.rate(100)
    elapsed = time.time() - start
```

8. Теперь мы можем получить новые значения углов тангажа и крена:

```
pitch, roll = imu.read_accelerometer_pitch_and_roll()
```

9. Затем строим их графики:

```
graph_pitch.plot(elapsed, pitch)
graph_roll.plot(elapsed, roll)
```

10. Загрузите файлы `robot_imu.py` и `plot_pitch_and_roll.py`. Запустите их с помощью команды `python plot_accel_pitch_and_roll.py` и подключитесь через браузер к порту 9020. Вы увидите результат, показанный на рис. 16.9.

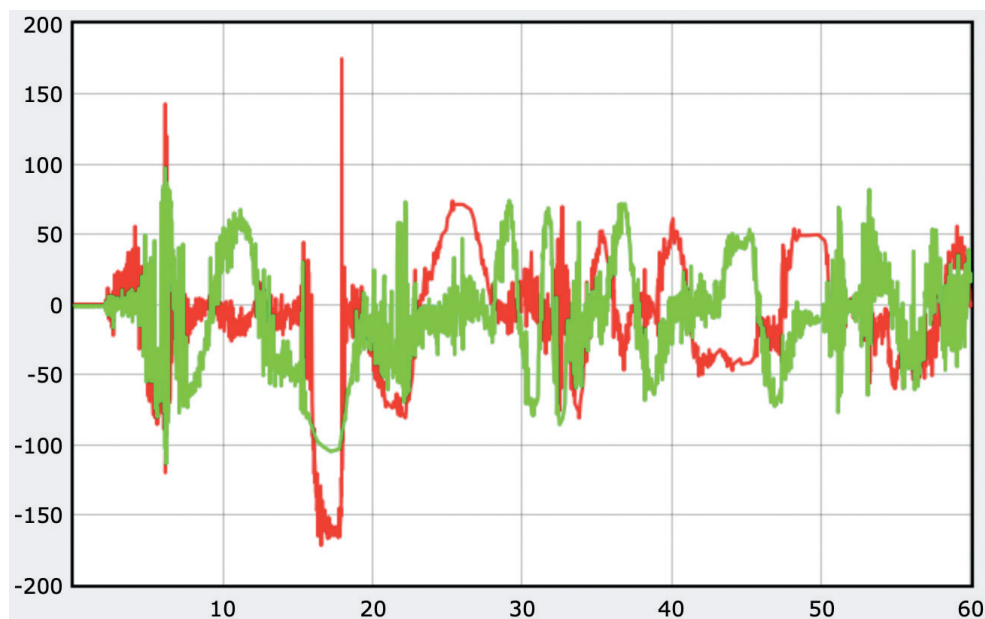


Рис. 16.9. График углов тангажа и крена на основе данных акселерометра

На рис. 16.9 показан скриншот графика. Красная кривая показывает угол тангажа вокруг оси  $y$ , а зеленая – угол крена вокруг оси  $x$ . Мы явно видим колебания от  $+90^\circ$  до  $-90^\circ$ , но, поскольку график сильно зашумлен, на нем не видны результаты движений, которые занимают меньше секунды.

Нам необходимо убрать шум. Сделать это можно с помощью дополнительного фильтра, который будет убирать вибрационный шум, объединяя новое значение с предыдущим. Позже мы создадим такой фильтр, но это замедлит отбор данных.

Проверим, работает ли наш код.

## Устранение неполадок

Если что-то работает не так, попробуйте это исправить.

1. Если график сильно зашумлен, попробуйте поворачивать робота более резко. При этом старайтесь сохранять устойчивое положение рук. Шум на графике обусловлен использованием акселерометра без других датчиков.
2. Если на графике появляются разрывы или он дает сбой за пределами диапазона от 0 до 90°, проверьте, используете ли вы функцию `atan2`. Она математически выполняет тригонометрическое правило КОСТ<sup>17</sup> (CAST).
3. Обратите внимание, что в методе `read_accelerometer_pitch_and_roll` перед функцией `atan2` стоит отрицательный знак.
4. При выходе за пределы 180° на графике могут появляться сбои – это ожидаемый недостаток данной системы. Постарайтесь пока не выходить за этот предел.

Теперь мы приблизительно определили углы тангажа и крена. Получить более точные значения можно посредством объединения данных датчиков и использования фильтра. У нас есть еще один датчик, который дает интегрированное значение углов тангажа и крена – гироскоп.

## Сглаживание данных акселерометра

Воспользуемся уже имеющимися знаниями и сгладим данные акселерометра при помощи сложения с данными гироскопа.

Мы будем использовать дельту времени (класс, который позже значительно сэкономит нам время).

### Дельта времени

Вы уже знаете, как определять прошедшее время для построения графика и дельту времени между обновлениями для интеграции значений. Создадим код.

1. Создайте файл `delta_timer.py` и импортируйте библиотеку `time`:

```
import time
```

2. Для отслеживания создайте класс `DeltaTimer`:

```
class DeltaTimer:
    def __init__(self):
        self.last = self.start = time.time()
```

Код инициализирует переменные `last` и `start` с текущим временем.

3. Затем используем метод `update`. Его вызывает каждый цикл. Начнем с определения текущего времени:

```
def update(self):
    current = time.time()
```

<sup>17</sup> Правило КОСТ (Косинус-Общая-Синус-Тангенс) указывает, какая функция в какой четверти координат имеет положительное значение. Отсчет идет от правой нижней четверти против часовой стрелки. – *Прим. ред.*

4. Дельта времени представляет собой разницу между текущим временем и последним зафиксированным временем, как показано в следующем фрагменте кода:

```
dt = current - self.last
```

5. Прошедшее время – это разница между текущим временем и временем начала:

```
elapsed = current - self.start
```

6. Для дельты времени нам нужно обновить последнее зафиксированное время и вернуть части, как показано в следующем фрагменте кода:

```
self.last = current
return dt, elapsed
```

Теперь мы можем использовать этот класс всякий раз, когда для построения графика нам понадобится дельта времени и прошедшее время. Используем его для объединения данных акселерометра и гироскопа.

## Совместная обработка данных акселерометра и гироскопа

При совместной работе датчики могут компенсировать недостатки друг друга. Акселерометр дает абсолютные значения угла тангажа и крена, что компенсирует дрейф данных гироскопа. Гироскоп не производит столько шума, как акселерометр, и может выполнять быстрые измерения. На рис. 16.10 показано, как объединить датчики.

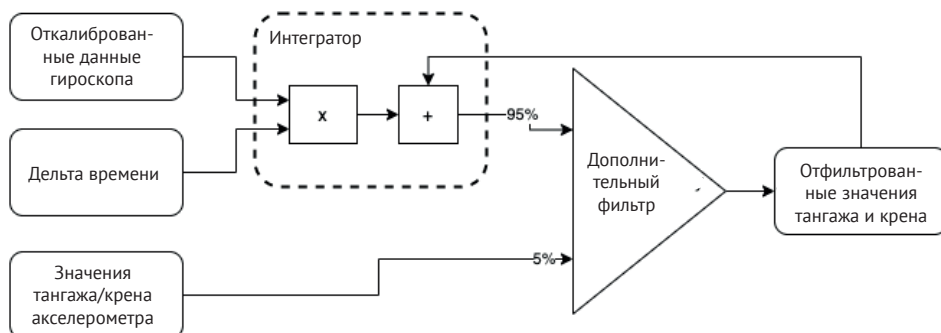


Рис. 16.10. Поток данных при объединении гироскопа и акселерометра

На рис. 16.10 показана блок-схема потока данных с использованием дополнительного фильтра для совместной обработки данных гироскопа и акселерометра. Для примера рассмотрим угол тангажа. Сначала система передает данные гироскопа и дельту времени в сумматор. Сумматор добавляет эти данные к предыдущему положению. Затем 95 % этого фильтра используется для учета значительных изменений движения, а оставшиеся 5 % приходятся на измерения акселерометра. Эти 5 % будут подтягивать измерение к среднему показанию акселерометра, отфильтровывая элемент хаотического шума. В качестве вывода мы получаем отфильтрованные значения углов тангажа и крена, которые отправляются в интегратор следующего цикла.

Создадим код для запуска этого фильтра.

1. Откройте файл `robot_imu.py`.
2. Добавьте класс `ComplementaryFilter`:

```
class ComplementaryFilter:
```

3. Мы можем построить этот фильтр со значением левой части, сохранить его и вычислить дополнение, чтобы получить правую часть (из единицы вычесть значение левой части):

```
def __init__(self, filter_left=0.9):  
    self.filter_left = filter_left  
    self.filter_right = 1.0 - filter_left
```

4. Этот класс имеет метод `filter`, который принимает две части и объединяет их, используя значения фильтра:

```
def filter(self, left, right):  
    return self.filter_left * left + \  
        self.filter_right * right
```

Данный фрагмент завершает фильтр.

5. Чтобы реализовать совместную обработку данных, нам нужен код, объединяющий датчики IMU с помощью фильтра. В файле `robot_imu.py` создадим нужный класс:

```
class ImuFusion:
```

6. В конструкторе сохраняем экземпляр `RobotImu`, создаем фильтр и задаем значения углов тангажа и крена:

```
def __init__(self, imu, filter_value=0.95):  
    self.imu = imu  
    self.filter = ComplementaryFilter(filter_value).  
filter  
    self.pitch = 0  
    self.roll = 0
```

7. Главная часть этого кода – функция `update`, которая принимает параметр `dt` (дельта времени). Она ничего не возвращает, только обновляет элементы тангажа/крена:

```
def update(self, dt):
```

8. Сначала нам нужно получить значения углов тангажа и крена от акселерометра:

```
    accel_pitch, accel_roll = self.imu.read_  
    accelerometer_pitch_and_roll()
```

9. Также нам нужны значения гироскопа. Для этого используем следующую команду:

```
    gyro = self.imu.read_gyroscope()
```

10. Чтобы получить значение угла тангажа, мы объединяем показания гироскопа по оси `y` и значение угла тангажа от акселерометра:

```
    self.pitch = self.filter(self.pitch + gyro.y *  
    dt, accel_pitch)
```



Обратите внимание, что здесь мы выполняем операции умножения и сложения, как показано на рис. 16.10.

11. Делаем то же самое, чтобы получить значение угла крена:

```
self.roll = self.filter(self.roll + gyro.x * dt,
    accel_roll)
```

Итак, мы подготовили класс `RobotImu` с фильтрами и объединили датчики. Давайте проверим этот код с помощью графика.

1. В файл `plot_pitch_and_roll.py` добавим импорты `DeltaTimer`, `ImuFusion` и импорт калибровки гироскопа. Обратите внимание, что в следующем фрагменте кода строка `import time` отсутствует:

```
import vpython as vp
from robot_imu import RobotImu, ImuFusion
from delta_timer import DeltaTimer
import imu_settings
```

2. Затем настраиваем экземпляр класса `RobotImu` (с настройками гироскопа) и создаем экземпляр `fusion`:

```
imu = RobotImu(gyro_offsets=imu_settings.gyro_offsets)
fusion = ImuFusion(imu)
```

3. Далее нам требуется `dt` (дельта времени) для вычислений, связанных с объединенными данными, и прошедшее время для построения графика. Все это есть в классе `DeltaTimer`. Помещаем его перед запуском цикла вместо оператора присваивания `start`:

```
timer = DeltaTimer()
```

4. Теперь мы используем таймер в цикле, который вычисляет значение `elapsed`:

```
while True:
    vp.rate(100)
    dt, elapsed = timer.update()
```

5. Чтобы обновить слияние данных с дельтой времени, замените показания акселерометра на следующий фрагмент кода, чтобы он выполнил свои вычисления:

```
fusion.update(dt)
```

6. Мы можем вызвать значения углов тангажа и крена из объекта `fusion`:

```
graph_pitch.plot(elapsed, fusion.pitch)
graph_roll.plot(elapsed, fusion.roll)
```

7. Загрузите в память робота файлы `robot_imu.py`, `delta_timer.py` и `plot_pitch_and_roll.py`.
8. Используйте команду `vpython plot_pitch_and_roll.py`; в браузере подключитесь к порту 9020.

Сначала результат будет похож на график тангажа и крена акселерометра, показанный на рис. 16.9. Однако когда вы начнете перемещать робота, то заметите, что шума стало гораздо меньше и график стал более плавный. Если вы

поставите робота на поверхность или будете держать его неподвижно – график выравнивается. Этот график может учитывать резкие вращения робота. Система работает плавно и точно!

### Устранение неполадок

При возникновении проблем попробуйте следующие способы устранения неполадок.

1. Если возникает синтаксическая ошибка или система начинает вести себя странно, проверьте, нет ли в коде опечаток.
2. Если система выдает странный результат, проверьте значение фильтра. Оно должно быть 0.95 (а не 95).
3. Проверьте, все ли файлы вы загрузили в память робота.
4. Чтобы график окончательно установился, системе потребуется 1-2 с.

Теперь вы знаете, как с помощью датчиков можно получить точные и плавные значения углов тангажа и крена. Роботу, передвигающемуся с помощью колес, эти значения могут не пригодиться по прямому назначению, однако одно из них необходимо для работы с компасом. Перейдем к магнитометру.

## ОПРЕДЕЛЕНИЕ НАПРАВЛЕНИЯ С ПОМОЩЬЮ МАГНИТОМЕТРА

В главе 12 вы узнали, как строится вектор магнитометра и как на него влияют магнитные металлы (например, небольшое количество стали или железа). Влияют на показания даже штыревые разъемы – эта проблема решается калибровкой.

В нашем случае определение компонентов  $X$ ,  $Y$  и  $Z$  не так уж полезно. Нам нужно, чтобы робот ориентировался на магнитный север. Позже вы узнаете, как использовать магнитометр для выполнения точных поворотов.

В этом разделе вам понадобится пространство без магнитных объектов поблизости. На показания магнитометра могут влиять ноутбуки, телефоны, микрофоны и накопители на магнитных дисках. Определить наличие магнитных полей вы можете с помощью обычного компаса. Я рекомендую сделать *стойку для IMU* настолько длинной, насколько позволяет провод (для этого вы можете соединить больше стоек), поскольку двигатели робота имеют собственное сильное магнитное поле.

Перед включением поставьте робота так, чтобы он был обращен примерно на север. Если робот будет обращен на юг, он может начать вести себя странно. Позже мы рассмотрим эту проблему детально и решим ее.

### Калибровка магнитометра

Мы выполним калибровку на основе **расчета смещения от твердого железа**. К твердому железу относятся любые магнитные объекты рядом с магнитометром, которые движутся вместе с ним. Чтобы определить интенсивность поля на каждой оси, мы будем перемещать робота по ним. Для компенсации смещения будем использовать средние показания по осям. Затем мы добавим это в настройки IMU. Весь процесс будет похож на калибровку гироскопа, но здесь мы будем перемещать робота в пространстве.

Перейдем к созданию кода.

1. Создайте файл с именем `magnetometer_calibration.py`. Добавьте импорт и настройку `RobotImu`:

```
import vpython as vp
from robot_imu import RobotImu
imu = RobotImu()
```

2. Найдите минимальный и максимальный векторы, как мы уже делали для гироскопа:

```
mag_min = vp.vector(0, 0, 0)
mag_max = vp.vector(0, 0, 0)
```

3. Мы визуализируем систему в виде набора из трех точечных диаграмм с кластерами из цветных точек. Каждый из трех кластеров будет представлять собой график, объединяющий две оси:  $xy$ ,  $yz$  и  $xz$ . Нам нужно выровнять оси путем калибровки устройства:

```
scatter_xy = vp.gdots(color=vp.color.red)
scatter_yz = vp.gdots(color=vp.color.green)
scatter_zx = vp.gdots(color=vp.color.blue)
```

4. Начинаем с главного цикла и считываем показания магнитометра:

```
while True:
    vp.rate(100)
    mag = imu.read_magnetometer()
```

5. Обновляем минимальные значения, как делали это для гироскопа:

```
mag_min.x = min(mag_min.x, mag.x)
mag_min.y = min(mag_min.y, mag.y)
mag_min.z = min(mag_min.z, mag.z)
```

6. Затем обновляем максимальные значения:

```
mag_max.x = max(mag_max.x, mag.x)
mag_max.y = max(mag_max.y, mag.y)
mag_max.z = max(mag_max.z, mag.z)
```

7. Вычисляем смещение (как для гироскопа):

```
offset = (mag_max + mag_min) / 2
```

8. Оператор `print` в следующем фрагменте кода показывает текущие значения и смещения:

```
print(f"Magnetometer: {mag}. Offsets: {offset}.")
```

9. Теперь строим точечные диаграммы. Они дадут достаточно калибровочных данных и покажут, в каких местах оси не совпадают. Добавьте следующий фрагмент кода:

```
scatter_xy.plot(mag.x, mag.y)
scatter_yz.plot(mag.y, mag.z)
scatter_zx.plot(mag.z, mag.x)
```

10. Загрузите код и запустите его с помощью VPython. Вы увидите результат, как на рис. 16.11.

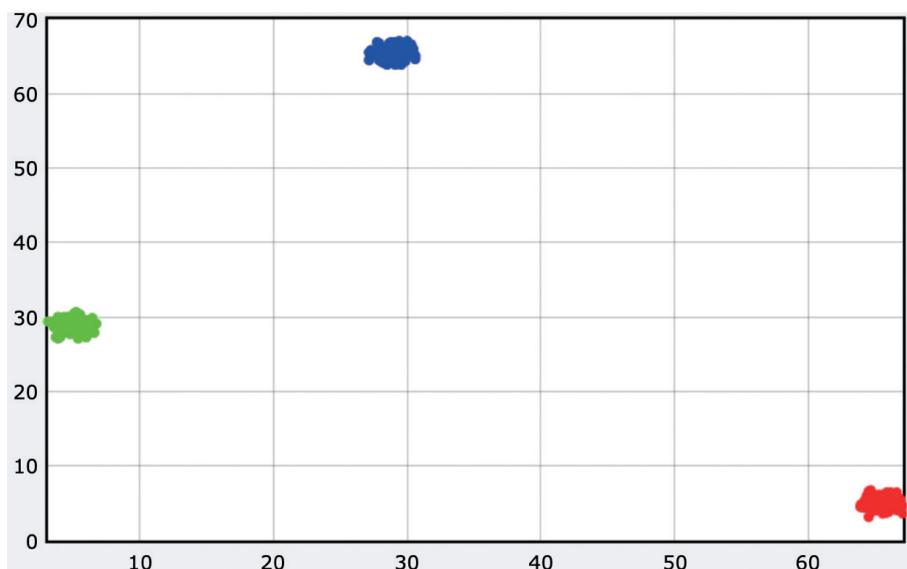


Рис. 16.11. Экран калибровки магнитометра на начальном этапе

На рис. 16.11 кластеры показаны в виде трех разноцветных пятен: справа снизу находится красный кластер (для  $x_y$ ), сверху – синий (для  $u_z$ ) и слева – зеленый (для  $z_x$ ). В зависимости от текущего положения вашего робота в пространстве кластеры могут быть расположены по-разному.

11. Начните медленно перемещать робота, вращая его вокруг оси  $y$  (вокруг оси колес). Зеленый кластер должен принять форму эллипса, как на рис. 16.12.

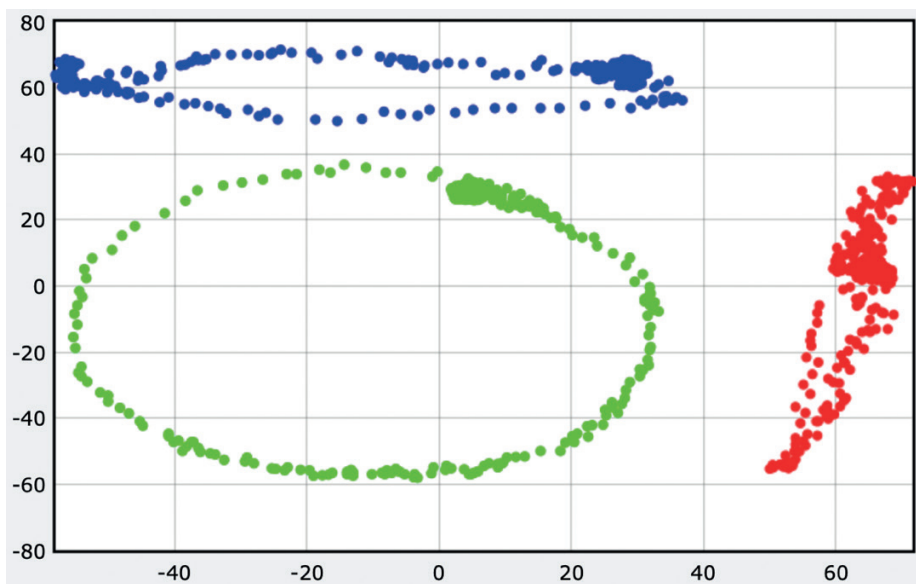


Рис. 16.12. Данные частично откалиброванного магнитометра

На рис. 16.12 мы видим зеленый кластер в форме эллипса и два других кластера – красный и синий. Чем медленнее вы перемещаете робота, тем точнее будет результат.

12. Поверните робота вокруг оси x (длина робота), а затем вокруг оси z (высота). Чем больше углов вы охватите, тем лучше. Чтобы заполнить пробелы, несколько раз переместите робота по «восьмерке». Результат должен выглядеть так, как показано на рис. 16.13.

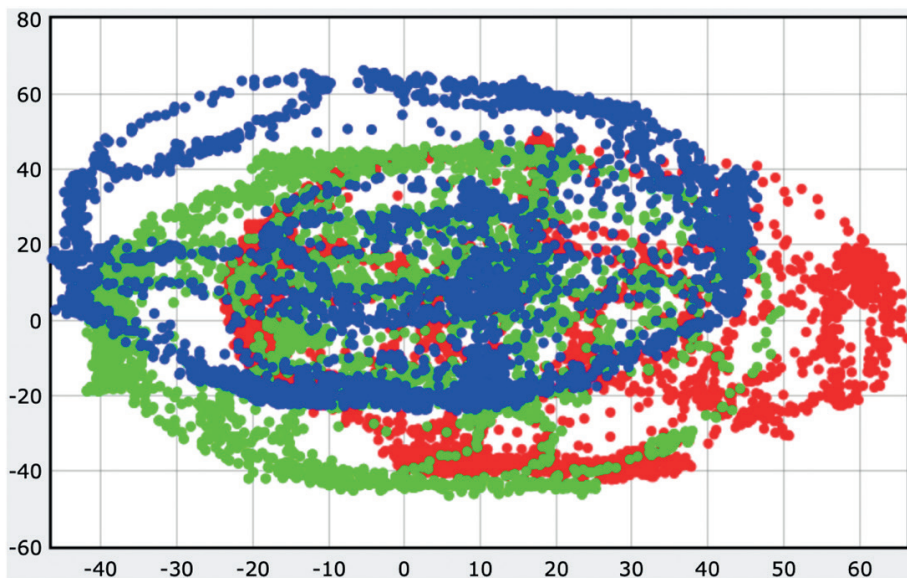


Рис. 16.13. Калибровка магнитометра: правильный набор данных

На рис. 16.13 показан правильный набор данных с тремя кластерами (красный, зеленый и синий) в виде эллипсов. Обратите внимание, что здесь есть значения, выходящие за пределы кластеров. Такие точки возникают, когда робот находится рядом с магнитными объектами, – остерегайтесь их!

13. Загрузите данные калибровки и закройте браузер.
14. В консоли вы увидите смещения калибровки:

```
Magnetometer: <30.6, -36.9, 10.35>. Offsets: <21.15,
3.15, 0.225>.
Magnetometer: <31.5, -36.75, 11.25>. Offsets: <21.15,
3.15, 0.225>.
```

Сначала значения смещений сильно меняются, но по мере сбора большего количества данных они стабилизируются, даже если показания магнитометра продолжают меняться.

Теперь у нас есть числа калибровки. У меня получились такие: 21.15, 3.15, 0.225. Удостоверимся, что получили правильные значения.

## Устранение неполадок

Калибровка могла не сработать. Рассмотрим несколько случаев.

1. Если значения не стабилизируются, продолжайте перемещать робота. Чтобы получить полный диапазон, вам нужно вращать робота на 360° по всем осям.
2. Если за пределами кластеров возникают странные точки, переместитесь в другую комнату и начните калибровку заново. Скорее всего, в помещении, где вы находитесь, есть сильное магнитное поле, которое искажает результаты.
3. Работа вашего браузера может замедлиться или ему может не хватить памяти – хоть я и советую перемещать робота медленно, это не всегда помогает, потому что он все равно продолжает добавлять точки.
4. Если вы вообще не видите кластеров (линий или небольших пятен), проверьте правильность комбинаций осей *xu*, *yz* и *zx*.

Теперь вы получили значения смещений калибровки. Используем их, чтобы выровнять точечные диаграммы.

## Проверка калибровочных значений

Проверим, эффективны ли полученные значения, и поместим их обратно в код. В результате кластеры должны выровняться. Сначала нам нужно настроить смещения в интерфейсе RobotImu. Выполните следующие шаги.

1. Откройте файл `robot_imu.py`.
2. Мы будем хранить смещения в методе `__init__`. В следующем фрагменте кода изменения выделены жирным шрифтом:

```
def __init__(self, gyro_offsets=None,
    mag_offsets=None):
    self._imu = ICM20948()
    self.gyro_offsets = gyro_offsets or vector(0, 0, 0)
    self.mag_offsets = mag_offsets or vector(0, 0, 0)
```

3. Метод `read_magnetometer` должен вычесть смещения для магнитометра:

```
def read_magnetometer(self):
    mag_x, mag_y, mag_z = self._imu.read_
magnetometer_data()
    return vector(mag_x, -mag_z, -mag_y) - self.mag_offsets
```

Теперь наши скрипты включают в себя смещение для магнитометра. Мы поместим их в тот же файл настроек, который мы использовали для калибровки гироскопа. Выполните следующие шаги.

1. Откройте файл `imu_settings.py`.
2. Добавьте показания калибровки магнитометра:

```
from vpython import vector
gyro_offsets = vector(0.557252, -0.354962, -0.522901)
mag_offsets = vector(21.15, 3.15, 0.225)
```

3. Теперь их можно использовать для построения точечной диаграммы. Откройте файл `magnetometer_calling.py` и выполните импорт настроек, как показано в следующем фрагменте кода:

```
import vpython as vp
from robot_imu import RobotImu
from imu_settings import mag_offsets
```

- После того как мы создали экземпляр `RobotImu`, можно применить смещение:

```
imu = RobotImu(mag_offsets=mag_offsets)
```

- Загрузите файлы в память робота и повторно запустите файл `magnetometer_caulibration.py`. Чтобы охватить множество точек в разных ориентациях, начните вращать робота и перемещать его по «восьмерке». После сбора данных вы увидите кластеры, которые накладываются друг на друга, как на рис. 16.14.

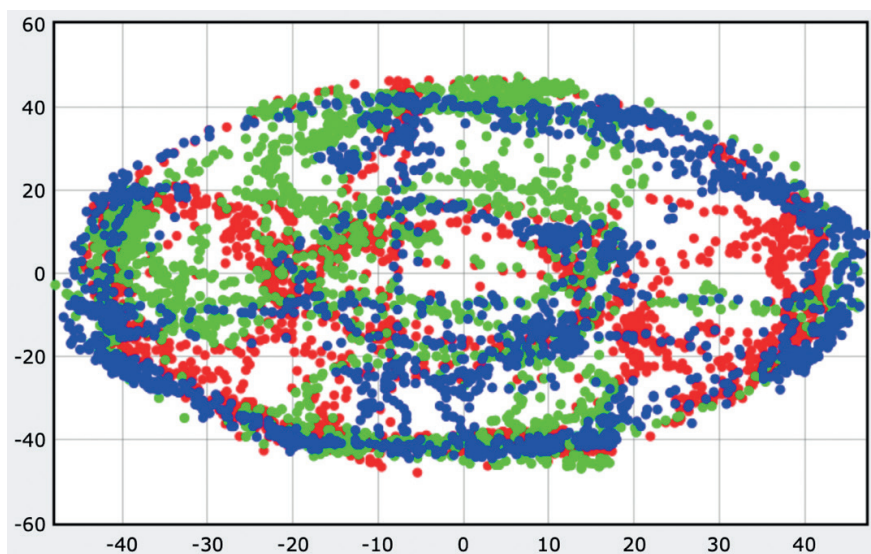


Рис. 16.14. Данные откалиброванного магнитометра

На рис. 16.14 показаны наложенные друг на друга кластеры. Поздравляю, вы откалибровали магнитометр!

В дальнейших экспериментах откалиброванный магнитометр будет давать более точные показания. Но сначала нужно устранить возможные неполадки.

### Что делать, если кластеры не накладываются друг на друга

На этом этапе вы можете столкнуться с неполадками, при которых кластеры не накладываются друг на друга. Чтобы устранить их, выполните следующие шаги.

- Запустите код калибровки повторно. Прежде чем сделать это, закомментируйте строку, которая применяет (настраивает) смещения в классе `RobotImu`. Запуск кода калибровки при активных смещениях приведет к некорректным результатам.
- Внимательно проверьте код калибровки и код IMU.



3. Рядом с роботом не должно быть сильных магнитов или больших металлических объектов (например, микрофонов или жестких дисков). Они должны находиться как минимум в метре от робота. Влиять на показания могут даже ваш ноутбук или мобильный телефон.
4. Перемещайте робота вокруг оси достаточно медленно и обязательно попробуйте двигать его по «восьмерке». Продолжайте до тех пор, пока не увидите три кластера в виде эллипсов.
5. Вращая и перемещая робота, обращайтесь к выходным данным консоли – там вы увидите, стабилизировались ли выходные значения смещения.
6. Когда выходные значения стабилизируются, попробуйте применить смещение еще раз и запустите калибровку. Проверьте, накладываются ли кластеры друг на друга.

Теперь у вас должны быть правильные значения калибровки, с которыми можно работать дальше. Давайте поместим их обратно в векторный вывод и определим направление робота.

## ПРИБЛИЗИТЕЛЬНОЕ ОПРЕДЕЛЕНИЕ НАПРАВЛЕНИЯ РОБОТА С ПОМОЩЬЮ МАГНИТОМЕТРА НА ПРАКТИКЕ

Теперь, когда у нас есть настройки калибровки, мы можем научить нашего робота определять, где находится север, с помощью магнитометра (как с помощью компаса). Слова *направление* и *рыскание* означают одно и то же – место, куда мы обращены относительно точки начала отсчета, – в данном случае это магнитный север. Давайте посмотрим, как это реализовать. Взгляните на рис. 16.15.

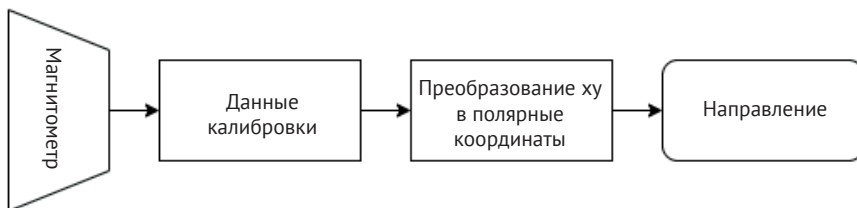


Рис. 16.15. Приблизительное определение направления с помощью магнитометра

На рис. 16.15 показан способ, который мы реализуем. Мы берем данные от калиброванного магнитометра, а затем используем функцию `atan2` (как уже делали при приблизительном определении направления с помощью гироскопа). То же самое можно сделать и для приблизительного определения направления с помощью компаса.

Выполните следующие шаги.

1. Создайте файл `plot_mag_heading.py`. Начните с импорта:

```
import vpython as vp
from robot_imu import RobotImu
from delta_timer import DeltaTimer
import imu_settings
```

2. Инициализируйте экземпляр RobotImu с настройками:
 

```
imu = RobotImu(mag_offsets=imu_settings.mag_offsets)
```
3. Чтобы создать виртуальное представление компаса на экране, нам нужен лимб (в виде цилиндра) и стрелка компаса (в виде стрелки) красного цвета:
 

```
vp.cylinder(radius=1, axis=vp.vector(0, 0, 1),
            pos=vp.vector(0, 0, -1))
needle = vp.arrow(axis=vp.vector(1, 0, 0),
                  color=vp.color.red)
```
4. Теперь создадим график для отображения направления и таймер (DeltaTimer) для прошедшего времени:
 

```
vp.graph(xmin=0, xmax=60, scroll=True)
graph_yaw = vp.gcurve(color=vp.color.blue)
timer = DeltaTimer()
```
5. Запускаем главный цикл (с определенной скоростью) и запрашиваем прошедшее время:
 

```
while True:
    vp.rate(100)
    dt, elapsed = timer.update()
```
6. С помощью следующей команды считываем показания магнитометра:
 

```
mag = imu.read_magnetometer()
```
7. Чтобы определить направление, мы берем плоскость ху и используем функцию atan2:
 

```
yaw = -vp.atan2(mag.y, mag.x)
```
8. Наносим значения в градусах на график:
 

```
graph_yaw.plot(elapsed, vp.degrees(yaw))
```
9. Нам необходимо задать оси стрелки компаса соответствующее направление. Преобразуем ее в единичный вектор с помощью функций sin/cos:
 

```
needle.axis = vp.vector(vp.sin(yaw), vp.cos(yaw), 0)
```
10. Сохраните код, загрузите его в память робота и запустите с помощью VPython. В браузере подключитесь к порту 9020.
11. Когда вы начнете перемещать робота в пространстве, вы увидите результат, показанный на рис. 16.16.

В верхней части рис. 16.16 показан компас с красной стрелкой, направленной вверх, – там для робота находится север. В нижней части показан график синего цвета с диапазоном значений от +180 до –180°. При перемещении робота вы увидите, как график меняется. Здесь 0° – это север. Обратите внимание, что робот должен находиться на плоской поверхности.

Помните, что компас показывает, где находится север относительно робота, а не где находится робот относительно севера!

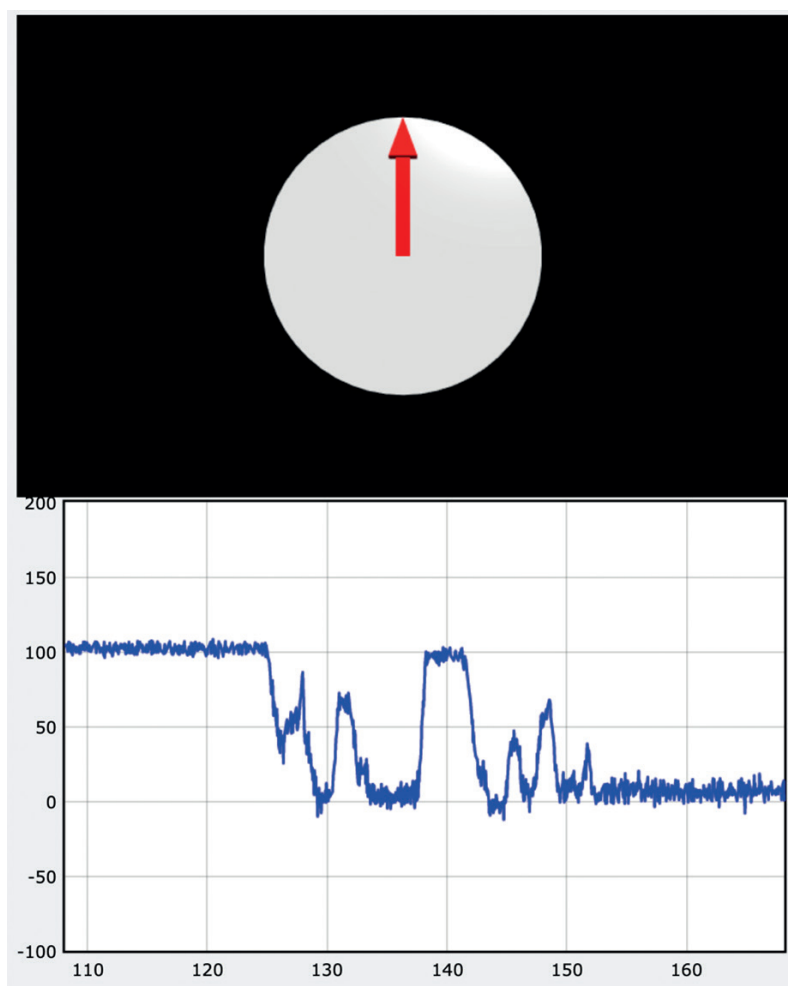


Рис. 16.16. Определение направления с помощью магнитометра

Реализованный алгоритм делает нашего робота еще «разумнее». Он может понимать, где находится север, выполнять некоторые измерения с помощью компаса и определять направление.

Результат не совсем стабильный, и на него легко может повлиять изменение углов тангажа или крена. Мы можем повысить эффективность этой системы посредством объединения данных магнитометра и других датчиков.

## ОБЪЕДИНЕНИЕ ПОКАЗАНИЙ ДАТЧИКОВ ДЛЯ ОРИЕНТИРОВАНИЯ РОБОТА

Вы уже знаете, как объединить показания акселерометра и гироскопа, чтобы сгладить показания углов тангажа и крена. Теперь мы можем объединить данные всех датчиков и сгладить показания магнитометра, чтобы робот мог

ориентироваться в пространстве. Такая система позволит нам приблизительно определять абсолютную ориентацию робота.

Взгляните на рис. 16.17. Здесь показан поток данных, объединяющий все этапы работы с датчиками.

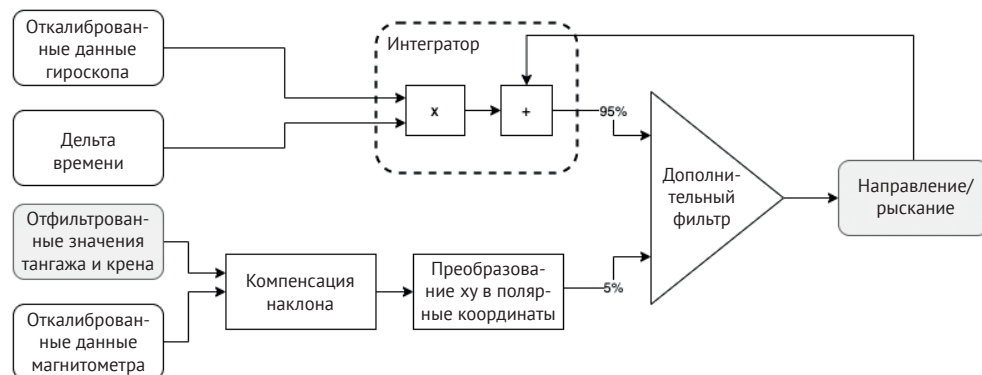


Рис. 16.17. Объединение данных трех датчиков

Слева на рис. 16.17 показаны данные со всех предыдущих этапов работы. Отфильтрованные значения углов тангажа и крена находятся в сером блоке, поскольку являются выходными данными. К входным данным относятся данные откалиброванного гироскопа (рыскание), дельта времени и данные откалиброванного магнитометра. Отфильтрованные значения углов тангажа и крена проходят через этап компенсации наклона, где мы поворачиваем вектор магнитометра. Затем данные магнитометра проходят через этап преобразования координат на плоскости  $xu$  в полярную систему координат, и мы определяем направление с помощью функции `atan2`.

В верхней части рисунка показаны данные откалиброванного гироскопа (рыскание) и дельта времени. Они проходят через этап интеграции, где к ним добавляются предыдущие показания рыскания. Выходные данные интегратора и магнитометра направляются в дополнительный фильтр, при этом доминирующими являются выходные данные интегратора. В результате мы получаем стабильные выходные данные направления/рыскания с высоким быстродействием, которые возвращаются к абсолютному направлению. Теперь у нас есть все три угла – тангаж, крен и рыскание!

Внесем изменения в наш код.

1. Откройте файл `robot_imu.py` и перейдите к классу `ImuFusion`.
2. Нам нужно преобразовать значения обратно в радианы, поэтому импортируем их из `VPython`:

```
from icm20948 import ICM20948
from vpython import vector, degrees, atan2, radians
```

3. В метод `__init__` добавим переменную для хранения значений рыскания:

```
def __init__(self, imu, filter_value=0.95):
    self.imu = imu
    self.filter = ComplementaryFilter(filter_value).
```

```
Filter
    self.pitch = 0
    self.roll = 0
    self.yaw = 0
```

Здесь мы будем использовать тот же фильтр, что и до этого.

4. В метод `update` после вычисления тангажа и крена добавьте следующий фрагмент кода для считывания показаний магнитометра:

```
mag = self.imu.read_magnetometer()
```

5. Переменная `mag` – это вектор. Мы поворачиваем его, используя значения тангажа и наклона, чтобы совместить компоненты `xу`:

```
mag = mag.rotate(radians(self.pitch), vector(0, 1, 0))
mag = mag.rotate(radians(self.roll), vector(1, 0, 0))
```

6. Теперь мы можем вычислить значение рыскания на основе данных магнитометра:

```
mag_yaw = -degrees(atan2(mag.y, mag.x))
```

7. Чтобы стабилизировать показания, используем дополнительный фильтр и данные гироскопа:

```
self.yaw = self.filter(self.yaw + gyro.z * dt, mag_yaw)
```

Теперь значение `self.yaw` будет иметь компенсированное значение рыскания (или направления), что позволит использовать IMU в качестве компаса. Для этого мы реализуем три визуальных представления – график, компас и движения робота. Выполните следующие шаги.

1. Создайте новый файл с именем `visual_fusion.py`. Код покажется вам знакомым. Новыми будут только значения смещения магнитометра и значения рыскания. Добавьте импорты, показанные в следующем фрагменте кода:

```
import vpython as vp
from robot_imu import RobotImu, ImuFusion
from delta_timer import DeltaTimer
import imu_settings
import virtual_robot
```

2. Подготовьте экземпляр `RobotImu` со значениями смещений магнитометра и инициализируйте `fusion`:

```
imu = RobotImu(gyro_offsets=imu_settings.gyro_offsets,
               mag_offsets=imu_settings.mag_offsets)
fusion = ImuFusion(imu)
```

3. Мы будем использовать два холста `VPython` – один для виртуального робота, а другой для компаса. Каждый холст может содержать одну трехмерную сцену. Текущий холст будет видом робота, его мы поместим влево. С ним будет связана модель робота. Добавьте следующий фрагмент кода:

```
robot_view = vp.canvas(align="left")
model = virtual_robot.make_robot()
virtual_robot.robot_view()
```

4. В дополнение создадим холст `compass`, в котором будем использовать уже знакомый нам цилиндр и стрелку. Обратите внимание, что новейший созданный холст связан с фигурами, созданными после него. Добавьте следующий фрагмент кода:

```
compass = vp.canvas(width=400, height=400)
vp.cylinder(radius=1, axis=vp.vector(0, 0, 1),
            pos=vp.vector(0, 0, -1))
needle = vp.arrow(axis=vp.vector(1, 0, 0),
                  color=vp.color.red)
```

5. Настроим графики для тангажа, крена и рыскания:

```
vp.graph(xmin=0, xmax=60, scroll=True)
```

```
graph_roll = vp.gcurve(color=vp.color.red)
graph_pitch = vp.gcurve(color=vp.color.green)
graph_yaw = vp.gcurve(color=vp.color.blue)
```

6. Создадим дельта-таймер, запустим цикл и запросим обновление времени:

```
timer = DeltaTimer()
while True:
    vp.rate(100)
    dt, elapsed = timer.update()
```

7. Теперь мы обновляем `fusion` и добавляем время (для считывания данных IMU и выполнения вычислений):

```
fusion.update(dt)
```

8. Необходимо сбросить настройки модели виртуального робота, прежде чем мы повернем ее:

```
model.up = vp.vector(0, 1, 0)
model.axis = vp.vector(1, 0, 0)
```

9. Теперь последовательно поворачиваем модель вокруг трех осей (крен, тангаж и рыскание):

```
model.rotate(angle=vp.radians(fusion.roll), axis=vp.vector(1, 0, 0))
model.rotate(angle=vp.radians(fusion.pitch), axis=vp.vector(0, 1, 0))
model.rotate(angle=vp.radians(fusion.yaw), axis=vp.vector(0, 0, 1))
```

10. Устанавливаем положение стрелки компаса – обратите внимание, что значение угла рыскания указано в градусах, поэтому мы преобразуем его следующим образом:

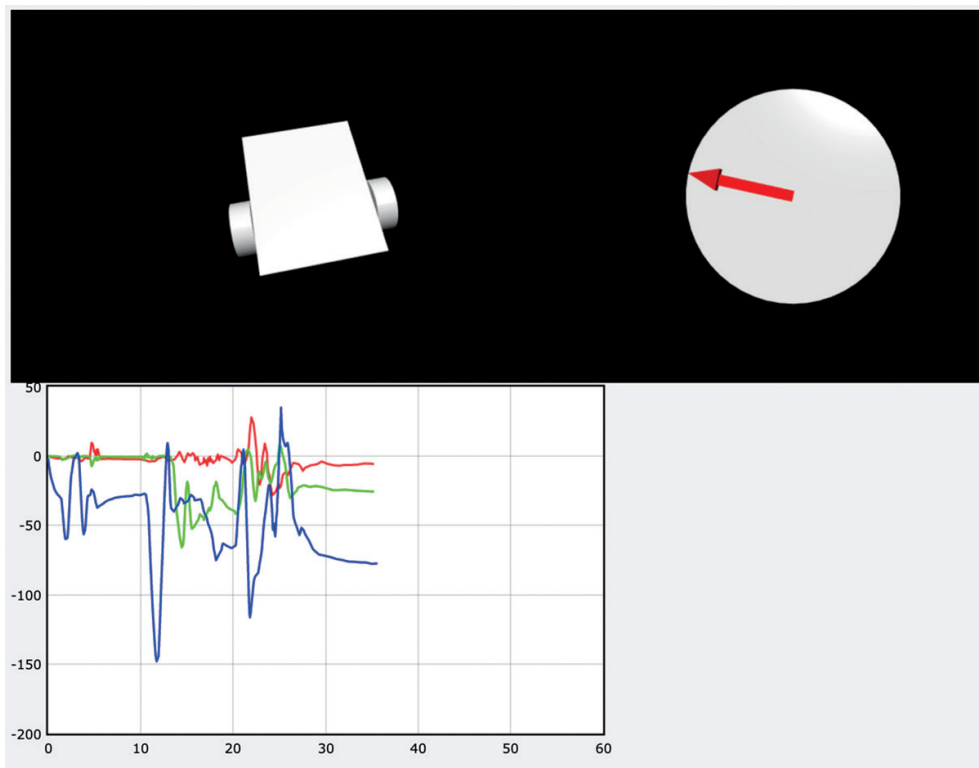
```
needle.axis = vp.vector(
    vp.sin(vp.radians(fusion.yaw)),
    vp.cos(vp.radians(fusion.yaw)),
    0)
```

11. Создаем графики для трех осей:

```
graph_roll.plot(elapsed, fusion.roll)
graph_pitch.plot(elapsed, fusion.pitch)
graph_yaw.plot(elapsed, fusion.yaw)
```

12. Загрузите файлы `robot_imu.py` и `visual_fusion.py` в память робота. Введите команду `python visual_fusion.py`. В браузере подключитесь к порту 9020.

В результате вы увидите виртуальную модель робота, компас и график для трех отображаемых осей. График должен быть относительно стабильным и отзывчивым, как показано на рис. 16.18.



**Рис. 16.18.** График тангажа, крена и рыскания

На рис. 16.18 показан скриншот экрана с результатом. В левой верхней части рисунка находится виртуальная модель робота – вы можете изменить ее вид, щелкнув правой кнопкой мыши. В правой верхней части рисунка показан компас. В нижней части рисунка показан прокручивающийся график тангажа, рыскания и крена. Красная линия – крен, зеленая – тангаж, синяя – рыскание. Прежде чем графики начнут отображать движение вашего робота, им нужно время для стабилизации. Вращение вокруг одной оси всегда незначительно влияет на вращение вокруг других, но все же оно может происходить независимо.

При выходе за пределы диапазона  $\pm 180^\circ$  графики начнут вести себя странно. Посмотрим, как решить эту проблему.



## Решение проблемы 180 градусов

Важно помнить, что углы на окружности расположены циклически, т. е. углы 200 и  $-160^\circ$  равны, также как и углы  $-180$  и  $180^\circ$ . Сейчас это не указано в нашем коде и у нас нет соответствующего фильтра, поэтому, когда мы достигаем точки  $180^\circ$  и функция `atan2` переключается между  $-179,988$  и  $179,991$  (или подобной очень близкой отметкой), линии на графике становятся хаотичными. График принимает разницу менее  $1^\circ$  за  $360^\circ$ , а затем начинает метаться между этими значениями.

Чтоб исправить эту проблему, необходимо внести некоторые изменения в код. Сначала мы объявим, что углы должны быть численно меньше 180 и больше  $-180$ , тем самым ограничив возможный диапазон. Мы будем использовать дополнительный фильтр углов, который определим следующим образом.

1. В начале файла `robot_imu.py` внутри класса `ComplementaryFilter` добавим метод для форматирования угла:

```
@staticmethod
def format_angle(angle):
```

2. Если угол меньше  $-180$ , нам нужно развернуть его, добавляя 360:

```
if angle < -180:
    angle += 360
```

3. Если угол больше 180, мы разворачиваем его, вычитая 360:

```
if angle > 180:
    angle -= 360
return angle
```

4. Для ограничения углов, внесем изменения во внутреннюю часть функции `filter`. При фильтрации мы форматируем входящие углы:

```
def filter(self, left, right):
    left = self.format_angle(left)
    right = self.format_angle(right)
```

5. Также нам нужно поместить отфильтрованные углы в один диапазон. Если разница больше  $350^\circ$ , можно предположить, что это произошло в результате разворачивания; поэтому мы добавляем 360 к самому низкому значению, чтобы фильтровать все вместе:

```
if left - right > 350:
    right += 360
elif right - left > 350:
    left += 360
filtered = self.filter_left * left + \ self.filter_right * right
```

6. Результат этой операции может выходить за пределы диапазона, поэтому мы форматируем его обратно:

```
return format_angle(filtered)
```

7. Этот фильтр готов к использованию, поэтому мы можем повторно открыть файл `visual_fusion.py` и снова попробовать повернуться на  $180^\circ$ . Когда вы подключаетесь к порту через браузер, на экране вы должны ви-

деть, как виртуальный робот повторяет движения реального. Он не должен отклоняться!

Обратите внимание, что при запуске робот не должен быть обращен на юг – с этим система не справится. Но тем не менее мы решили как минимум одну проблему нашей системы и сгладили погрешности.

Этот поведенческий сценарий очень интересный. Теперь вы можете пользоваться моделью виртуального робота и с ее помощью смотреть, как поворачивает реальный робот. Пришло время применить сценарий на практике и задействовать двигатели!

## УПРАВЛЕНИЕ РОБОТОМ НА ОСНОВЕ ПОКАЗАНИЙ IMU

В предыдущих главах вы узнали, как использовать ПИД-алгоритм, а в этой главе научились определять углы тангажа, крена и рыскания с помощью магнитометра. Сейчас наш робот не может выполнять движения тангажа или крена, но может менять свое направление.

В этом разделе мы заставим робота следовать по направлению и отслеживать, где находится север независимо от того, куда мы его повернем. Посмотрим, как это сделать. Взгляните на рис. 16.19.

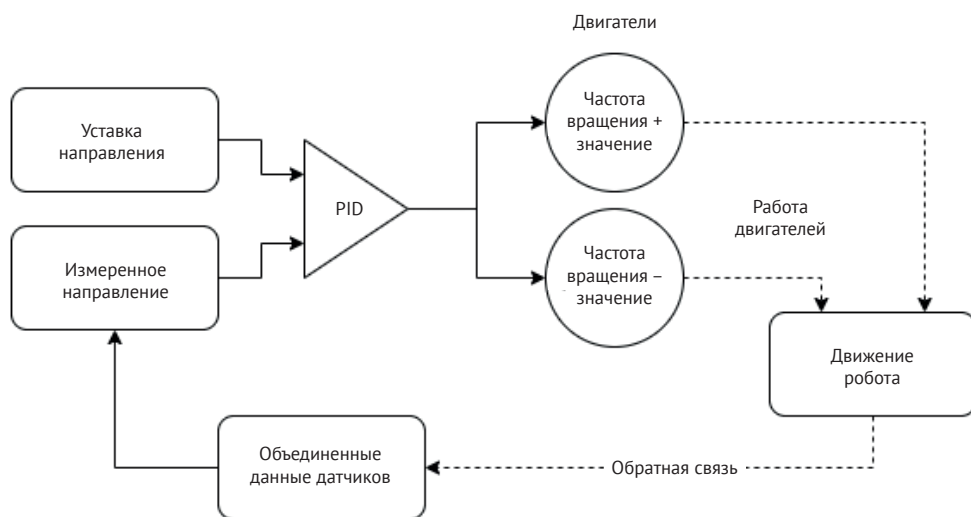


Рис. 16.19. Поведенческий сценарий следования по направлению

На рис. 16.19 показан поток данных. Слева мы видим измеренное направление и уставку для направления, которые переходят в ПИД-регулятор, – значение ошибки будет являться разницей между ними. Измеренное направление мы получаем от алгоритма IMU + **Объединение данных датчиков**. Управление двигателями осуществляется на основе выходных данных ПИД-регулятора. Двигатели работают с фиксированной частотой вращения (плюс-минус некоторое значение), поэтому робот будет поворачивать, чтобы компенсировать ошибку. Данные о движении робота возвращаются в алгоритм IMU + **Объединение данных датчиков**, а затем снова проходят через ПИД-регулятор.

Перейдем к созданию кода на основе рассмотренного потока данных.

1. Создайте файл `drive_north_behavior.py` со следующими импортами:

```
from robot_imu import RobotImu, ImuFusion
from delta_timer import DeltaTimer
from pid_controller import PIController
from robot import Robot
import imu_settings
```

2. Далее инициализируйте `RobotImu`, `fusion` и `DeltaTimer`:

```
imu = RobotImu(mag_offsets=imu_settings.mag_offsets,
               gyro_offsets=imu_settings.gyro_offsets)
fusion = ImuFusion(imu)
timer = DeltaTimer()
```

3. Затем настройте ПИД- (или ПИ-) регулятор и робота:

```
pid = PIController(0.7, 0.01)
robot = Robot()
```

4. Добавьте две константы – базовую частоту вращения двигателей робота и уставку ПИД для направления (в градусах от севера):

```
base_speed = 70
heading_set_point = 0
```

5. Основной цикл обновляет таймер и объединение показаний датчиков IMU. Обратите внимание, что в следующем фрагменте кода отсутствует визуализация показателей:

```
while True:
    dt, elapsed = timer.update()
    fusion.update(dt)
```

6. Теперь вычисляем ошибку и передаем ее в ПИД-регулятор вместе с дельтой времени:

```
heading_error = fusion.yaw - heading_set_point
steer_value = pid.get_value(heading_error, delta_
time=dt)
```

7. Для отладки выводим значения на печать и настраиваем частоту вращения двигателей:

```
print(f"Error: {heading_error}, Value:{steer_
value:2f}, t: {elapsed}")
robot.set_left(base_speed + steer_value)
robot.set_right(base_speed - steer_value)
```

Загрузите код в память робота и запустите с помощью Python 3. Робот начнет двигаться на север. Если попробовать отклонить робота от курса, он скорректирует направление и снова повернется на север. Чем сильнее вы отвернете робота, тем быстрее двигатели вернуться к исходному курсу. Экспериментировать с этим очень увлекательно!

После того как вы закончите экспериментировать с разными направлениями, нажмите **Ctrl+C**, чтобы завершить программу.

В этом разделе вы еще раз увидели, как строятся блок-схемы потоков данных и как происходит разработка кода на их основе. Также вы узнали, что если преобразовать показания датчика в числовое значение, то их можно использовать в ПИД-алгоритме и создать соответствующий поведенческий сценарий. Затем вы научили робота двигаться по заданному (вычисленному нами ранее) направлению с помощью ПИД-регулятора и компаса.

## Выводы

В этой главе вы узнали, как получить приблизительное значение абсолютной ориентации робота в пространстве посредством объединения данных датчиков IMU. Вы увидели, как визуализировать получаемые данные в виде графиков и как создать модель виртуального робота. Затем вы узнали, как связать систему датчиков и ПИД-регулятор, чтобы заставить робота двигаться.

Вы узнали о том, какие математические операции применяются для преобразования векторных компонентов в углы в трехмерном пространстве, а также о том, как использовать дополнительные фильтры для компенсации шума в одной системе и отклонения в другой. Вы увидели, как посредством объединения показаний датчиков робот может получать представление о мире вокруг. Вы закрепили навыки по работе с блок-схемами и потоками данных, а также еще раз поработали с ПИД-алгоритмом.

В следующей главе вы узнаете, как управлять роботом и переключать поведенческие сценарии через меню на смартфоне.

## Задание

Следующие задания позволят вам лучше разобраться в теме и получить дополнительную информацию:

- вы можете улучшить виртуальную модель робота, используя более сложные формы и разные цвета. Это не является целью главы, но позволяет вам лучше разобраться в VPython и поэкспериментировать с ним;
- сейчас настройки магнитометра жестко закодированы в файле Python. Попробуйте загрузить настройки из файла данных. Узнать об этом подробнее вы можете по адресу <http://zetcode.com/python/yaml/>;
- подумайте, можно ли использовать нашу виртуальную модель для отбраковки показаний и отладки других датчиков и интеграций?
- можно ли заставить робота двигаться по квадратной траектории с точными поворотами, объединив абсолютное позиционирование и энкодеры?

## Дополнительные материалы

Больше информации по темам, описанным в этой главе, вы можете найти в следующих источниках:

- на сайте **Консорциума Всемирной Паутины (W3C – World Wide Web Consortium)** вы можете найти руководство по работе с магнитометрами в браузере. По адресу <https://www.w3.org/TR/magnetometer/> можно найти

интересную информацию о различных методиках. Также здесь вы можете узнать о том, как подобные алгоритмы работают в смартфонах для определения их ориентации;

- в этой главе часто упоминалась функция `atan2`. Узнать о ней больше вы можете по адресу <http://www.c-cpp.ru/content/atan2-atan2l>;
- по адресу <https://toptechboy.com/arduino-based-9-axis-inertial-measurement-unit-imu-based-on-bno055-sensor/> вы можете найти эксперименты по работе с Arduino и IMU, а также руководство по VPython от Пола Маквортера (Paul McWhorter). Этот материал сыграл важную роль при работе над книгой;
- по адресу [https://www.researchgate.net/publication/51873462\\_Data\\_Fusion\\_Algorithms\\_for\\_Multiple\\_Inertial\\_Measurement\\_Units](https://www.researchgate.net/publication/51873462_Data_Fusion_Algorithms_for_Multiple_Inertial_Measurement_Units) можно ознакомиться со статьей об использовании **глобальной системы позиционирования (GPS – Global Positioning System)** для объединения показаний датчиков;
- если вы хотите глубже изучить слияние датчиков и алгоритмы их объединения при фильтрации ошибок, для этого подойдут фильтры Калмана. Узнать больше вы можете в статье по адресу <https://towardsdatascience.com/sensor-fusion-part-1-kalman-filter-basics-4692a653a74c>.

# Глава 17

## Разработка кода на Python для управления роботом с помощью смартфона

У нашего робота есть множество поведенческих сценариев, некоторые из них предполагают движение, поэтому во время их выполнения он может просто уехать в другой конец комнаты. Чтобы заставить робота возвращаться, можно разработать специальный код, но это будет достаточно сложно. Сейчас при выполнении сценариев робот использует камеру с обратной связью, но почему бы не реализовать ручное управление?

До этого мы отправляли команды управления через терминал **SSH**, но гораздо интереснее и удобнее управлять роботом с помощью меню. Для этого мы можем использовать код **API** из главы 15.

В этой главе мы разработаем систему меню для смартфона, посредством которого можно будет управлять поведенческими сценариями. Затем создадим веб-приложение, с помощью которого вы сможете управлять роботом касаниями к сенсорному экрану вашего смартфона, ориентируясь при этом на данные с камеры. Вы научитесь управлять роботом самостоятельно, а также узнаете больше о готовых веб-приложениях для смартфонов.

В этой главе мы рассмотрим следующие темы:

- когда голосовое управление не работает (или зачем нам нужно управлять роботом);
- режимы меню – выбор поведенческих сценариев;
- выбор контроллера – как лучше управлять роботом и почему;
- подготовку Raspberry Pi к удаленному управлению – основы системы управления;
- реализацию полного управления роботом через смартфон;
- реализацию запуска меню вместе с Pi.

## ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Вам потребуются:

- робот с настроенной камерой и Raspberry Pi, а также код, разработанный в предыдущих главах;
- устройство с сенсорным экраном, например смартфон с доступом к Wi-Fi;
- беспроводная сеть.

Код из этой главы вы можете найти на GitHub по адресу <https://github.com/PacktPublishing/Learn-Robotics-Programming-Second-Edition/tree/master/chapter17>.

Полный код из предыдущих глав вы можете найти в папке `_starting_point`, а код из текущей главы в папке `full_system`.

Посмотреть видеоролик Code in Action на YouTube можно по адресу <https://bit.ly/2Kb7rp8>.

## Когда голосовое управление НЕ РАБОТАЕТ (или ЗАЧЕМ НАМ НУЖНО УПРАВЛЯТЬ РОБОТОМ)

В главе 15 мы реализовали систему запуска поведенческих сценариев с помощью голосового помощника Microsoft. Вы могли заметить, что в случаях, когда вам было необходимо остановить робота или заставить его повернуть налево/направо, системе требовалось некоторое время на обработку команд (даже если вы четко проговаривали их).

На самом деле голосовое управление нормально работает только в помещениях без постороннего шума. Если вы используете робота на улице (например, ему нужно куда-то поехать), такая система управления будет неэффективна.

Microsoft работает только с доступом к интернету. Одно дело – когда роботу и контроллеру достаточно небольшой общей сети, другое – когда робот требует постоянного подключения к интернету. Если вы используете его вне дома (например, в школе или в лаборатории), это может стать серьезной проблемой.

Раньше мы управляли роботом через терминал SSH – для этого нужно было начать сессию и ввести команду для запуска/остановки поведенческого скрипта. Такая система управления подходит только на этапах тестирования, но для обычного использования она слишком трудоемкая и неудобная. При неверном вводе команды или при необходимости перезагрузки сессии SSH тратится много времени.

Более удобный способ – использовать веб-приложение на смартфоне, позволяющее осуществлять точное управление роботом. Такие приложения могут работать в локальной сети и не требуют подключения к внешнему интернету. Через приложение вы можете заставить робота вернуться после успешного завершения сценария или остановить выполнение сценария при возникновении ошибок. Приложив немного усилий, вы можете реализовать отображение полезной (или просто интересной) информации о действиях робота.

## РЕЖИМЫ МЕНЮ – ВЫБОР ПОВЕДЕНЧЕСКИХ СЦЕНАРИЕВ

В этой книге вы создали множество различных поведенческих сценариев, а в дальнейшем их может стать еще больше. Как я упоминал ранее, управлять

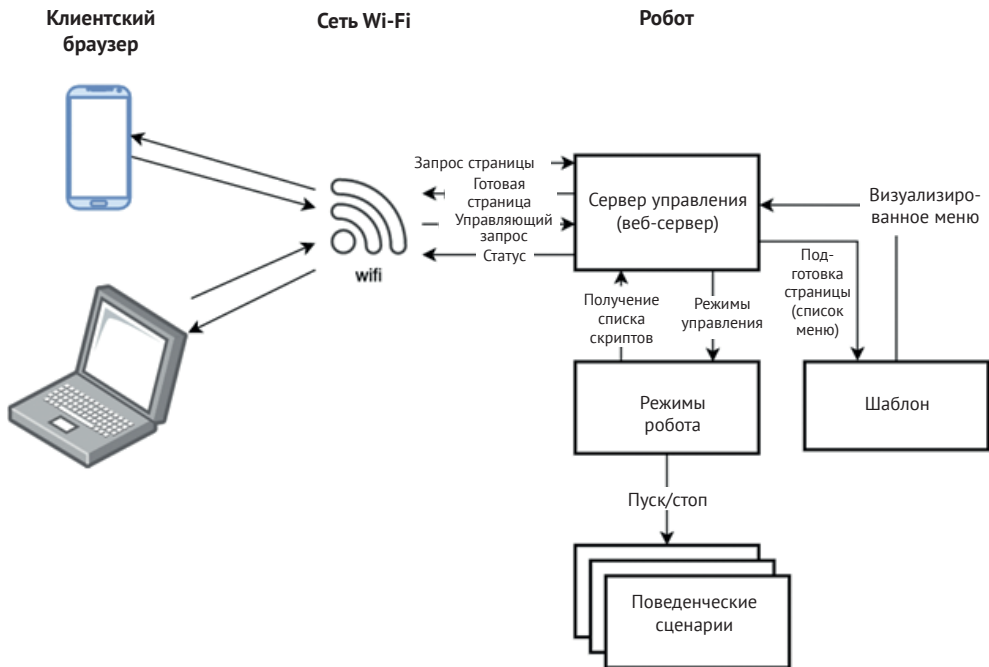


программами робота через SSH не совсем удобно. Даже простое сохранение параметров или остановка программы (для этого нужно использовать комбинацию **Ctrl+C**) может занимать слишком много времени.

В этом разделе мы разработаем систему меню для выбора программ. Для удобства мы реализуем ее в виде веб-приложения, доступного для смартфона. Тестировать систему мы будем в браузере на ПК.

Мы расширим систему, созданную в разделе «Удаленный запуск поведенческого скрипта» в главе 15, добавив в нее **пользовательский интерфейс (UI – User Interface)**. Для его создания мы воспользуемся шаблонами, в которых заменим кодом некоторые заполнители.

Обратите внимание на блок-схему на рис. 17.1.



**Рис. 17.1.** Как работает сервер управления и система меню

На рис. 17.1 показано, как работает наша система.

1. Клиентский браузер (на смартфоне или компьютере) через Wi-Fi отправляет на веб-сервер запрос для отображения страницы.
2. Веб-сервер получает список скриптов, которые он может запустить, из системы режимов робота.
3. Веб-сервер отправляет список режимов в шаблон, чтобы отобразить его на странице меню, которая затем отправляется пользователю.
4. Когда вы касаетесь сенсорных кнопок (ссылок пунктов меню) в браузере (или нажимаете на них мышкой), на веб-сервер отправляются соответствующие запросы.
5. Веб-сервер обрабатывает запросы управления, вызывая режимы управления (run и stop) из системы режимов робота.

6. Система режимов робота запускает/останавливает поведенческие скрипты.
7. Чтобы сообщить пользователю об успешной обработке, сервер управления отправляет статус в клиентский браузер.

Начнем с расширения списка скриптов (режимов) и рассмотрим, как система обрабатывает их.

## Управление режимами робота

Мы вернемся к коду из главы 15, расширим список режимов и добавим конфигурацию меню.

Сначала добавим в систему режимов новые элементы.

1. Откройте файл `robot_modes.py`.
2. Найдите переменную `mode_config`. Добавим в нее еще несколько скриптов, как показано в следующем фрагменте кода:

```
mode_config = {
    "avoid_behavior": "avoid_behavior.py",
    "circle_head": "circle_pan_tilt_behavior.py",
    "test_rainbow": "test_rainbow.py",
    "test_leds": "leds_test.py", "line_following": "line_follow_behavior.py",
    "behavior_line": "straight_line_drive.py",
    "drive_north": "drive_north.py"
}
```

3. После переменной `mode_config` добавляем список конфигурации меню. В браузере элементы меню будут отображаться в том же порядке, что и в этом фрагменте кода. У каждого элемента есть параметр `mode_name`, соответствующий короткому слагю в переменной `mode_config`, а также метка `text`, которая выводит текст в подходящем для чтения пользователем формате:

```
menu_config = [
    {"mode_name": "avoid_behavior", "text": "Avoid
Behavior"},
    {"mode_name": "circle_head", "text": "Circle
Head"},
    {"mode_name": "test_leds", "text": "Test LEDs"},
    {"mode_name": "test_rainbow", "text": "LED
Rainbow"},
    {"mode_name": "line_following", "text": "Line
Following"},
    {"mode_name": "behavior_line", "text": "Drive In
A Line"},
    {"mode_name": "drive_north", "text": "Drive
North"}
]
```

Помните, что если вы захотите расширить меню, то новые скрипты нужно добавлять как в переменную `menu_config`, так и в `mode_config`.

4. Чтобы пользователь мог поменять режим, не нажимая кнопку **Stop**, нам понадобится метод `run`, который будет останавливать текущий процесс:

```
def run(self, mode_name):
    while self.is_running():
        self.stop()
        script = self.mode_config[mode_name]
        self.current_process = subprocess.
Popen(["python3", script])
```

Этот файл будет работать как конфигурация, поэтому в дальнейшем вы можете расширить его. Сейчас можем проверить наш код.

5. Загрузите файл `robot_modes.py` в память робота. Также у вас должен быть загружен файл `control_server.py` из главы 15.
6. Запустите код на Pi с помощью команды `python 3 control_server.py`.
7. Как вы уже знаете из главы 15, для отправки запроса используется команда `curl`:

```
curl -X POST http://myrobot.local:5000/run/test_leds
```

Эта команда заставит **светодиоды** поочередно вспыхивать.

8. Чтобы поменять поведенческий сценарий необходимо остановить текущий и запустить новый. Введите следующую команду:

```
curl -X POST http://myrobot.local:5000/run/circle_head
```

Светодиоды погаснут, и, если двигатели включены, робот начнет двигаться.

9. Для остановки введите следующую команду:

```
curl -X POST http://myrobot.local:5000/stop
```

В файл `robot_modes.py` мы добавили несколько новых режимов и конфигурацию (для их описания), а затем проверили, работают ли они. Перейдем к устранению возможных неполадок.

## Устранение неполадок

Если запрос к серверу не выполняется, пользователь получает код ошибки. В нашей системе вам могут встретиться только три кода состояния:

- 200 – этот код говорит том, что система работает нормально. Даже если возникла какая-то логическая ошибка, которая не вызвала сбоя в системе, веб-сервер сообщит, что все в порядке;
- 404 – этот код сообщает, что сервер не может найти маршрут. Это может означать, что либо в запросе, либо в маршрутизаторах в коде сервера есть опечатка. Проверьте код и попробуйте снова;
- 500 – этот код означает, что сервер вышел из строя. Обычно после такой ошибки программа выдает исключение/трассировку. В коде на Python эта ошибка не страшна.

Теперь, когда списки конфигурации режимов готовы, нам нужен веб-сервер для отображения меню.

## Веб-сервер

В главе 15 мы уже подключили `robot_modes.py` к веб-серверу Flask `control_server.py`. В главе 13 мы использовали Flask для рендеринга шаблонов, отображающих потоковый вывод камеры. В этом разделе мы создадим шаблон пользовательского меню режимов.

Сначала внесем в код изменения, необходимые для рендеринга шаблона.

1. Откройте файл `control_server.py`.
2. К импортам для Flask добавьте `render_template`:

```
from flask import Flask, render_template
from robot_modes import RobotModes
```

3. Поскольку таблица стилей будет меняться, необходимо остановить работу компонентов, у которых есть устаревшие кешированные копии таблиц. Для этого добавляем заголовки ко всем ответам:

```
@app.after_request
def add_header(response):
    response.headers['Cache-Control'] = "no-cache,
    no-store, must-revalidate"
    return response
```

4. Для отображения меню добавляем маршрут. Создайте шаблон с именем `menu.html`, в котором за отображение будет отвечать переменная `menu_config`. Она нужна для работы большинства режимов. Для рендеринга шаблона добавьте код из следующего фрагмента:

```
@app.route("/")
def index():
    return render_template('menu.html', menu=mode_
    manager.menu_config)
```

Теперь у нас есть код для рендеринга шаблона, который мы создали на основе ранее созданного скрипта для обработки запросов `run` и `stop`. Прежде чем мы сможем запустить сервер, ему нужно предоставить шаблон – `menu.html`.

## Шаблон

Шаблон определяет отображение интерфейса, а также позволяет разделить то, как выглядит меню от того, как происходит обработка в системе управления. Этот шаблон сочетает в себе HTML (см. главы 13 и 14) и шаблонизатор **Jinja2**, который позволяет осуществлять замену данных. В разделе «Дополнительные материалы» вы можете найти больше информации о подобных системах. Мы добавим шаблон меню в папку `folder`, созданную в главе 15. Также в этот шаблон можно добавить стили, но пока что мы оставим его простым.

Создайте файл `templates/menu.html`, а затем выполните следующие шаги.

1. Шаблон начинается с заголовка, который устанавливает заголовок страницы. Также здесь мы используем уже знакомый инструмент `jQuery`:

```

<html>
<head>
  <script src="https://code.jquery.com/jquery-
3.5.1.min.js"></script>
  <title>My Robot Menu</title>
</head>

```

2. В тело шаблона помещаем заголовок My Robot Menu. Вы можете заменить этот заголовок на имя вашего робота. Код показан в следующем фрагменте:

```

<body>
  <h1>My Robot Menu</h1>

```

3. Далее оставляем место для сообщения. Пока оставляем его пустым:

```

<p id="message"></p>

```

4. Затем следует список – это и есть меню. Сначала идет тег <ul>, а затем цикл for, который создает элемент списка со ссылками для каждого элемента меню. Двойные скобки {{ }} используются для обозначения заполнителя, который будет заменен кодом при запуске. Здесь находится настройка mode\_name и text для создания ссылки. Далее следует /run и имя режима:

```

<ul>
  {% for item in menu %}
    <li>
      <a href="#" onclick="run('/run/{{ item.mode_
name }}')">
        {{ item.text }}
      </a>
    </li>
  {% endfor %}

```

Мы используем обработчик onclick, поэтому действие run можно обработать в коде.

5. Прежде чем завершить список, добавим еще один элемент меню – кнопку **Stop**:

```

<li><a href="#" onclick="run('/stop')">Stop</a></li></ul>

```

6. До этого мы упомянули обработку действия run в коде JavaScript. Код из фрагмента ниже выполняет запросы POST, отправляя данные на веб-сервер, а затем обновляет сообщение из ответа. Поместим его в теги <script>:

```

<script>
  function run(url) {
    $.post(url, '', response => $('#message').
html(response))
  }
</script>

```

Функция `run` вызывает метод `.post` с **URL** и пустым местом для данных. При получении ответа оно заполняется содержимым элемента сообщения. Оператор `=>` – это сокращение JavaScript для определения небольшой функции – в данном случае для той, которая имеет параметр `response`. В JavaScript функция может быть частью данных, поэтому очень часто одна функция передается как параметр другой. Таким функциям даже не даются имена – их называют анонимными функциями или лямбдами.

7. Наконец, мы закрываем HTML-документ:

```
</body>
</html>
```

Преимущество этого шаблона заключается в том, что можно просмотреть как выглядит код в браузере, не обращая к серверу. На рис. 17.2 показан скриншот режима предварительного просмотра.

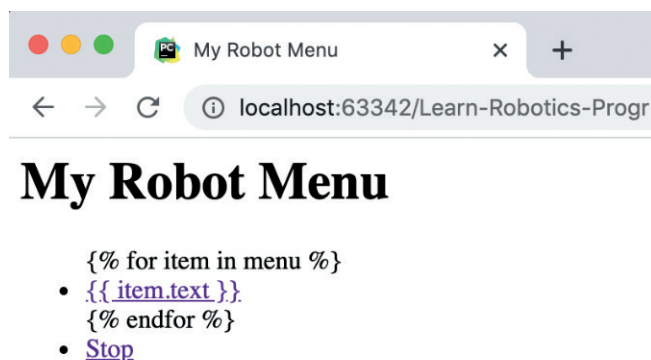


Рис. 17.2. Предварительный просмотр шаблона

Как показано на рис. 17.2, в режиме предварительного просмотра видны заполнители шаблона, поскольку браузер не знает, что их нужно заменить.

Теперь нужно запустить приложение и посмотреть, как оно выглядит на самом деле.

## Запуск приложения

Загрузите в память робота файлы `robot_modes.py` и `control_server.py`, а затем папку `templates`. Запустите приложение на Raspberry Pi через SSH с помощью следующей команды:

```
pi@myrobot:~ $ python3 control_server.py
* Serving Flask app "control_server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a
  production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 270-549-285
```

Чтобы увидеть меню, откройте браузер и подключитесь к порту вашего робота (<http://myrobot.local:5000/>). Результат показан на рис. 17.3.

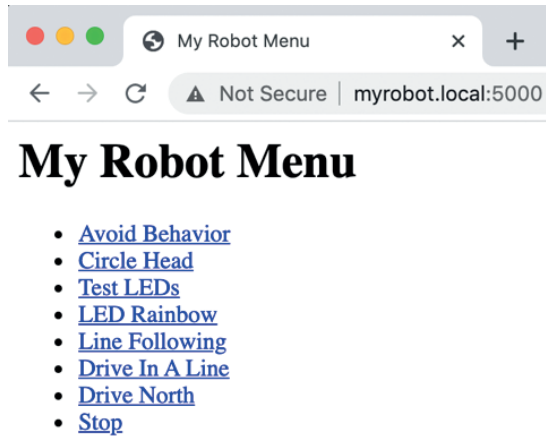


Рис. 17.3. Меню в браузере

На рис. 17.3 показан готовый список, где вместо заполнителей шаблона мы видим элементы меню. Нажмите на один из элементов, и робот начнет выполнение поведенческого сценария. При нажатии кнопки **Stop** код `robot_modes.py` отправляет эквивалент действия **Ctrl+C** в запущенный сценарий и останавливает его.

Когда вы запускаете сценарий или останавливаете его, в области сообщений появляется вывод, как показано на рис. 17.4.

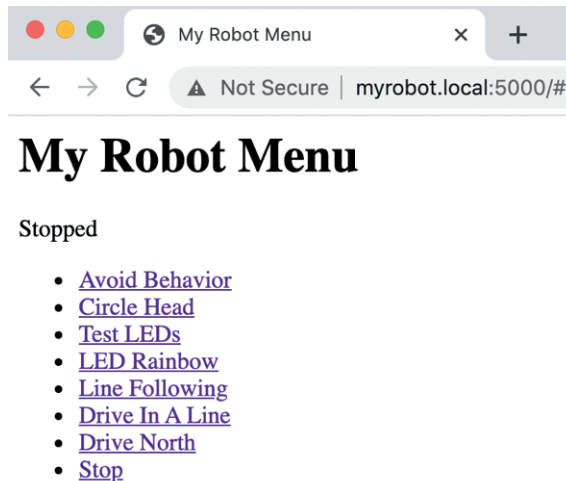


Рис. 17.4. Сообщение, возникшее после остановки сценария

На рис. 17.4 также показано меню. Я нажал кнопку **Stop**, и здесь появилось ответное сообщение **Stopped** (Остановлено).



Обратите внимание, что вывод сценария (его операторы `print`) отображается в консоли веб-сервера, как показано в следующем фрагменте кода:

```
192.168.1.149 - - [17/Oct/2020 22:42:57] "POST /run/test_leds
HTTP/1.1" 200 -
red
blue
red
blue
red
Traceback (most recent call last):
  File "leds_test.py", line 16, in <module>
    sleep(0.5)
KeyboardInterrupt
192.168.1.149 - - [17/Oct/2020 22:43:41] "POST /stop HTTP/1.1"
200 -
```

Чтобы завершить работу серверного приложения меню, нажмите **Ctrl+C**.

### Важное примечание

В этом небольшом веб-приложении не предусмотрен механизм защиты, аутентификация или пароли. Данная тема выходит за рамки этой книги, однако, если вы планируете использовать приложение при подключении к общедоступной сети Wi-Fi, стоит задуматься о безопасности.

Существуют и другие способы передать вывод консоли из скрипта на страницу. Я рекомендую ознакомиться с рекомендациями по работе с Flask, которые можно найти в разделе «Дополнительные материалы».

## Устранение неполадок

Надеюсь, что у вас все работает. Если возникли неполадки, воспользуйтесь следующими советами:

- в журнале вывода находятся коды возврата веб-системы. Вы можете использовать их для устранения неполадок, как делали это раньше;
- 200 – этот код состояния означает, что система в порядке. Если у вас не получается запустить режим, проверьте функцию `run`;
- 404 – не найдено. Проверьте маршруты;
- 500 – этот код сообщает о внутренней ошибке сервера и сопровождается ошибкой Python;
- если при отображении меню вместо названия режима вы видите `{ item.text }`, поставьте двойные фигурные скобки (это необходимо для работы системы шаблонов);
- при возникновении ошибки `TemplateSyntaxError: unexpected '<'` проверьте шаблон – возможно, вы пропустили закрывающую фигурную скобку `}`).

Итак, вы создали систему меню, с помощью которой можно запускать и останавливать различные поведенческие сценарии робота. Вы уже можете открыть ее в браузере на смартфоне, но пока что она не очень удобная. Мы рассмотрели тему поверхностно, и сейчас система находится лишь на начальном этапе разработки.

Мы рассмотрим более интересный интерфейс управления роботом для смартфона, но сначала нужно поговорить о других вариантах.

## ВЫБОР КОНТРОЛЛЕРА – КАК ЛУЧШЕ УПРАВЛЯТЬ РОБОТОМ И ПОЧЕМУ

Нам нужно управлять роботом с помощью портативного и беспроводного устройства. Ограничивать робота проводами не имеет особого смысла. В главе 7 мы научили его передвигаться с помощью колес, а теперь необходимо создать систему управления, которая будет напрямую взаимодействовать с ними.

Один из вариантов – Bluetooth-геймпад. Однако многим моделям, представленным на рынке, для считывания данных требуются специальные драйверы. Также Bluetooth может периодически терять сопряжение устройств в самый неподходящий момент.

Для некоторых геймпадов можно использовать донглы<sup>18</sup>. Такой подход безопаснее, чем просто использование Bluetooth, но для нашего робота он не совсем удобен.

Так или иначе, у вас уже есть подходящее портативное устройство, и находится оно у вас прямо в кармане – это смартфон. У него есть сенсорный экран, способный считывать тактильные сигналы. Разработав нужный код, вы сможете создать нечто похожее на *перископ* для робота, отображающий видео в середине экрана виртуального контроллера (управлять роботом, ориентируясь на камеру, может быть сложнее, чем делать это, смотря на него со стороны). Ранее мы уже создавали веб-приложения для подключения к роботу через Wi-Fi, поэтому подключить смартфон будет просто. Вместо того чтобы приобретать отдельный геймпад, мы будем использовать уже имеющийся смартфон, который позволит управлять роботом и смотреть на мир его глазами.

## Обзор будущего приложения

Прежде чем приступить к разработке веб-приложения для смартфона, нужно разобраться как оно будет выглядеть и работать. Чтобы представить, как выглядит система, ее можно нарисовать на бумаге или в графическом редакторе (тогда результат будет выглядеть как работа настоящего профессионала). На рис. 17.5 показан макет системы.

На рис. 17.5 показан макет на экране смартфона в горизонтальном режиме. В верхней части есть кнопка **Exit** (Выход), которую позже мы будем использовать для выхода в меню режимов.

В центре экрана находится видеопоток камеры (мы реализуем это с помощью механизма из главы 13). Слева и справа показаны слайдеры (ползунки). На рис. 17.6 показано, как они работают.

<sup>18</sup> Донгл (англ. *Dongle*) – беспроводное периферийное устройство, например внешний порт связи типа Bluetooth. Обычно выполняет функцию шифрования данных, а по внешнему виду напоминает заглушку или ключ, подключаемый к порту USB компьютера. – Прим. ред.

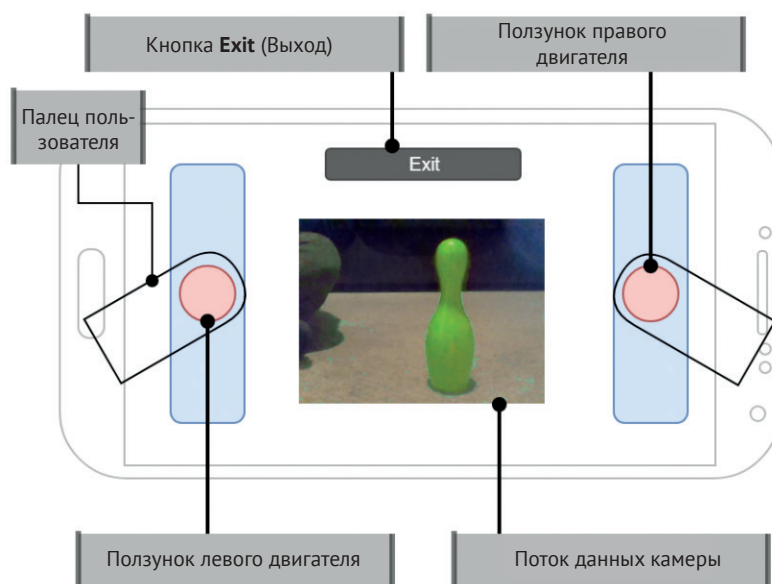


Рис. 17.5. Макет веб-приложения для управления

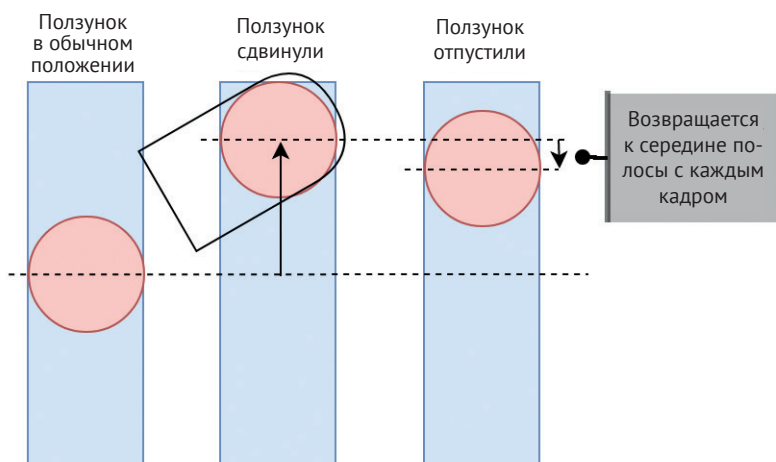


Рис. 17.6. Ползунок возвращается к середине полосы

На рис. 17.6 показан принцип действия ползунков. Как и в случае с аналоговым геймпадом, их можно перетаскивать в любое место на полосе, но делается это путем касаний к сенсорному экрану. Когда вы отпускаете ползунок, он перемещается к середине полосы.

Обратите внимание, что ползунок перемещается к середине не моментально – для этого требуется несколько кадров. Реализовать это в коде нам помогут математические операции.

Ползунки позволяют управлять роботом как танком с двумя гусеницами (при управлении геймпадом для этого используются аналоговые стики). Каждый ползунок отвечает за частоту вращения одного двигателя. Несмотря на то что это звучит достаточно сложно (не так легко, как водить машину), для управления двухколесным роботом такой способ является наиболее простым (особенно если вы начали изучать робототехнику совсем недавно). Чем выше вы переместите ползунок по полосе, тем быстрее будет вращаться соответствующий двигатель. Также мы реализуем сценарий, при котором в случае потери соединения двигателя будут останавливаться через секунду.

Описанная система, в которой для каждого двигателя можно отдельно настроить частоту вращения, в сущности, является той же системой управления поведенческими сценариями, которая использовалась нами ранее в этой книге, только теперь она стала интерактивной. На рис. 17.7 показано несколько вариантов положений ползунков (для стандартных движений робота).

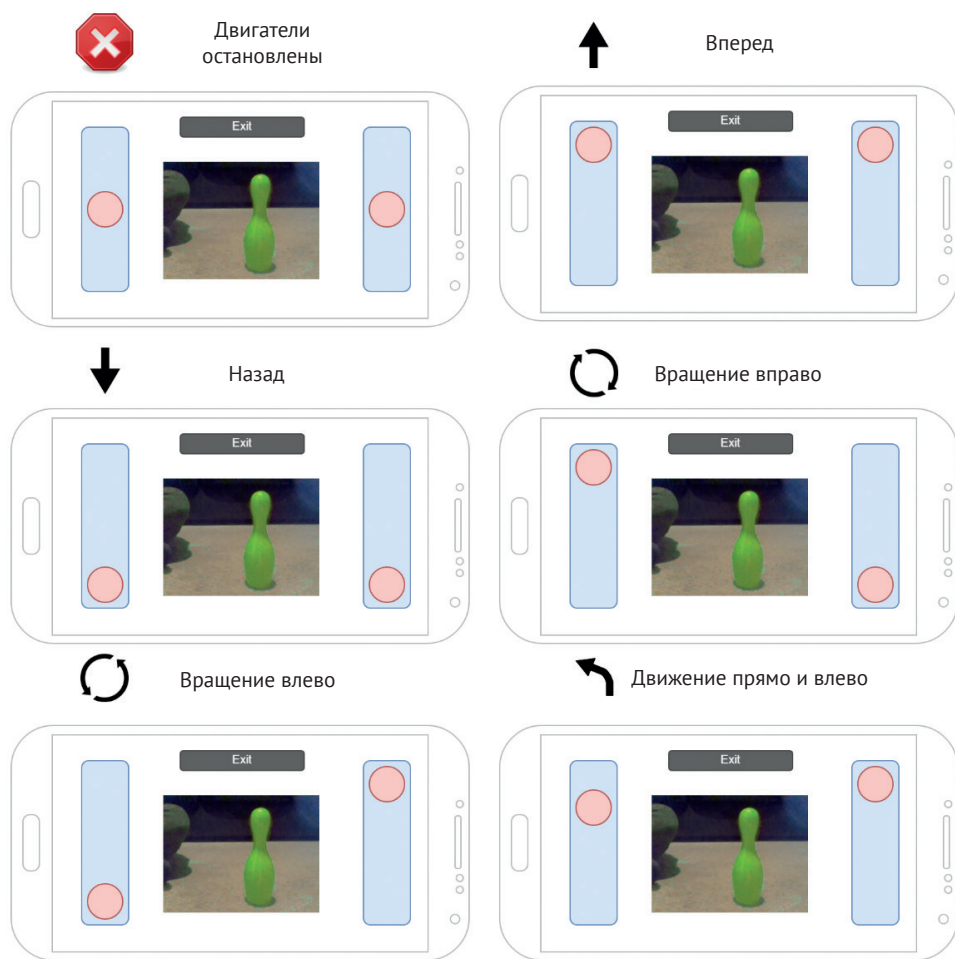


Рис. 17.7. Положения ползунков для стандартных движений робота

Красным на рис. 17.7 отмечены точки соприкосновения ваших пальцев с экраном. Если сдвинуть оба ползунка вверх, робот поедет вперед (чем выше ползунки, тем выше скорость). Если переместить оба ползунка вниз, робот поедет назад. Чтобы робот начал выполнять повороты на месте, сдвиньте ползунки в противоположные стороны. Чтобы робот поехал вперед и немного повернул влево/вправо, ползунки нужно сдвинуть так чтобы правый был выше, чем левый (или наоборот). Используя этот подход, можно компенсировать отклонение от курса.

Итак, у нас есть макет удобного пользовательского интерфейса. Прежде чем перейти к коду, необходимо рассмотреть, какие блоки будут в него входить. Затем мы создадим код и посмотрим, как работает система на практике.

## Подготовка Raspberry Pi к удаленному управлению – ОСНОВЫ СИСТЕМЫ УПРАВЛЕНИЯ

Наш Raspberry Pi уже умеет запускать веб-службы, например использовать веб-сервер Flask для меню и видеосерверов. Также вы знаете, как реализовать взаимодействие поведенческого сценария и веб-сервера, используя очереди изображений и очереди управления. В этом разделе мы еще раз воспользуемся этим способом. Ползунки в приложении для смартфона должны быть «интеллектуальными». На рис. 17.8 показано, как будет устроена наша система ручного управления.

На рис. 17.8 поля, выделенные пунктирной линией, показывают, где выполняется код: в верхнем поле показан код, запускаемый на смартфоне, а в нижнем – на Raspberry Pi. Внутри есть другие поля, выделенные сплошной линией. В них показаны блоки или системы, которые будут использоваться в коде. В нижней части рисунка показано поле **Робот**, которое принимает вызовы **остановки двигателей** и **частоты вращения двигателей**. Эти вызовы исходят из поля **Поведенческий сценарий** и зависят от установленного временного предела или данных **очереди управляющих сообщений** от **Веб-сервера Flask**. Тем временем цикл **Поведенческий сценарий** также принимает **фреймы изображений** от **камеры**, кодирует их и передает в **очередь фреймов для отображения**.

Уровнем выше находится **Веб-сервер Flask**. Он передает данные из **очереди фреймов для отображения** в **поток изображений**, состоящий из нескольких частей. Сервер Flask обрабатывает **управляющие запросы** и передает их в **очередь управляющих сообщений**.

Блок **Скрипт страницы** обрабатывает **изменение положения ползунков** и преобразует их в **управляющие запросы** с помощью библиотеки jQuery. Блок **Устройство с ползунками** преобразует **прикосновения к экрану** в **изменение положения ползунков** (он отвечает за анимацию и преобразование данных).

Для отображения видеопотока и подвижных ползунков на странице по-прежнему используется тег `img`. Кнопка **Exit** (Выход) осуществляет соответствующий управляющий запрос.

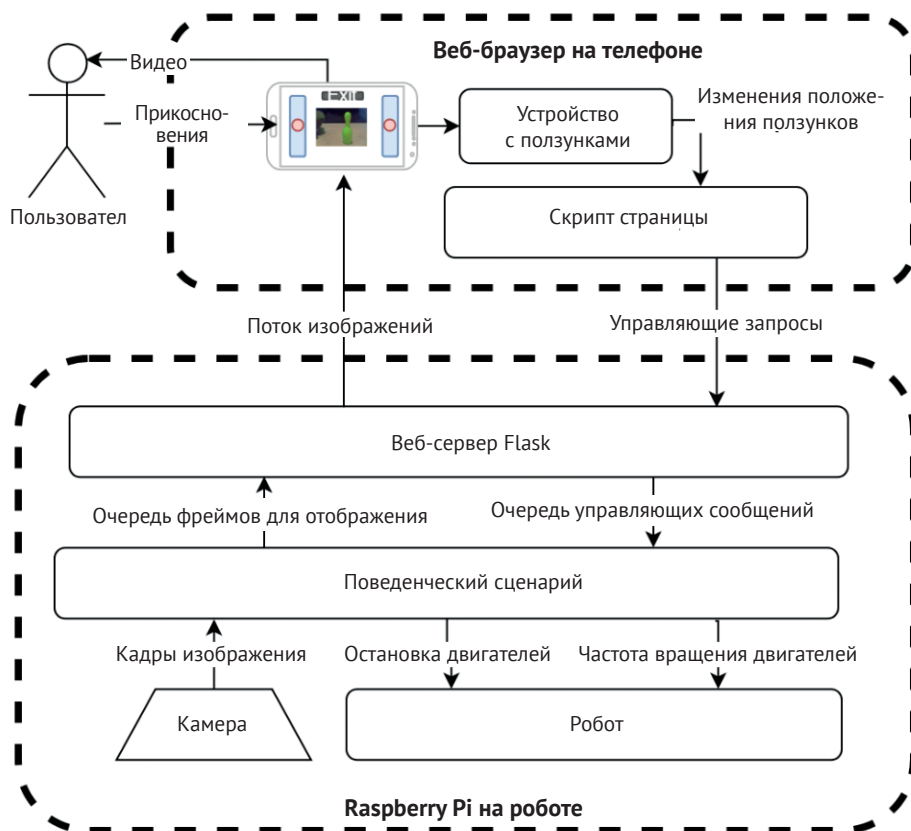


Рис. 17.8. Обзор системы приложения для ручного управления

Для блоков **Скрипт страницы** и **Устройство с ползунками** мы будем использовать JavaScript и формальный язык **CSS (Cascading Style Sheets – каскадные таблицы стилей)**. Прежде чем начать, вам понадобится ядро изображения (файл `image_app_core.py`) из главы 14. Мы создадим несколько дополнительных функций для передачи кода в браузер.

## Расширение кода ядра изображения

В этом разделе мы будем повторно использовать ядро изображения (файл `image_app_core.py` из главы 14), предварительно добавив ссылки на нужные статические файлы.

Статические файлы не вызывают каких-либо действий и обслуживаются системой пассивно. Мы будем использовать JavaScript и CSS вместе с локальной копией библиотеки jQuery. Flask делает это автоматически.

Для начала нам понадобится папка для статических файлов.

1. Создайте папку `static`. В нее мы поместим код JavaScript и CSS.
2. Нам нужна локальная копия библиотеки jQuery. Создайте директорию `lib` в папке `static`.

3. Скачайте код jQuery (<https://code.jquery.com/jquery-3.5.1.min.js>), нажмите в браузере кнопку **Save** и поместите его в папку `lib`. У вас должен получится файл `static/lib/jquery-3.5.1.min.js`.
4. В файле `image_app_core.py` нам нужно указать коду прекратить использовать кешированные файлы, чтобы он перезагрузил файлы CSS и JavaScript:

```
@app.after_request
def add_header(response):
    response.headers['Cache-Control'] = "no-cache,
no-store, must-revalidate"
    return response
```

Теперь в коде ядра изображения есть статическая копия jQuery, которую мы можем использовать офлайн. Благодаря этому при взаимодействии с роботом вам больше не придется беспокоиться о состоянии интернет-соединения.

## Разработка сценария ручного управления

Теперь нам нужен поведенческий скрипт. Мы разработаем его на основе кода из главы 14 и добавим управляющие сообщения, позволяющие менять частоту вращения двигателей, а также простой видеовывод.

В системе будет предусмотрена остановка робота, если управляющие сообщения не будут поступать в течение секунды. Нам нужно, чтобы робот не уезжал далеко или не съезжал со стола, поэтому научим его останавливаться в нужный момент.

Перейдем к разработке кода.

1. Запустите файл `manual_drive.py` с импортами для камеры и управления:

```
import time
from robot import Robot
from image_app_core import start_server_process, get_
control_instruction, put_output_image
import camera_stream
```

2. Далее объявите предел времени для остановки в секундах, как показано в следующем фрагменте кода:

```
TIMEOUT_IN_SECONDS = 1
```

3. Создайте класс `ManualDriveBehavior`. Здесь мы будем хранить объект `robot` и отслеживать время:

```
class ManualDriveBehavior(object):
    def __init__(self, robot):
        self.robot = robot
        self.last_time = time.time()
```

4. Далее переходим к разделу управления. Он будет сбрасывать последнее значение времени для каждой инструкции:



```
def process_control(self):
    instruction = get_control_instruction()
    while instruction:
        self.last_time = time.time()
        self.handle_instruction(instruction)
        instruction = get_control_instruction()
```

В ожидании данных от камеры в цикл могут быть помещены несколько инструкций, поэтому, пока они не закончатся, цикл будет повторяться. За одной инструкцией сразу следует другая. Затем цикл делегирует обработку методу `self.handle_instruction`.

5. Код обрабатывает инструкцию в `handle_instuction`. Инструкция представляет собой словарь, в качестве элементов которого выступают ее имя и параметры. Мы можем проверить, какая это команда – `set_left` или `set_right`, как показано в следующем фрагменте кода:

```
def handle_instruction(self, instruction):
    command = instruction['command']
    if command == "set_left":
        self.robot.set_left(int(instruction['speed']))
    elif command == "set_right":
        self.robot.set_
right(int(instruction['speed']))
```

Если команда подходит нам, устанавливаем частоту вращения двигателей. Сейчас частота вращения представлена в коде в виде строки, поэтому для двигателей ее значение необходимо преобразовать в целое число с помощью функции `int`.

6. Также нам потребуется выполнить обработку команды `exit`:

```
elif command == "exit":
    print("stopping")
    exit()
```

7. Нам необходимо, чтобы код отображал неизвестные инструкции (при их наличии). Особенно это пригодится на этапе проверки. Для этого мы вызываем исключение:

```
else:
    raise ValueError(f"Unknown instruction:
{instruction}")
```

8. Наше приложение должно быть способно отображать информацию, помещая кадры в очередь изображений сервера:

```
def make_display(self, frame):
    encoded_bytes = camera_stream.get_encoded_bytes_
for_frame(frame)
    put_output_image(encoded_bytes)
```

9. Для выполнения настройки и запуска главного цикла в коде используется метод `gcp`. Сначала нужно настроить положение механизма поворота и наклона (чтобы он был обращен прямо вперед), дать камере время для запуска и остановить работу сервоприводов:

```
def run(self):
    self.robot.set_pan(0)
    self.robot.set_tilt(0)
    camera = camera_stream.setup_camera()
    time.sleep(0.1)
    self.robot.servos.stop_all()
    print("Setup Complete")
```

10. Затем проходимся по кадрам с камеры и обрабатываем управляющие команды:

```
for frame in camera_stream.start_stream(camera):
    self.make_display(frame)
    self.process_control()
```

11. Наконец, реализуем автоматическую остановку двигателей в соответствии с установленным пределом времени:

```
if time.time() > self.last_time + TIMEOUT_IN_SECONDS:
    self.robot.stop_motors()
```

12. Для создания и запуска компонентов добавляем высокоуровневый код:

```
print("Setting up")
behavior = ManualDriveBehavior(Robot())
process = start_server_process('manual_drive.html')
```

13. Нам нужно, чтобы при выходе или возникновении ошибки сервер останавливал работу. Для этого используем код из следующего фрагмента:

```
try:
    behavior.run()
except:
    process.terminate()
```

Итак, работа над серверной частью поведенческого сценария завершена. Теперь нам нужно создать шаблон (для отображения), а также разработать код и стиль для запуска приложения на смартфоне.

## Шаблон (веб-страница)

В шаблон мы поместим ползунки и код для обработки их положения.

Приступим к созданию.

1. Создайте файл `templates/manual_drive.html`. Начните с преамбулы HTML:

```
<html>
  <head>
```

2. Интерфейс приложения должен быть адаптивным и подходить по размеру для экрана смартфона. При этом важно, чтобы случайными касаниями пользователь не менял масштаб. Строка из фрагмента кода ниже сообщает это браузеру:

```
<meta name="viewport" content="width=device-
width, initial-scale=1.0, user-scalable=no">
```

3. Этому интерфейсу (а впоследствии и другим) нужно добавить стиль. Для этого используем таблицу стилей `display.css`, как показано в следующем фрагменте кода:

```
<link rel="stylesheet" type="text/css" href="/
static/display.css?">
```

4. Чтобы сделать элементы интерфейса интерактивными, мы реализуем систему сенсорных ползунков, используя библиотеку jQuery. В следующем фрагменте кода показаны HTML-эквиваленты импортов:

```
<script src="/static/lib/jquery-3.5.1.min.js"></
script>
<script src="/static/touch-slider.js?"></script>
```

Сюда мы поместим конкретный фрагмент стиля. Остальное будет находиться в таблице стилей. Нам необходимо, чтобы интерфейс приложения сразу занимал весь экран устройства (без необходимости его прокрутки пользователем). Для этого добавим следующий фрагмент кода:

```
<style>html, body {margin: 0; height: 100%;
overflow: hidden}</style>
```

5. Чтобы заголовок переместился в верхнюю часть страницы, завершаем его HTML тегом `<title>`:

```
<title>Manually Drive The Robot</title>
</head>
```

6. Переходим к телу документа. Начнем с первого ползунка и определим его с помощью языка **SVG (Scalable Vector Graphics – масштабируемая векторная графика)**. SVG предназначен для описания графики внутри веб-страницы. С его помощью мы определим полосу, по которой будет двигаться ползунок. Для интеграции SVG и HTML помещаем нужные части в тег `svg`, как показано в следующем фрагменте кода:

```
<body>
<svg id="left_slider" class="slider_track"
viewBox="-10 -100 20 200"><le=no">
```

С помощью класса `slider_track` задаем стиль для двух полос – мы используем классы HTML для идентификации нескольких объектов. Идентификатор `left_slider` поможет позиционировать ползунок и добавить события касаний. Как правило, идентификаторы в HTML уникальны и используются для ссылки на один объект.

Атрибут `viewBox` определяет параметры положения поломы для `svg`: нижние координаты `x` и `y`, ширину и высоту. При переносе элемента `svg` в код для других устройств эти координаты по-прежнему будут актуальны. Диапазон высоты составляет от `-100` до `+100` (соответствует частоте вращения двигателя), а ширины – от `-10` до `+10`.

7. Внутри полосы мы обозначим круг. Значение его радиуса (`r`) можно поместить к значениям в `viewBox`. Для обоих направлений центром будет являться `0`. Введите код из следующего фрагмента:

```
<circle r="18" class="slider_tick"/>
```

8. Чтобы мы могли завершить поведенческий сценарий, добавим ссылку Exit. Она представляет собой класс и идентификатор:

```
<a class="button" id="exitbutton" href="/
exit">Exit</a>
```

С классом button мы свяжем шрифты и цвета, поэтому в дальнейшем можно будет использовать этот стиль для других кнопок (которые могут быть добавлены при расширении меню). Чтобы поместить кнопку в нужное место, используем идентификатор exitbutton.

9. Теперь переходим к блоку, в котором будет отображаться видео с камеры. Тег `img` помещаем в тег `div`, что позволит сохранять пропорции (менять размер) блока при отображении на любом экране:

```
<div id="video"></div>
```

10. Для правого ползунка используем тот же код, что и для левого, но меняем идентификатор (скопируйте и вставьте, заменив нужную часть):

```
<svg id="right_slider" class="slider_track"
viewBox="-10 -100 20 200">
  <circle r="18" class="slider_tick"/>
</svg>
```

11. Теперь добавим в HTML-документ код на JavaScript. Код на странице свяжет код ползунка с имеющимися графическими элементами и двигателями. Сначала объявляем блок JavaScript:

```
<script type="text/javascript">
```

12. Затем добавляем функцию для отправки роботу данных для управления двигателями. Она принимает имя (левый или правый двигатель) и частоту вращения:

```
function set_motor(name, speed) {
    $.post('/control', {'command': 'set_' +
name, 'speed': speed});
}
```

Отправляем эту инструкцию на сервер с помощью запроса POST.

13. Следующий фрагмент кода должен запускаться только после завершения загрузки страницы. Нам нужно проверить, загружены ли предыдущие библиотеки JavaScript. В jQuery есть специальная функция `$(())`, которая запускает любую переданную ей функцию после завершения загрузки страницы:

```
$(()) => {
```

14. Далее нужно связать кнопку выхода с запросом POST, который после завершения сценария будет перенаправлять нас в меню:

```
$('#exitbutton').click(function() {
    $.post('/control', {'command':
'exit'});
    window.location.replace('// ' +
window.location.hostname + ":5000");
});
```

15. Настраиваем ползунки и связываем их с их идентификаторами элементов svg и set\_motor, чтобы они обновлялись при каждом изменении положения:

```
makeSlider('left_slider', speed => set_
motor('left', speed));
makeSlider('right_slider', speed => set_
motor('right', speed));
});
```

Для каждой стороны мы используем элемент makeSlider, где id – это идентификатор объекта, который мы превращаем в ползунок (полоса svg), а также функцию, вызов которой происходит при изменении положения ползунка.

16. Завершаем страницу и закрываем все теги:

```
</script>
</body>
</html>
```

Сейчас у страницы нет стиля. По умолчанию у блока видео и ползунков не определена форма, размер или цвет, поэтому если вы попытаетесь загрузить этот код, то увидите пустую страницу. Далее мы сообщим браузеру, где должны располагаться элементы и какого цвета они должны быть. Также у нас еще нет кода для ползунков.

Мы разработали код, где с помощью тегов определили кнопку выхода и ссылки на ползунки. В следующем разделе мы добавим таблицу стилей, которая позволит отобразить все это на странице.

## Таблица стилей

Пришло время стилизовать наше приложение. Таблицы стилей достаточно сложны, и на их правильную настройку уходит много времени, так что мы коснемся этой темы лишь поверхностно. По желанию вы можете выбрать для приложения другие цвета. Я рекомендую использовать цвета, предложенные в стандарте W3C (см. <https://www.w3schools.com/colors/default.asp>). Можете пользоваться как названиями цветов, так и их шестнадцатеричными кодами (например, #1ab3c5).

CSS позволяет выделить элемент на странице и связать с ним атрибуты стиля. Разделы в CSS начинаются с **селекторов**, которые выбирают элементы HTML страницы для применения стиля. Это могут быть имена тегов, имена классов с префиксами в виде точек или идентификаторы с префиксами в виде символа #. В разделе «Дополнительные материалы» вы можете найти подробный обзор селекторов CSS. Для разграничения стилей раздела каждый из них заключается в фигурные скобки ({}). Стили разделов состоят из имени свойства, двоеточия (:) и настроек (значений). После каждой настройки ставится точка с запятой (;).

Перейдем к созданию стиля.

1. Создайте файл static/display.css, в котором мы будем хранить информацию о стиле.

Мы можем задать ширину дорожки ползунка равной 10 % относительно ширины окна просмотра (т. е. 10 % от размера экрана). В CSS для этого есть специальная единица измерения vw. С помощью единицы vh задаем высоту дорожки относительно высоты окна просмотра. В разделе «Дополнительные материалы» вы можете найти больше информации о единицах измерения в CSS. В нашем коде мы используем селектор CSS `.slider_track`, который выбирает все объекты класса для применения стиля. Этот класс используют оба ползунка поэтому и изменения применяются к ним обоим. Код показан в следующем фрагменте:

```
.slider_track {
  width: 10vw;
  height: 90vh;
```

2. В соответствии с нашими макетами задаем дорожке ползунка голубой фон и синюю границу в виде сплошной линии:

```
border: 1px solid blue;
background-color: lightblue;
}
```

3. Как и на макете, для стилизации ползунка (в форме круга) мы выбираем светло-розовую заливку:

```
.slider_tick {
  fill: mistyrose;
}
```

4. Теперь нам нужно расположить ползунки слева и справа (в соответствии с идентификаторами). Чтобы положение ползунков точно соответствовало нашему макету, мы используем **абсолютное позиционирование** относительно окна просмотра (в процентах):

```
#left_slider {
  position: absolute;
  left: 5vw;
  top: 5vh;
}
#right_slider {
  position: absolute;
  right: 5vw;
  top: 5vh;
}
```

5. На этом этапе можно посмотреть, как выглядит наша страница, для этого нужно загрузить ее, остановить выполнение текущего поведенческого сценария, затем снова запустить его и перезагрузить. Ползунки выглядят как надо, но кнопка выхода и блок видео находятся не в том месте.
6. Сделаем кнопку выхода больше похожей на кнопку. Стили для `.button` будут применяться ко всем кнопкам того же класса. Мы сделаем из кнопки блок – элемент, у которого есть такие свойства, как ширина и высота. Высота блока будет составлять 10 % от высоты окна просмотра, как показано в следующем фрагменте кода:

```
.button {
  display: block;
  height: 10vh;
```

7. Теперь выравниваем текст по середине блока, используя свойства `line-height` и `text-align`. Чтобы увеличить шрифт в два раза относительно обычного размера, используем значение `2e`:

```
text-align: center;
font-size: 2em;
line-height: 10vh;
```

8. Как и все ссылки, текст нашей кнопки подчеркнут снизу. Это подчеркивание нужно убрать. Также зададим цвет блоку (синий) и тексту (белый):

```
text-decoration: none;
background-color: blue;
color: white;
}
```

9. Добавляем кнопке еще параметры, используя ее идентификатор. Настраиваем ширину блока и определяем отступ сверху. Для центрирования блока слева и справа используем отступы `auto`:

```
#exitvutton {
  width: 40vh;
  margin-top: 5vh;
  margin-left: auto;
  margin-right: auto;
}
```

Попробуйте загрузить страницу. Сейчас кнопка должна находиться в нужном месте.

10. Теперь стилизуем блок видео. Нам нужно поместить его в центр экрана. Для этого используем внешний видеоэлемент:

```
.video {
  text-align: center;
}
```

11. Затем нужно указать положение и размер внутреннего блока изображения. Верхний отступ будет составлять 20 % от ширины экрана (указываем в `vh`). Единица измерения `vmin` – это процент от меньшего размера окна браузера по ширине или высоте. Использование этой единицы дает нам гарантию, что блок видео не будет настолько большим, чтобы закрыть собой два ползунка. Высоту определяем автоматически. Для того чтобы применить стиль к объекту `img`, содержащемуся в объекте `object`, мы используем селектор `#video img`:

```
#video img {
  margin-top: 20vh;
  width: 80vmin;
  height: auto;
}
```

Теперь страница полностью стилизована. Мы можем загрузить ее и посмотреть, как она выглядит. Загрузите всю папку (в том числе и шаблоны)



в память робота, а затем используйте команду `python3 manual_drive.py`. Чтобы увидеть страницу, в браузере на вашем компьютере подключитесь к <http://myrobot.local:5001/> (при необходимости заменив имя хоста и адрес). Сначала нужно просмотреть страницу в браузере на компьютере, поскольку так легче всего обнаружить ошибки в коде HTML или JavaScript. На момент написания этой книги эмуляцию мобильных устройств и события касаний поддерживают такие браузеры, как Firefox и Chrome. Страница должна выглядеть так, как показано на рис. 17.9 (полностью соответствовать макету и отображать видео с камеры).

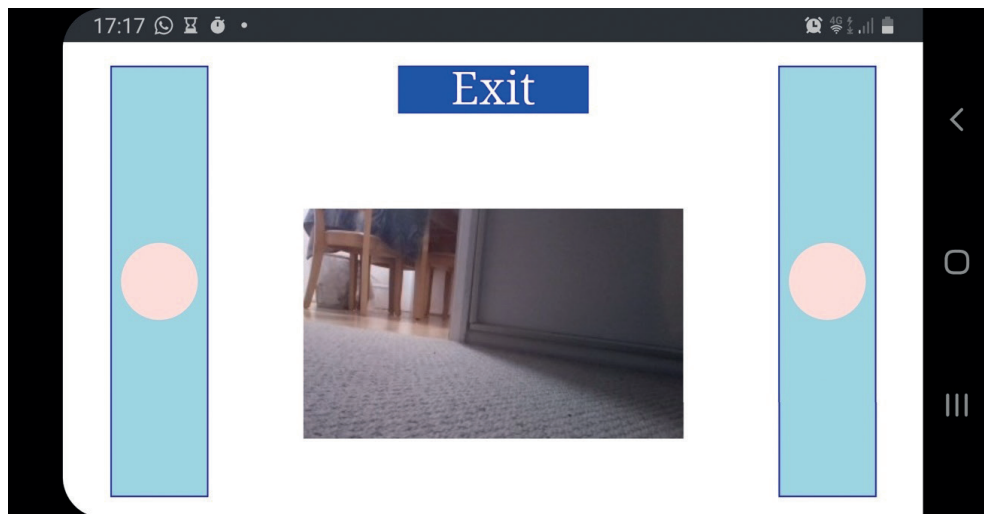


Рис. 17.9. Скриншот приложения, запущенного на смартфоне

На рис. 17.9 показано приложение, запущенное на смартфоне. На данный момент ползунки еще не работают. Возможно, вам понадобится принудительно перезагрузить таблицу стилей в браузере.

Перейдем к разработке кода для ползунков.

## Разработка кода для ползунков

Ползунки должны реагировать на события касаний. При прикосновении круг должен перемещаться в соответствующее место, а система должна отправлять сообщение об обновлении, указывающее, насколько далеко круг переместился от середины. Если не касаться экрана, ползунки вернутся в исходное положение (в центр дорожки). JavaScript позволяет нам запускать код в браузере, поэтому создадим функцию `makeSlider`.

Для начала разберемся, как прикосновения меняют положение ползунков и частоту вращения двигателя. На рис. 17.10 показано, как это работает.

Как показано на рис. 17.10 положение ползунков – это достаточно сложно. Когда пользователь прикасается к экрану, положение ползунка определяется координатами экрана. Сначала необходимо определить положение внутри дорожки ползунка, для этого вычитаем отступ сверху. Полученный результат делим на высоту дорожки, умножаем на 200 и вычитаем 100 – таким образом

вычисляем положение ползунка относительно окна просмотра (такой же способ используется для описания графики с помощью SVG). В координатах окна просмотра отступ сверху составляет  $-100$ , но, чтобы робот поехал вперед, нам нужно значение  $+100$ . Таким образом, чтобы получить значение частоты вращения двигателя, мы выполняем операцию смены знака.

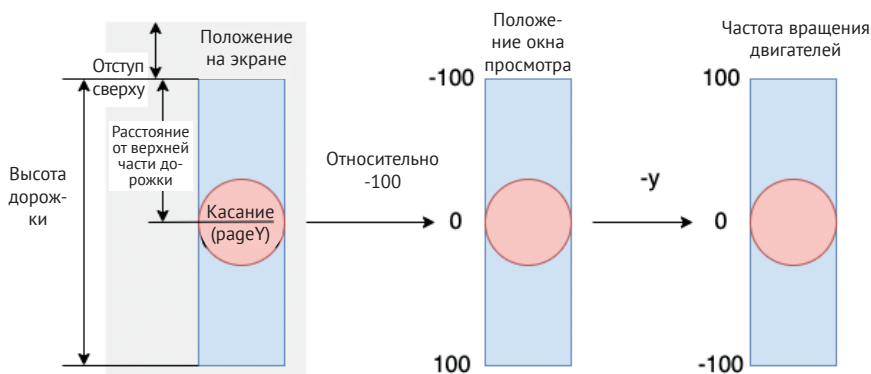


Рис. 17.10. Преобразование событий касаний в изменение частоты вращения двигателей

Наш скрипт настроит данные необходимые для перемещения ползунка и внутренние функции для сопоставления положения ползунка и события касания. При перемещении ползунков скрипт будет выполнять обратный вызов кода `manual_drive.html` (или любого другого). Чтобы создать код для ползунка, выполните следующие шаги.

1. Мы поместим скрипт в файл `static/touch-slider.js`. Поскольку он имеет расширение `.js`, теги `<script>` нам не понадобятся.
2. Все, что нужно для наших ползунков, будет находиться в фабричной функции `makeSlider`:

```
function makeSlider(id, when_changed) {
```

3. Сначала нам понадобятся внутренние данные. Код должен понимать, когда мы касаемся ползунка, чтобы во время касания он не возвращался в исходное положение. Код должен знать, изменилась ли точка прикосновения, и отслеживать ее положение. Нам нужно определить ползунок по идентификатору и сохранить найденный объект для дальнейшего использования:

```
let touched = false;  
let changed = false;  
let position = 0;  
const slider = $('#' + id);
```

4. Для взаимодействия с ползунком нам понадобятся дополнительные функции. Начнем с функции для обновления положения, которая будет гарантировать, что положение ползунка изменилось. Здесь подходят только целые числа (браузер не принимает точки с десятичными значениями). Затем обновляется значение признака состояния `changed`, как показано в следующем фрагменте кода:

```
const set_position = function(new_position) {
  position = Math.round(new_position);
  slider.find('.slider_tick')[0].setAttribute('cy',
position);
  changed = true;
};
```

- Следующий этап – обработка событий касаний. Обработчики касаний – это функции, вызываемые при каком-либо событии (например, обработчик нажатия кнопки выхода). Касание состоит из трех событий: `touchstart` – когда пользователь касается экрана, `touchmove` – когда пользователь перемещает палец по экрану и `touchend` – когда пользователь убирает палец от экрана. Функцию `touchstart` мы использовать не будем, поэтому начнем с создания анонимной функции `touchmove`:

```
slider.on('touchmove', event => {
  let touch = event.targetTouches[0];
```

Обратите внимание, что из данных события мы сразу получаем переменную `touch`. Мы получаем список касаний, но используем только первое.

- Затем определяем положение этого касания относительно верхней части ползунка:

```
let from_top = touch.pageY - slider.offset().top;
```

- Учитывая полученный результат и высоту, мы преобразуем положение точки касания в число от  $-100$  до  $+100$ , которое соответствует координате окна просмотра SVG:

```
let relative_touch = (from_top / slider.height())
* 200;
set_position(relative_touch - 100);
```

- Когда код получает событие касания, признаку состояния `touched` дается значение `true`. Также мы должны предотвратить любые другие последствия события касания, как показано в следующем фрагменте кода:

```
touched = true;
event.preventDefault();
});
```

- Мы настроили признак состояния для события, когда происходит касание. Когда пользователь убирает палец от экрана, оно должно сбрасываться (т. е. для него должно устанавливаться значение `false`):

```
slider.on('touchend', event => touched = false);
```

- В нашей системе есть анимация. Чтобы ползунок вернулся в исходное положение (к середине дорожки), система должна обновить цикл. Обновление должно происходить только тогда, когда пользователь не касается ползунка; иначе он всегда должен находиться в том месте, где вы держите палец. Обновление положения будет происходить, когда пользователь уже убрал палец, но ползунок еще не вернулся на исходную позицию:

```
const update = function() {
  if(!touched && Math.abs(position) > 0) {
```

11. Следующая часть похожа на код для **ПИД-регуляторов**, поскольку здесь мы также умножаем ошибку на значение пропорционального компонента. Мы масштабируем ошибку с коэффициентом 0,3 и прибавляем/вычитаем еще 0,5, чтобы приблизить ее значение к минимуму (1 %). При каждом обновлении значения ползунков перемещается ближе к центру. Введите следующий фрагмент кода:

```
let error = 0 - position;
let change = (0.3 * error) + (Math.
sign(error) * 0.5);
set_position(position + change);
// console.log(id + ": " + position);
}
};
```

Также в эту часть кода можно записывать данные о положении (это пригодится, если что-то пойдет не так).

12. Чтобы код мог регулярно запускать функцию `update`, мы используем встроенную функцию `setInterval`, которая будет запускать ее повторно с определенным интервалом. Чтобы дисплей был отзывчивым, он должен часто обновляться. Временные параметры в следующем фрагменте кода указаны в миллисекундах:

```
setInterval(update, 50);
```

13. Помимо обновления изображения, код должен вызывать функцию `when_changed`. Он должен делать это только тогда, когда произошло какое-то событие и значение признака состояния изменилось (если оно не меняется, код не должен вызывать функцию). Назовем эту функцию `update_when_changed`. Она проверяет наличие изменений и запускается реже, чем обновляется дисплей. Это позволяет не допустить переполнения обработчика `when_changed` и очереди. Код показан в следующем фрагменте:

```
const update_if_changed = function() {
  if(changed) {
    changed = false;
    when_changed(-position);
  }
};
setInterval(update_if_changed, 200);
}
```

Обратите внимание, что для значения положения мы выполняем операцию смены знака; это значит, что значение верхней части экрана (–100) преобразуется в значение, заставляющее оба двигателя перемещать робота вперед. Не забудьте, что после функции `makeSlider` нужно закрыть скобку.

Теперь наша система готова к запуску.

## Запуск системы

Загрузите весь набор файлов в память робота. Как и прежде, для запуска мы будем использовать команду `python3 manual_drive.py`.

Прежде чем открыть страницу на смартфоне, необходимо просмотреть ее в режиме разработчика в браузере на ПК.

1. В браузере (Chrome или Firefox) подключитесь к <http://myrobot.local:5001> (используйте имя хоста и адрес своего робота).
2. На странице кликните правой кнопкой мыши и выберите пункт **Просмотр кода страницы** в раскрывающемся меню.
3. На панели инструментов разработчика вы найдете кнопки для эмуляции программы со смартфона и событий касания. Включите эмуляцию.
4. Все возможные неполадки лучше всего устранить в браузере на ПК. Проверьте, дает ли перетаскивание ползунков ожидаемые результаты. Затем нажмите **Консоль** и проверьте код JavaScript на правильность.

Распространенной ошибкой в JavaScript и CSS является пропуск знаков препинания, таких как точка с запятой, запятая или скобки. Если селекторы или идентификаторы не будут совпадать (или у них будет отсутствовать точка/решетка), стили не будут применяться, а поиск элементов в JavaScript не даст результатов.

5. Для просмотра страницы вам нужно будет использовать IP-адрес вашего робота, так как устройства большинства производителей не поддерживают адреса .local. IP-адрес робота вы можете узнать на ПК с помощью команды `ping myrobot.local`, как показано в следующем фрагменте кода:

```
$ ping myrobot.local
PING myrobot.local (192.168.1.107): 56 data bytes
64 bytes from 192.168.1.107: icmp_seq=0 ttl=64 time=5.156
ms
```

В примере показано, что IP-адрес робота – 192.168.1.107. Ваш адрес будет отличаться. Запишите его и введите в браузере на смартфоне с портом. Ссылка для моего робота будет выглядеть так: <http://192.168.1.107:5000>.

6. Теперь вы можете управлять движением робота через сенсорный экран смартфона.

Для умелого управления вам нужно будет потренироваться. Для начала я рекомендую попробовать управлять роботом, глядя на него со стороны, а затем, когда освоитесь, вы легко сможете ориентироваться с помощью камеры. Сейчас частота обновления кадров камеры не очень высока, и это может ограничивать движение.

## Устранение неполадок

Наша программа представляет собой достаточно сложную комбинацию кода на Python, HTML, JavaScript и CSS. Если вы столкнулись с какими-либо неполадками, воспользуйтесь следующими советами:

- при возникновении ошибок в Python проверьте каждую строку кода, сравнивая ее с примерами;
- если при запуске на смартфоне на веб-странице что-то не работает, попробуйте запустить эмуляцию в браузере на ПК, выберите вкладку **Консоль** (на панели инструментов разработчика) и повторите операцию. Так вы сможете увидеть ошибки в коде на JavaScript;

- если страница отображается неправильно (например, блоки имеют не тот цвет), проверьте разделы CSS / таблицы стилей и HTML;
- при возникновении ошибки 404 проверьте, соответствует ли URL-адрес в коде HTML маршрутам в Flask/Python;
- если вам кажется, что система не успевает за событиями, увеличьте интервал времени в функции `update_if_change`.

Теперь вы можете управлять роботом удаленно с помощью смартфона, ориентируясь при этом по камере. Вы узнали, как обрабатывать сенсорные события и использовать таблицы стилей и SVG для создания пользовательских виджетов. Также вы увидели, как посредством кода на JavaScript можно «оживлять» виджеты (добавлять анимацию) и отправлять роботу управляющие команды.

В следующем разделе мы сделаем меню более удобным, что в дальнейшем позволит управлять роботом преимущественно со смартфона.

## РЕАЛИЗАЦИЯ ПОЛНОГО УПРАВЛЕНИЯ РОБОТОМ ЧЕРЕЗ СМАРТФОН

Наша цель – полностью перейти на управление роботом через смартфон. После включения мы должны проверить, готов ли робот к работе, а также доступно ли меню на смартфоне. Сейчас меню не совсем удобное. Также на этом этапе вы не сможете запустить поведенческие сценарии, предполагающие отображение данных через Flask. Используя стили (как при разработке сценария ручного управления), мы увеличим кнопки меню и сделаем их более удобными. Меню будет загружать страницу сервера после нажатия на поведенческий сценарий ручного управления или отслеживания объектов.

Сначала исправим проблему с запуском сценариев, использующих Flask.

### Совместимость режимов меню с поведенческими сценариями, использующими Flask

Если вы пробовали запускать поведенческие сценарии, использующие Flask (например, скрипты для камеры) на сервере управления, вы могли заметить странности. Сценарий будет нормально работать на роботе и взаимодействовать с датчиками, но веб-служба не будет работать с портом 5001.

Flask использует подпроцессы для управления режимом отладки, поэтому мы не можем взаимодействовать с ними. Режим отладки нам не нужен, поэтому его необходимо удалить, выполнив следующие шаги.

1. Откройте файл `control_server.py` и перейдите к последним несколько строкам кода.
2. Запустив код из следующего фрагмента, удалите элемент `debug=True` из строки `app.run`:

```
app.run(host="0.0.0.0")
```

Теперь вы можете добавить сценарии ручного управления, отслеживания цветных объектов и лиц на сервер управления, и они запустятся правильно.

## Загрузка видеосервисов

Когда в меню мы выбираем сценарий, использующий видеосервер, после запуска браузер должен подключиться к порту робота 5001, чтобы мы могли увидеть видеовывод.

Сейчас файл `menu.html` принимает ответ от процесса `control_server` и помещает его в окно сообщений. Этот код можно обновить. Начнем с настройки элементов в переменной `mode_config`, которые должны отображать страницу сервера.

На данный момент каждый элемент в переменной `mode_config` содержит только скрипт режима. Внеся изменения в код, мы можем сделать так, чтобы элемент содержал как скрипт, так и информацию о необходимости отображения страницы сервера.

1. Откройте файл `robot.modes.py`.
2. В переменной `mode_config` мы берем простой текст, являющийся названием скрипта (например, `"avoid_behavior.py"`), и заменяем его словарем, допускающим наиболее простой случай (`{"script": "avoid_behavior.py"}`) или более сложный (`{"script": "manual_drive.py", "server": True}`). Такую замену нужно выполнить во всех элементах переменной `mode_config`:

```
mode_config = {
    "avoid_behavior": {"script": "avoid_behavior.
py"},
    "circle_head": {"script": "circle_pan_tilt_
behavior.py"},
    ...
```

3. Затем необходимо обновить сценарии серверного типа в переменной `mode_config`, используя более сложный случай:

```
"color_track": {"script": "color_track_behavior.
py", "server": True},
"face_track": {"script": "face_track_behavior.
py", "server": True},
"manual_drive": {"script": "manual_drive.py",
"server": True}
```

В примере добавлен сценарий ручного управления. Если вы добавили конфигурацию `manual_drive` сюда, то для отображения в меню ее нужно добавить и в переменную `menu_config`.

4. Теперь внесем изменения в метод `run`, чтобы он мог выбирать скрипт из структуры с внесенными изменениями:

```
def run(self, mode_name):
    while self.is_running():
        self.stop()
        script = self.mode_config[mode_name]['script']
        self.current_process = subprocess.
Popen(["python", script])
```

5. Далее проверяем, нужно ли выполнять перенаправление на сервер (если его использует сценарий), а также жив ли текущий процесс. Чтобы сразу было понятно, что значение является признаком состояния (`True/False`), в пример я добавил ключевое слово `is True`:



```
def should_redirect(self, mode_name):
    return self.mode_config[mode_name].get('server')
is True and self.is_running()
```

Файл `robot_modes.py` готов. Файл `control_server.py` отправляет ответы на веб-страницу. Внесем в него изменения таким же образом, как и в `mode_config`, чтобы он возвращал словарь с данными, а не просто строку.

1. Для форматирования ответа мы будем использовать **JavaScript Object Notation (JSON)**. Откройте файл `control_server.py` и добавьте `jsonify` к импортам `Flask`, как показано в следующем фрагменте кода:

```
from flask import Flask, render_template, jsonify
```

2. Теперь меняем метод `run` так, чтобы он создавал словарь `response`:

```
@app.route("/run/<mode_name>", methods=['POST'])
def run(mode_name):
    mode_manager.run(mode_name)
    response = {'message': f'{mode_name} running'}
```

Ответ – это основное сообщение.

3. Если необходимо перенаправление, со следующим ответом отправляем настройку `redirect`:

```
if mode_manager.should_redirect(mode_name):
    response['redirect'] = True
```

4. Нам нужно отправить ответ в формате JSON. В нем удобно передавать данные из Python в JavaScript (особенно словари). Введите код из следующего фрагмента:

```
return jsonify(response)
```

5. Мы отправили сообщение также в команду `stop`, поэтому ее нужно обернуть таким же образом:

```
@app.route("/stop", methods=['POST'])
def stop():
    mode_manager.stop()
    return jsonify({'message': "Stopped"})
```

Теперь сервер управления может отправлять словарь `response` и при необходимости выполнять перенаправление. Другая сторона принимает объект JSON, поэтому для обработки новых ответов необходимо внести изменения в скрипт.

1. Откройте файл `templates/menu.html` и найдите функцию `run`:

```
function run(url) {
```

2. Здесь мы меняем обработку сообщений. Для этого вносим изменения в HTML-элемент сообщения, используя элемент сообщения из ответа:

```
$.post(url, '', response => {
    $('#message').html(response.message);
```

3. Также мы можем проверить, нужно ли выполнять перенаправление. Используем тот же способ, который мы использовали в разделе «Шаблон», для кнопки запуска в сценарии ручного управления, но теперь устанавливаем предел времени:

```

    if(response.redirect) {
        setTimeout(() => window.location.
replace('//' + window.location.hostname + ":5001"),
3000);
    }
})

```

Благодаря функции `setTimeout` вызов функции происходит через определенный промежуток времени. У нас он равен 3000 мс (или 3 с) – этого достаточно, чтобы камера успела прийти в рабочее состояние.

Теперь, когда вы загрузите код в память робота и запустите его с помощью команды `python3 control_server.py`, вы увидите, что меню стало более функциональным, но выглядит все еще простовато. Если вы нажмете на какой-либо сценарий, связанный с отслеживанием или движением, через 3 с вы будете перенаправлены на соответствующую веб-страницу. Нажав кнопку выхода, вы вернетесь в меню.

Перейдем к стилизации меню.

## Стилизация меню

При разработке сценария ручного управления мы уже использовали таблицу стилей для стилизации кнопки выхода. Обновленное меню также представляет собой набор кнопок. В этом разделе мы расширим таблицу стилей и сделаем меню более удобным для использования на смартфоне.

### Преобразование шаблона меню в набор кнопок

Сейчас нам нужно внести изменения в уже существующую таблицу стилей `static/display.css`. Чтобы сделать шаблон меню максимально эффективным, оптимизируем таблицу стилей.

1. Откройте файл `templates/menu.html`. В него нужно добавить ссылку на таблицу стилей и определение `charset`:

```

<head>
  <script src="https://code.jquery.com/jquery-
3.5.1.min.js"></script>
  <title>My Robot Menu</title>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="/static/
display.css">
</head>

```

2. Шаблон меню использует список элементов меню. Чтобы применить существующий стиль, нужно добавить класс `menu` в список и класс `button` в ссылки:

```

<ul class="menu">
  {% for item in menu %}
    <li>
      <a class="button" href="#" onclick="run('/
run/{% item.mode_name %}')">
        {% item.text %}
      </a>
    </li>
  {% endfor %}
</ul>

```

```

    </li>
    {% endfor %}
    <li><a class="button" href="#" onclick="run('/
stop')">Stop</a></li>

```

3. Теперь откройте файл `static/display.css` и определите стиль для класса `menu`, как показано в следующем фрагменте кода:

```

.menu {
    width: 100%;
    margin-top: 0;
    margin-bottom: 0;
    padding: 0;
}

```

Мы сделали так, чтобы контейнер списка заполнял всю ширину экрана без дополнительных полей (пространства вокруг внешней части элемента) или отступов (пространства между внутренней частью элемента и его дочерними элементами списка).

4. Меню состоит из пунктов списка. По умолчанию они отмечаются точкой (как пункты маркированного списка). Нам нужно удалить эту точку, поэтому для параметра устанавливаем значение `none` (без формы). Здесь мы используем свойства CSS `list-style`. Селектор применяется к элементам списка (`li`), которые являются дочерними элементами объекта класса `.menu`. Код показан в следующем фрагменте:

```

.menu li {
    list-style-type: none;
    list-style-position: initial;
}

```

5. Для удобства меню мы сделаем все кнопки одной ширины – `60vw` (60 % от ширины окна просмотра) будет достаточно. Для центрирования задаем значения внешних полей `auto`. Также добавим нашим кнопкам светло-голубую рамку толщиной 1 пиксель:

```

.menu .button {
    margin-left: auto;
    margin-right: auto;
    width: 60vw;
    border: 1px solid lightblue;
}

```

Загрузите весь каталог в память робота и запустите сервер меню с помощью команды `python3 control_server.py`. Теперь на смартфоне меню выглядит более удобным.

Итак, вы оптимизировали работу сервера управления на смартфоне. Также вы узнали больше о взаимодействии JavaScript, HTML и CSS с Python. Однако сейчас у системы есть недостаток – запустить ее можно только через SSH-терминал. Попробуем это исправить.

## РЕАЛИЗАЦИЯ ЗАПУСКА МЕНЮ ВМЕСТЕ С РІ

Сейчас в систему меню для запуска поведенческих сценариев робота мы можем войти только через терминал SSH – такой вариант подходит для задач отладки, обнаружения и устранения проблем, однако для обычного использования это не совсем удобно.

Самый удобный вариант – включить робота, дождаться, пока загорится светодиодный индикатор, а затем подключиться через браузер на смартфоне с возможностью управлять им.

В этом разделе мы реализуем две полезные функции:

- прежде чем мы подключимся к роботу через браузер, он будет сообщать о готовности к работе (в режиме меню) с помощью светодиодной индикации;
- при включении робота Flask сервер меню будет запускаться автоматически с помощью подсистемы инициализации `systemd`.

Начнем с настройки светодиодов.

### Добавляем светодиоды на сервер меню

Нам не нужно, чтобы в меню загружался весь класс робота, – для индикации готовности робота к работе мы будем использовать светодиоды. Мы импортируем систему светодиодов, которая будет запускаться вместе с сервером и выключаться (или прекращать выполнять текущую задачу) при поступлении первого запроса `/gun`. Выполните следующие шаги.

1. Откройте файл `control_server.py` и импортируйте систему светодиодов:

```
from robot_modes import RobotModes
from leds_led_shim import Leds
```

2. Теперь нужно настроить светодиоды. Мы сделаем так, чтобы один светодиод вспыхивал зеленым светом при запуске:

```
mode_manager = RobotModes()

leds = Leds()
leds.set_one(1, [0, 255, 0])
leds.show()
```

3. При запуске какого-либо сценария мы знаем, что кто-то использовал меню. Сбросить значение светодиодов можно в методе `gun`. Нам нужно сбросить значение только одного светодиода, поэтому устанавливаем значение `None` для `leds`. Обратите внимание, что в следующем примере мы вставляем выделенный жирным шрифтом фрагмент кода в существующую функцию `gun`:

```
def run(mode_name):
    global leds
    if leds:
        leds.clear()
        leds.show()
        leds = None
    ...
```

Для проверки загрузите новый код в память робота и запустите его. При запуске светодиод должен вспыхнуть, а затем, когда вы выберете другой сценарий, погаснуть. При переходе из меню в режим проверки светодиодов все должно работать нормально.

## Автоматический запуск с помощью инструмента **systemd**

Инструмент **systemd** предназначен для автоматического запуска программ в Linux. Он позволяет запускать серверы меню/управления, чтобы робот был сразу готов к работе. В разделе «Дополнительные материалы» вы можете найти больше информации об использовании **systemd** на Raspberry Pi.

Регистрация веб-службы происходит путем создания файла модуля и его копирования в нужную папку на Pi. Выполните следующие шаги.

1. Создайте файл `menu_server.service`. Начните с описания, а затем сообщите **systemd**, что служба должна запускаться после подключения к Raspberry Pi:

```
[Unit]
Description=Robot Menu Web Service
After=network.target
```

2. Далее сообщаем **systemd**, что запуск должен происходить, когда Pi готов ко входу пользователей в систему:

```
[Install]
WantedBy=multi-user.target
```

3. В разделе `Service`, показанном в следующем примере, происходит настройка запуска нашего кода:

```
[Service]
```

4. Рабочий каталог – это место, куда вы скопировали файлы робота, например `/home/pi`. Здесь мы устанавливаем имя пользователя `pi`, которое использовали до этого. Рабочий каталог показывает, как код находит другие компоненты. Взгляните на следующий пример:

```
WorkingDirectory=/home/pi
User=pi
```

5. Оператор `ExecStart` передает **systemd** команду для запуска веб-службы. Он не предполагает маршрут (в отличие от оболочки), поэтому мы добавляем к команде `python3` префикс `/usr/bin/env`:

```
ExecStart=/usr/bin/env python3 control_server.py
```

6. Теперь нужно выполнить настройку на Raspberry Pi. Загрузите файл в домашний каталог Pi.

7. Чтобы скопировать файл в конфигурацию системы, используйте `sudo`. Введите код из следующего примера в терминале SSH на Pi. Обратите внимание, что если вы пропустите команду `sudo`, то возникнет ошибка доступа:

```
$ sudo cp menu_server.service /etc/systemd/system/
```

8. Теперь **systemd** должен загрузить нашу конфигурацию, а затем включить службу. Сообщаем ему это с помощью следующего фрагмента кода:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable menu_server
```

- После запуска вы увидите подтверждающее сообщение:

```
Created symlink /etc/systemd/system/multi-user.target.
wants/menu_server.service → /etc/systemd/system/menu_
server.service.
```

- Теперь запускаем службу с помощью следующей команды:

```
$ sudo systemctl start menu_server
```

Если запуск сервера прошел успешно, вы увидите, как вспыхнет зеленый светодиод, сообщающий что система готова к работе. Теперь вы можете подключиться через браузер и начать управлять роботом.

Проверим, все ли в порядке.

## Устранение неполадок

На этом этапе могли возникнуть неполадки. Для их устранения воспользуйтесь следующими советами.

- При запуске/включении сервера меню с помощью `systemd` может возникнуть проблема. Если в файле `menu_server.service` есть ошибка, вы увидите сообщение `Unit menu_server.service is not loaded properly: Invalid argument`. Проверьте его содержимое, скопируйте новую версию файла в память робота и выполните установку с помощью команд `sudo`.
- Чтобы получить больше информации о сервере, введите следующую команду:

```
$ systemctl status menu_server
```

Рi ответит вам подобным сообщением:

```
• menu_server.service - Robot Menu Web Service Loaded: loaded (/etc/
systemd/system/menu_server.
```

```
service; enabled; vendor preset: enabled)
```

```
Active: active (running) since Wed 2020-10-21 23:41:55
BST; 2s ago
```

```
Main PID: 1187 (python3)
```

```
Tasks: 1 (limit: 860)
```

```
Memory: 10.0M
```

```
CGroup: /system.slice/menu_server.service
```

```
└─1187 python3 control_server.py
```

```
Oct 21 23:41:55 myrobot systemd[1]: Started Robot Menu
Web Service.
```

```
Oct 21 23:41:56 myrobot env[1187]: * Serving Flask app
"control_server" (lazy loading)
```

```
Oct 21 23:41:56 myrobot env[1187]: * Environment:
production
```

```
Oct 21 23:41:56 myrobot env[1187]: WARNING: This is
a development server. Do not use it in a production
deployment.
```

3. Команда `systemctl` показывает недавнюю активность, но вам может понадобиться отслеживать вывод сценария по мере его выполнения. Для этого можно использовать команду `journalctl`. Чтобы определить созданную нами службу, используйте префикс `-u`, а чтобы отслеживать журнал – префикс `-f`:

```
$ journalctl -u menu_server -f
```

Теперь вы можете просматривать активность серверов во время их работы. Возможно, это не очень удобно для отладки, зато удобно для запуска служб. Чтобы выйти из режима просмотра журнала воспользуйтесь комбинацией `Ctrl+C`.

Сейчас можно перезагрузить робота, дождаться зеленой вспышки индикатора и начать управление. Кроме того, зеленый свет означает, что голосовой помощник Microsoft тоже готов к работе.

При загрузке нового кода службу необходимо перезапустить. Для этого используйте следующую команду:

```
$ sudo systemctl restart menu_server
```

Поздравляем – теперь управление вашим роботом происходит по-настоящему автономно! Для запуска сценариев не нужен даже компьютер или ноутбук.

## Выводы

В этой главе мы добавили нашему роботу небольшую систему меню для запуска различных режимов через браузер.

Вы узнали, как управлять роботом через смартфон и создавать интересные анимированные виджеты с помощью SVG и JavaScript.

Теперь роботом можно управлять вручную. Чтобы привыкнуть к этой системе управления, может потребовать время. Также при ручном управлении коррекция отклонения представляется более сложной задачей, чем когда ПИД-системы делают это самостоятельно, поскольку двигатели ведут себя немного по-другому. Несмотря на сложности, вы получите ценные навыки управления роботом через смартфон. Чтобы увидеть мир его глазами, при управлении вы можете ориентироваться на данные с камеры (она размещена на передней части).

На основе сервера управления вы разработали сервер меню, а затем реализовали его автоматический запуск вместе с запуском робота. Также вы узнали, как сделать сервер меню совместимым с приложениями, взаимодействующим с видеосервером (например, сценарии ручного управления, отслеживания цветных объектов и лиц). Кнопки меню стали более удобными, и теперь вы можете запускать большинство действий через смартфон.

В завершение вы реализовали светодиодную индикацию готовности меню к работе, а затем настроили его автоматический запуск при включении робота. Если подключить робота и смартфон к одной сети (попробуйте подключить робота к точке доступа смартфона в файле `wpa_supplicant.conf`), то вы сможете использовать робота за пределами вашей мастерской и показывать другим людям, что он умеет. Сейчас управление полностью осуществляется через смартфон!



В следующей главе вы узнаете больше о существующих робототехнических сообществах. Также мы поговорим о дальнейшем развитии ваших навыков в области робототехники и программирования.

## Задание

Созданную систему можно улучшить. Вот несколько советов.

1. В файле `manual_drive.py` функция `handle_instruction` использует набор операторов `if` для обработки инструкции. Если обработчиков больше пяти, то можно расширить список, используя словарь (например, `menu_modes`), что позволит вызывать различные методы обработчиков.
2. Сенсорный интерфейс можно представить в виде двух круглых панелей на экране – левая будет отвечать за управление двигателями, а правая – за положение камеры.
3. Вы можете создать интерфейсы для смартфона для управления параметрами других поведенческих сценариев.
4. Вы можете разнообразить стилизацию меню, добавив круглые кнопки или расстояние между ними (с помощью CSS).
5. Сейчас кнопки в меню содержат только текст. Вы можете подобрать для каждого сценария свою картинку и сделать сетку кнопок.
6. Для удобства выключения Pi в меню можно добавить кнопку **Выключить** (Shutdown), нажатие на которую будет вызывать команду `sudo poweroff`.
7. В систему можно добавить управление роботом с помощью клавиатуры – тогда ее можно будет использовать на компьютере. Конечно, это не так весело, но зато мы получаем удобный запасной вариант.
8. Если хотите добавить в систему серьезное улучшение – можно реализовать управление двигателями на основе подсчета числа импульсов в секунду, используя по одному ПИД-регулятору для каждого колеса. В таком случае система будет сравнивать полученное и ожидаемое количество импульсов. Это сделает движение робота более прямолинейным и, следовательно, им будет проще управлять.

## Дополнительные материалы

Больше информации по темам, рассмотренным в этой главе, вы можете найти в следующих источниках:

- я настоятельно рекомендую изучить документацию по Flask API (<http://flask.pocoo.org/docs/1.0/api/>). Это поможет разобраться в используемых нами функциях Flask, а также рассмотреть другие способы использования этой библиотеки веб-сервера;
- по адресу: <https://www.packtpub.com/product/flask-by-example/9781785286933> вы можете найти книгу от Packt Publishing под названием «*Flask By Example*» автора *Гарета Дуайера* (Gareth Dwyer). В ней подробно рассматривается работа веб-сервера Flask, а также рассказывается, как с его помощью можно создавать более сложные приложения;

- также я рекомендую еще одну книгу от *Packt Publishing* – «*Mastering Flask*» автора Джека Стоуффера (*Jack Stouffer*). Ее вы можете найти по адресу <https://www.packtpub.com/product/mastering-flask/9781784393656>;
- код HTML в этой главе очень простой. Для более подробного изучения рекомендую видеоруководство *Beginning Responsive Web Development with HTML and CSS [eLearning]* авторов Бена Фрейна (*Ben Frain*), Корда Слэттона-Валле (*Cord Slatton-Valle*) и Джошуа Миллера (*Joshua Miller*) от *Packt Publishing* (<https://www.packtpub.com/web-development/beginning-responsive-web-development-html-and-css-elearning-video>);
- во всех приложениях HTML, CSS и JavaScript мы использовали селекторы CSS. На веб-сайте *W3C Schools* в разделе *CSS Selectors* доступны хорошие справочные материалы и руководства ([https://www.w3schools.com/cssref/css\\_selectors.asp](https://www.w3schools.com/cssref/css_selectors.asp)). Также здесь можно найти информацию о других технологиях веб-приложений. В разделе *CSS Units* ([https://www.w3schools.com/cssref/css\\_units.asp](https://www.w3schools.com/cssref/css_units.asp)) можно узнать больше о новых и уже знакомых единицах измерения CSS. Помимо прочего, на этом сайте можно найти много других справочных и учебных материалов по этим веб-технологиям;
- для более подробного ознакомления с используемыми здесь технологиями JavaScript, CSS и HTML можно воспользоваться ресурсом *freeCodeCamp* (<https://www.freecodecamp.org/>), где предоставлены модули для самостоятельного изучения;
- по адресу [https://www.raspberrypi.com/documentation/computers/using\\_linux.html](https://www.raspberrypi.com/documentation/computers/using_linux.html) вы можете найти полезную документацию Raspberry Pi, посвященную пользовательским файлам `systemd`;
- в книге от *Packt Publishing* 2015 года под названием «*Mastering Linux Network Administration*» автора Джея Лакруа (*Jay LaCroix*) есть глава, посвященная инструменту `systemd` (<https://www.packtpub.com/product/mastering-linux-network-administration/9781784399597>);
- полный справочник по службам `systemd` можно найти в руководствах от *freedesktop* по адресу <https://www.freedesktop.org/software/systemd/man/systemd.service.html>.



## Часть 4

# Дальнейшее изучение робототехники

Эта часть расскажет вам, как найти новые интересные проекты и продолжить совершенствовать навыки, полученные в этой книге. Также вы узнаете больше о робототехнических сообществах. Здесь мы подведем итог всех полученных знаний.

Часть включает в себя следующие главы:

- главу 18. *Дальнейшее развитие навыков создания программ для роботов;*
- главу 19. *Планирование следующего робототехнического проекта – подведем итоги.*



# Глава 18

## Дальнейшее развитие навыков программирования робототехнических систем

К этому моменту вы освоили основные навыки сборки роботов и познакомились с некоторыми наиболее интересными приемами программирования, которые применяются в робототехнике. Однако робот, которого вы создали, подходит только для использования в мастерской. Он не годится для соревнований и не может осуществлять поездки на большие расстояния. Этот проект – лишь начало вашего путешествия в мир робототехники. Существует обширное сообщество робототехников и создателей других устройств, которые имеют собственные интересные идеи.

Эта глава расскажет, как продолжить ваше путешествие, найти сообщества робототехников и новые проекты. Также мы поговорим о том, где можно освоить новые навыки. Вы узнаете, какие навыки, не вошедшие в эту книгу, стоит изучить, и почему они так важны для создания роботов.

Разберемся, как стать частью сообщества!

В этой главе мы рассмотрим следующие темы:

- робототехнические интернет-сообщества – форумы и социальные сети;
- мероприятия – соревнования, площадки для совместной работы и встречи;
- навыки, которые стоит освоить, – 3D-печать, пайка, печатные платы и станки с ЧПУ;
- подробнее о компьютерном зрении;
- переход к машинному обучению.

### РОБОТОТЕХНИЧЕСКИЕ ИНТЕРНЕТ-СООБЩЕСТВА – ФОРУМЫ И СОЦИАЛЬНЫЕ СЕТИ

Робототехника – это часть глобального сообщества создателей различных вещей, охватывающего множество областей. Многие радиолюбители и электротехники-любители работают по большей части с робототехническими проектами. Также существуют художники, которые используют такие устройства, как Arduino и Raspberry Pi, чтобы воплотить в жизнь свои творения. Преподавате-

ли используют такие устройства, чтобы познакомить детей с удивительным миром технологий, а также применяют при обучении другим дисциплинам. Частью сообщества являются люди, столкнувшиеся с проблемами при работе над своим проектом и желающие их решить, а также те, у кого есть блестящие, а иногда даже сумасшедшие идеи, которые непременно нужно опробовать.

Члены сообщества активно ведут блоги в Twitter, Instagram и YouTube. Найти такие блоги можно по тегам #raspberrypi (<https://twitter.com/hashtag/RaspberryPi>), #arduino (<https://twitter.com/hashtag/Arduino>) и #makersgonnamake (<https://twitter.com/hashtag/makersgonnamake>). Аккаунт @GuildOfMakers (<https://twitter.com/guildofmakers>) в Twitter – это место, которое объединяет и вдохновляет многих создателей. У меня тоже есть аккаунт в Twitter – @Orionrobots (<https://twitter.com/orionrobots>), где я делюсь своим опытом в создании роботов. Больше интересных блогов можно найти в моих подписках.

#### Примечание редактора перевода

Российские читатели без труда найдут в интернете множество ресурсов по любительской робототехнике на русском языке. Например, посетите такие сайты:

<http://robozone.su/links.html> – коллекция ссылок на ресурсы по робототехнике;

<https://robototehnika.ru/communication/forum/forum4/> – общение на тему самодельных роботов, технические вопросы;

<https://vk.com/robo4u> – клуб любителей робототехники ВКонтакте.

Есть и другая часть сообщества, представители которой больше сосредоточены на таких темах, как использование искусственного интеллекта в робототехнике, обработка видеоданных, распознавание речи, а также на различных реализациях этих технологий. Также эти люди занимаются развитием более сложных областей, таких как нейронные сети, глубокое обучение и генетические алгоритмы. Очень часто такие сообщества формируются при университетах и исследовательских центрах различных компаний. Публикации по обработке речи вы можете найти в Twitter по тегам #mycroft (<https://twitter.com/hashtag/mycroft>) и #voiceassistant (<https://twitter.com/hashtag/voiceassistant>). Для поиска обсуждений и блогов по обработке видеоданных используйте теги #computervision (<https://twitter.com/hashtag/computervision>) и #opencv (<https://twitter.com/hashtag/opencv>). Также вы можете попробовать такие поисковые запросы, как «TensorFlow» и «машинное обучение».

Попробуйте поискать в Twitter аккаунты университетов, например MIT Robotics (<https://twitter.com/MITRobotics>) и CMU Robotics Institute ([https://twitter.com/cmu\\_robotics](https://twitter.com/cmu_robotics)), а также веб-сайт *The Stanford Vision and Learning Lab* (<http://svl.stanford.edu/>). Там вы откроете для себя множество удивительных проектов. Сайты компаний, занимающихся промышленной робототехникой, как правило, менее интересны, но и там можно найти интересные идеи для творчества.

Поставщики компонентов для роботов также имеют большое влияние на сообщество и на своих веб-сайтах часто предлагают отличные проекты.



Многие из них работают на международном уровне. В Великобритании можно отметить Pimoroni (<https://blog.pimoroni.com/>), 4Tronix (<http://4tronix.co.uk/blog/>), Cool Components (<https://coolcomponents.co.uk/>). блоги/новости), а также многие другие. В США есть Adafruit (<https://blog.adafruit.com/>) и Sparkfun (<https://www.sparkfun.com/news>). При поиске таких компаний очень часто поисковая система предлагает ссылки на различные обсуждения и полезные материалы по тем или иным компонентам или проектам.

### Примечание редактора

Давним надежным поставщиком деталей и наборов для любителей технического творчества является интернет-магазин «Амперка» ([www.amperka.ru](http://www.amperka.ru)). В этом магазине можно приобрести многие компоненты, необходимые для сборки робота, описанного в данной книге, а также многие другие детали и наборы.

На веб-сайте сообщества Instructables (<https://www.instructables.com/>) вы можете найти множество проектов, как исключительно робототехнических, так и проектов из смежных областей, которые могут быть полезны робототехникам (сюда входят различные материалы от других пользователей и информация о полезных инструментах). На веб-сайте Hackaday (<https://hackaday.com/>) вы можете найти много отличных историй и руководств.

Помимо перечисленных веб-сайтов, многие сообщества присутствуют на YouTube.

## Каналы на YouTube, с которыми стоит ознакомиться

У меня есть свой канал на YouTube: Orionbots (<https://www.youtube.com/orionrobots>). Здесь я делюсь процессом сборки роботов и экспериментами с датчиками и кодом. Также я размещаю код на GitHub, чтобы люди могли учиться и развивать эти идеи.

Джеймс Брутон (James Bruton) (<https://www.youtube.com/user/jamesbruton>), также известный как XRobots, создает большие и сложные модели роботов с помощью 3D-печати. Его творения легко конкурируют с известными роботами, созданными университетскими командами. В их числе роботизированные костюмы с реальной функциональностью и самобалансирующиеся шагающие роботы.

The Ben Heck Show (<https://www.youtube.com/playlist?list=PLwO8CTSLTkiG2zFzQVbFnbmLY3Akla>) – проект, посвященный не столько робототехнике, сколько созданию каких-либо устройств в целом, в том числе и роботов. Здесь чаще рассказывается об аппаратной, а не о программной стороне, но тем не менее это очень вдохновляет!

Computerphile (<https://www.youtube.com/user/Computerphile>) – это отличный YouTube-канал о программировании, который также выпускает ролики на такие темы, как программирование робототехнических систем, обработка изображений и искусственный интеллект. Здесь вы можете найти множество интервью со значимыми фигурами в области вычислительной техники.

YouTube канал Adam Savage's Tested (<https://www.youtube.com/user/testedcom>) ведет Адам Сэвидж (Adam Savage) из команды Разрушителей легенд (MythBusters).

На этом канале опытные специалисты подробно рассказывают о процессе создания своих устройств, делятся опытом и методами.

У таких производителей, как Makezine (<https://www.youtube.com/user/make-magazine>), Adafruit (<https://www.youtube.com/user/adafruit>), Sparkfun (<https://www.youtube.com/user/sparkfun>) и Pimoroni (<https://www.youtube.com/channel/UCuiDN-TaTdPTGZZzHm0iriGQ>) тоже есть каналы на YouTube (а также веб-сайты). Там вы можете найти информацию о доступных компонентах и различные руководства.

К перечисленным YouTube каналам вы можете обратиться, если захотите посмотреть, как и над чем работают другие робототехники. Эти площадки – отличный способ пообщаться с другими пользователями и задать вопросы.

## Технические вопросы – куда обратиться за помощью

Любые технические вопросы вы можете задать на Stack Exchange в разделах Raspberry Pi (<https://raspberrypi.stackexchange.com>), Electronics (<https://electronics.stackexchange.com/>), Robotics (<https://robotics.stackexchange.com>). По вопросам, касающимся программирования, можно обратиться к сообществу Stack Overflow (<https://stackoverflow.com>). С техническими вопросами можете обратиться к сообществу Quora (<https://hi.quora.com>), работающему в формате вопрос–ответ. Также существуют форумы Raspberry Pi (<https://forums.raspberrypi.com>) и Mycroft (<https://community.mycroft.ai>).

У OpenCV есть форум для технических вопросов, похожий на Stack Overflow. Его вы можете найти по адресу <http://answers.opencv.org/questions/>.

В плане технических вопросов Twitter является площадкой с более открытым форматом. Вы можете задать вопрос с тегом по интересующей теме и найти популярные аккаунты Twitter, посвященные робототехнике.

Задавать вопросы можно под видео на различных YouTube-каналах, но, прежде чем спросить что-либо, посмотрите само видео (там уже может быть ответ).

Поделюсь небольшой хитростью по поиску альтернативных технологий или решений в поисковых системах. Введите название интересующей вас технологии, затем напишите «против» (vs. или versus) и посмотрите, какие варианты замены предлагает поисковик. Это позволит вам взглянуть на решение проблемы с другой стороны.

Общение в интернете – это отличный способ получить ответы на интересные вопросы, но ничто не сравнится со встречей с профессиональными робототехниками в реальной жизни. Где проводятся подобные мероприятия, и как можно поучаствовать в них?

## Мероприятия – соревнования, площадки для совместной работы и встречи

Если вы планируете и дальше заниматься робототехникой, то посещать различные тематические мероприятия нужно обязательно. На них специалисты делятся ценным опытом и знаниями. Также эти события играют важную социальную роль. Крупные мероприятия обычно платные, но некоторые можно посетить и бесплатно.

## Площадки для совместной работы

На них встречаются специалисты из разных областей – робототехники, мастера, создающие изделия ручной работы, деятели искусства, радиолюбители и многие другие. Площадки предоставляют место для экспериментов и необходимые инструменты.

На многих площадках есть **станки с ЧПУ (числовое программное управление – Computer Numerical Control (CNC))**, например 3D-принтеры, станки лазерной резки и фрезерные станки для резки различных материалов и нарезания резьбы. Также обычно там есть рабочие столы со всей необходимой электроникой и ручные инструменты.

В некоторых местах доступны материалы для создания **печатных плат (Printed Circuit Boards – PCB)**. У таких площадок также есть свои сообщества, объединяющие людей, создающих проекты на базе этих устройств. Обычно участники с радостью делятся опытом и знаниями со всеми желающими.

Площадки для совместной работы дают возможность развить практические навыки. У некоторых из них, например у **Cambridge Makerspace** (<https://twitter.com/cammakespace>), есть отдельные робототехнические клубы.

Такие площадки есть как в больших, так и в маленьких городах по всему миру. Также их называют хакспейсы (Hackerspace) и фаблабы (fab lab). Например, на юго-западе Лондона действуют такие площадки, как London Hackspace (<https://london.hackspace.org.uk/>), Richmond Makerlabs (<https://richmondmakerlabs.uk/>) и South London Makerspace (<https://southlondonmakerspace.org/>). В Мумбаи есть площадка Makers Asylum (<https://www.makersasylum.com/>). По адресу <https://makerspaces.make.co> вы можете найти карту площадок. Также их можно найти в Google-картах по запросам «пространство для совместной работы», «хакспейс» или «fab lab».

Площадки можно легко найти в поисковых системах и социальных сетях. Если в вашем регионе их нет, можете найти единомышленников в интернете и организовать свою. При выборе места ориентируйтесь на предполагаемые задачи. Например, проводить паяльные работы можно только в специальных помещениях и с достаточно большим коллективом.

## Maker Faire, Raspberry Jam и Coder Dojo

Maker Faire (<https://makerfaire.com/>) – это фестивали, которые проводятся по всему миру и посвящены созданию различных вещей. Здесь люди собираются для демонстрации своих проектов и совместной работы над ними. Проводятся и фестивали робототехники. Это могут быть однодневные мероприятия или фестивали с кемпингами, длящиеся несколько дней, например EmfCamp (<https://www.emfcamp.org/>) в Великобритании. Такие события – это отличный повод начать осваивать новые навыки. Они дают возможность рассказать о своих проектах, а также посмотреть на проекты других создателей.

Raspberry Jam (<https://www.raspberrypi.org/jam/>) и Coder Dojo (<https://coderdojo.com>) – это групповые события, где люди вместе совершенствуют свои навыки (в основном по части программирования, но, бывает, и в технической области). Coder Dojo – это рабочая группа сообщества программистов. Raspberry Jam – это нечто похожее, но с упором на работу с Raspberry Pi. Raspberry Jam прово-

дятся как для взрослых, так и для детей (узнайте, какие проводятся в вашем городе). Также можно стать наставником в Dojo или Jam – это отличный способ завести новые полезные контакты.

Ежегодно проводятся тематические вечеринки от сообщества Raspberry Pi – веселые мероприятия, нацеленные скорее на встречи, чем на совместную работу.

Все эти группы, как правило, имеют странички в Twitter, на которые вы можете подписаться и получать вдохновение!

## Соревнования

За пределами академических кругов соревнования роботов – это все еще редкое явление. В США существует инженерная инициатива FIRST (<https://www.firstinspires.org/robotics/frc>), которая поддерживает создание клубов робототехники в школах и колледжах, а также их участие в соревнованиях с несколькими спонтанно организованными командами FIRST за пределами страны. Соревнования проводятся как для автономных роботов, так и для роботов с ручным управлением. В большинстве стран существуют организации **STEM (Science Technology Engineering and Mathematics – наука, технология, инжиниринг и математика)**; например, такая есть в Великобритании (<https://www.stem.org.uk/>). Иногда они проводят робототехнические соревнования, о которых вы сможете узнать на их сайтах и в новостях. Перед посещением обязательно уточните, доступны ли они для широкой аудитории.

Ежегодно в Великобритании проводится соревнование PiWars (<https://piwars.org/>), которое включает в себя множество испытаний для автономных и управляемых вручную роботов, созданных учениками школы вычислительной техники Кембриджского университета (Cambridge University School of computing). У соревнований есть сплоченное сообщество. Вы можете посетить PiWars в качестве участника или зрителя. По тегу #piwars (<https://twitter.com/hashtag/PiWars>) в Twitter вы можете найти активные обсуждения, а также информацию о встречах, которые проводят робототехники для создания и тестирования роботов перед соревнованиями.

Также в Великобритании проводятся соревнования Micromouse (<https://www.micromouseonline.com>). В основном в них участвуют роботы, созданные для прохождения лабиринтов, но встречаются и другие. Некоторые из них выставлены на продажу.

Robotex International (<http://robotex.international>) – это робототехническая выставка, проходящая в Эстонии. Она длится несколько дней и включает в себя различные шоу, рассказы о робототехнике и соревнования, победители которых получают серьезные призы. Выставка открыта для робототехников, работающих с электроникой и Raspberry Pi, а также с Lego и другими материалами.

Для участия в соревнованиях вам, скорее всего, понадобится перевозить своих роботов. Для безопасной транспортировки запаситесь большими коробками, воздушно-пузырчатой пленкой и упаковочным пенопластом.

При транспортировке я рекомендую вынуть батарейки (чтобы избежать короткого замыкания) и упаковать их в полиэтиленовый пакет, изолировав от соприкосновения с металлическими компонентами. Начните изучать размещение проводов в конструкции робота – тема выходит за рамки этой книги, но это позволит сделать ваши проекты более надежными.

Всегда берите с собой набор для ремонта в полевых условиях с макетной платой, проводами, запасными батарейками, зарядным устройством, несколькими типами отверток, запасными компонентами для преобразователей логического уровня, «липучкой», комплектом стоек и по возможности мультиметром. По прибытии на место работы часто нуждаются в настройке или небольшом ремонте.

В этом разделе вы узнали о местах, где можно пообщаться с другими робототехниками, о рабочих площадках с необходимыми инструментами, а также о соревнованиях. В следующем разделе мы поговорим о том, какие дополнительные навыки могут пригодиться при создании роботов.

## НАВЫКИ, КОТОРЫЕ СТОИТ ОСВОИТЬ, – 3D-ПЕЧАТЬ, ПАЙКА, ПЕЧАТНЫЕ ПЛАТЫ И СТАНКИ С ЧПУ

По мере углубления в робототехнику вам захочется создавать все более сложные или индивидуально настроенные системы.

Чтобы создать робота для соревнований, необходимо освоить новые навыки работы с аппаратными компонентами.

### Навыки проектирования

В этой книге мы использовали блок-схемы и простые рисунки. Однако, когда вы занимаетесь робототехникой на более продвинутом уровне, часто возникает необходимость изготавливать компоненты или проверять, подходят ли приобретенные компоненты для вашей системы. С помощью **систем автоматизированного проектирования (САПР) (CAD – Computer-Aided Design)** вы можете спроектировать корпус, шасси, крепления датчиков, опоры, различные типы колес и другие компоненты.

#### 2D-проектирование для иллюстраций и схем

Для 2D-проектирования и создания иллюстраций я рекомендую графический редактор Inkscape (<https://inkscape.org/>). Он предназначен скорее для художников и не совсем похож на САПР, но для создания логотипов и рисунков этот инструмент отлично подойдет. Разобраться в нем достаточно сложно, поэтому я рекомендую ознакомиться с книгой от *Packt Publishing* под названием «*Inkscape Beginner's Guide*» автора *Бетани Хиитола (Bethany Hiitola)*.

**Draw.io** (<https://app.diagrams.net>) – это удобный инструмент для создания схем, подобных тем, которые встречались в этой книге. В Inkscape вы можете создавать фигуры, которые затем можно использовать в Draw.io. Редактор Inkscape позволяет проводить больше манипуляций с фигурами, а в Draw.io удобнее проверять размещение компонентов и комбинировать их.

#### 3D-САПР

Я рекомендую тщательно изучить 3D-САПР системы, такие как FreeCAD (<https://www.freecadweb.org>) и Fusion 360 (<https://www.autodesk.com/campes/fusion-360-for-hobbyists>). FreeCAD – это бесплатная система с открытым исходным кодом. У Fusion 360 также есть бесплатная САПР для начинающих.

С помощью 3D-САПР можно проектировать части робота и комбинировать компоненты, а также проверять их размещение. В дальнейшем вы можете использовать эти проекты для работы с ручным инструментом или 3D-печати.

Чтобы разобраться в этих системах, потребуется время, поэтому я рекомендую обратиться к учебным пособиям и видеороликам на YouTube. Можете начать с канала Maker's Muse (<https://www.youtube.com/channel/UCxQbYGpbdrh-b2ND-Aflybg>).

В сообществе Thingiverse (<https://www.thingiverse.com/>) вы найдете 3D-проекты для печати и изготовления компонентов. Из этих проектов вы можете черпать вдохновение, а затем создавать на их основе собственные. Вместо того чтобы рисовать монтажное основание с нуля, попробуйте импортировать его в FreeCAD и добавить необходимые отверстия, стойки или контакты. В дальнейшем это может значительно сэкономить вам время. Также на веб-сайте вы можете найти множество советов по печати компонентов. Если на Thingiverse не нашлось того, что вы ищете, попробуйте поискать информацию на других сайтах, таких как Pinshape (<https://pinshape.com/>) и GrabCad (<https://grabcad.com/>).

Созданные в САПР проекты вы можете отправить на изготовление (или изготовить их самостоятельно, изучив необходимые технологии).

## Навыки обработки компонентов и сборки

В качестве общей рекомендации я могу отметить материалы курса How To Make Almost Anything (<http://fab.cba.mit.edu/classes/863.14/>) от Массачусетского технологического института (MIT – Massachusetts Institute of Technology). Они содержат много полезной информации о том, как соединять компоненты между собой. Материал кажется очень простым, но он содержит полезные ссылки и обновляется каждый год. Как уже упоминалось в разделе «Робототехнические интернет-сообщества – форумы и социальные сети», каналы на YouTube и других видеохостингах изобилуют практическими примерами и руководствами по созданию каких-либо устройств.

## Навыки по работе со станками

С помощью фрезерования на станке с ЧПУ, лазерной резки и 3D-печати вы можете создавать цельные компоненты отличного качества, но для этого вам необходимо освоить соответствующие навыки. Лазерная резка позволяет изготавливать плоские детали, которые затем, при должной изобретательности, можно собрать (так, как собирают мебель) в сложные трехмерные объекты.

На YouTube-канале NYC CNC (<https://www.youtube.com/user/saunixcomp>) вы можете найти множество советов по работе на станках с ЧПУ. Также я рекомендую книгу «*Guerrilla guide to CNC machining, mold making, and resin casting*» автора Михала Залевски (Michal Zalewski).

Я не советую приобретать собственные станки для обработки компонентов. Лучше всего поискать поблизости площадки для совместной работы (о которых мы говорили ранее) – как правило, там есть подходящее оборудование. Зачастую 3D-принтеры и простые материалы для изготовления моделей есть в библиотеках. Это позволит вам сэкономить средства. К тому же на площадках вы будете работать в окружении опытных людей, которые могут помочь.



Если нужно просто изготовить компоненты на 3D-принтере или с помощью лазерной резки, в интернете обязательно найдутся люди, которые сделают это для вас. Например, можно обратиться в такие компании, как Ponoko (<https://www.ponoko.com/>), RazorLAB (<https://razorlab.online/>), 3DIng (<https://www.3ding.in/>), Protolabs (<https://www.protolabs.co.uk/>), Shapeways (<https://www.shapeways.com/>) и 3D Hubs (<https://www.3dhubs.com/>). Найти услуги 3D-печати и лазерной резки в вашем регионе через поисковую систему совсем несложно, но я все же советую посетить площадки и посмотреть, на что способны эти станки. Если вы приобретете неподходящий станок или разработаете неподходящий проект, это приведет к серьезным затратам.

3D-принтеры, станки лазерной резки и станки с ЧПУ требуют регулярно технического обслуживания – например, необходимо часто выравнивать платформу для 3D-печати и регулировать положение патрона на станке с ЧПУ. Также для станков нужны расходные материалы – материал для изготовления компонентов (пластиковая нить, древесина для фрезерных станков или станков лазерной резки), сменные компоненты, а также адгезивы (клеящие вещества, например для стола принтера). Если вы не изготавливаете компоненты роботов регулярно, оптимальным решением будет посещать площадки для работы или пользоваться услугами в интернете.

Станки позволяют соблюдать высокую точность при изготовлении компонентов, но, чтобы окончательно обработать или подправить их, вам понадобится ручной инструмент. В некоторых случаях лучше изготавливать компоненты вручную (например, для уникальных проектов).

### Навыки по работе с ручным инструментом

Навыки по работе с деревом и созданию чего-либо руками всегда пригодятся. Попрактиковавшись на площадках, вы поймете, как сочетать работу на станках и вручную. Также вы разберетесь, как выбрать подходящую древесину, поскольку некоторые виды могут быть слишком мягкими, тяжелыми или иметь неподходящую форму. Деревянные компоненты можно изготавливать как вручную, так и на станках с ЧПУ.

Трехмерные компоненты можно изготавливать другими способами – например, из пластиковых (стирольных) листов или используя формы и литье. Стирольные листы – это недорогой и пластичный материал, который можно легко разрезать вручную, а затем собрать из него компонент по распечатанному шаблону.

Из дерева можно изготавливать формы и импровизированные шасси для роботов. С помощью форм вы можете изготавливать несколько одинаковых компонентов или использовать материалы для высококачественных компонентов. Отливать компоненты немного сложнее, и особенно неприятно, когда в них появляются пузырьки. На эту тему есть хорошие обучающие материалы, например статья по адресу <https://medium.com/jaycon-systems/the-complete-guide-to-diy-molding-resin-casting-4921301873ad>, видеоролик <https://youtu.be/BwLGK-uuQ90> и книга «*Guerrilla guide to CNC machining, mold making, and resin casting*», которая упоминалась в разделе «Навыки по работе со станками».

Для создания крупных роботов следует освоить навыки изготовления компонентов из металла, т. е. научиться вырезать, придавать форму и сваривать их.



Материалы из углеродного волокна или кевлара подходят для создания больших боевых роботов, а также для роботов, которые предназначены для работы с тяжелыми материалами.

Чтобы освоить навыки, перечисленные в этом разделе, обратитесь к сообществам Instructables (<https://www.instructables.com/>) и Hackaday (<https://hackaday.com>). Там есть практические инструкции и учебные пособия по созданию различных вещей. Вы можете постоянно следить за какими-либо проектами или же просто бегло ознакомиться с содержанием и подчерпнуть нужные знания. Помимо робототехнических проектов, попробуйте поискать информацию о разных техниках моделирования (часто они схожи), о сборке пластиковых, деревянных или металлических изделий. Тематические уроки часто проводятся на различных площадках для совместной работы.

Теперь вы знаете, как можно изготовить структурные и механические компоненты, поэтому пришло время поговорить об электронике.

## Навыки работы с электроникой

Следующее, что необходимо сделать, – это расширить навыки по работе с электроникой. В нашем роботе мы использовали HAT-платы и модули для Raspberry Pi. Для первого проекта это нормально, но все же множество компонентов, которые требуют дополнительного пространства, и электропроводка, которую легко повредить, делают нашего робота далеким от идеала. Позже вы заметите, что робот сильно перегружен проводами.

### Принципы электроники

Чем больше вы узнаете об электронных компонентах и общих функциях электронных схем, тем лучше будете понимать своего робота. Благодаря этому вы сможете расширить его функционал, найти способы уменьшить размер и устранить возникающие проблемы.

Знание силовой электроники поможет разобраться, как работают контроллер двигателя и схемы заряда аккумуляторов. Изучая цифровую электронику, вы научитесь подключать к роботу различные логические устройства и датчики, а также объединять их. Углубившись в аналоговую электронику, вы откроете новые типы датчиков и приводов, а также узнаете, с помощью каких инструментов можно обнаруживать неполадки в электронике.

Прежде всего необходимо научиться создавать и понимать принципиальные схемы для основных компонентов. На эту тему есть множество онлайн-курсов и YouTube каналов, где об электронике рассказывается поэтапно. Также я рекомендую книгу «*Make: Electronics*» от автора Чарльза Платта (*Charles Platt*), где на первый план выдвигается именно практика.

В блоге EEVBlog (<https://www.eevblog.com/episodes/>) представлено меньше пошаговых уроков, поскольку авторы делают упор на вопросы электронной инженерии.

### Совершенствование навыков пайки

В этой книге вы уже занимались пайкой, но лишь самую малость. Эту тему стоит изучить основательно. Пайка – очень важный навык для создания различных электронных устройств.

Для начала я рекомендую руководство по пайке от Raspberry Pi (<https://www.raspberrypi.org/blog/getting-started-soldering/>), руководство *The Adafruit Guide To Excellent Soldering* (<https://learn.adafruit.com/adafruit-guide-excellent-soldering>), а также видеоматериалы *EEVBlog Soldering* (<https://www.youtube.com/watch?v=J5Sb21qbpEQ>).

Лучше всего начать с работы на совместных площадках, где можно выполнять простые проекты с пайкой, общаясь с более опытными людьми. Научитесь припаивать разъемы к платам модулей. Освоив навык пайки, вы сможете самостоятельно создавать HAT-платы для Raspberry Pi и другие типы плат.

Начните с припаивания простых разъемов и освоите **монтаж компонентов в отверстия**. Научившись делать это, вы станете паять гораздо увереннее.

Освоившись, можете перейти к работе с наборами по обучению технологии поверхностного монтажа. У компонентов, монтирующихся на поверхности, нет «ножек», которые должны проходить через отверстие платы. Они представляют собой простые металлические компоненты, которые предназначены для припаивания к медным контактным площадкам платы. Такие компоненты занимают гораздо меньше места, что позволяет создавать небольшие конструкции. Однако, с другой стороны, работа с ними – это достаточно кропотливое занятие, для которого требуются профессиональные инструменты. Простые компоненты для поверхностного монтажа, такие как светодиоды, резисторы и конденсаторы, можно паять вручную. Ознакомьтесь с видеороликом *Surface Mount* от *EEVBlog* (<https://www.youtube.com/watch?v=b9FC9fAlfQE>).

Устройства с десятками паяных соединений поверхностного монтажа могут не работать из-за некачественной пайки. Кроме того, для работы с ними требуется специальный фен и паяльная паста. Освоив описанные выше аспекты, вы сможете создавать схемы самостоятельно. Также можете обратиться к тем, кто предоставляет услуги **PCBA (Printed Circuit Board and Assembly – сборка печатных плат)**.

## Создание собственных схем

Когда вы разберетесь в электронике и научитесь паять, вы начнете создавать все больше схем, а на их основе – печатные платы профессионального уровня, что позволит экономить пространство и упростить подключение. Макетные платы отлично подходят для обучения и экспериментов, но для создания робота, предназначенного для участия в соревнованиях, они являются слишком громоздкими и неаккуратными. Кроме того, при работе с последовательным соединением контактов можно легко допустить ошибку или повредить схему.

Для первой надежной платы я рекомендую выбрать универсальную отладочную печатную плату и припаять к ней компоненты. В сравнении с беспаячными макетными платами такой вариант позволяет сэкономить гораздо больше места. Однако такие платы все же остаются довольно громоздкими и неаккуратными. Также для подобных плат не получится использовать компоненты, предназначенные для поверхностного монтажа, или компоненты с «ножками» разных размеров, расположенными в разных местах.

Чтобы вывести схемы на новый уровень, вы можете научиться создавать печатные платы. Они значительно надежнее, экономят больше места, а также позволяют использовать миниатюрные компоненты. В дальнейшем вы може-

те спроектировать печатные платы, которые предназначены в том числе для размещения конструктивных компонентов.

При работе с макетными платами вы можете использовать Fritzing (<http://fritzing.org/home/>), но я не рекомендую его для работы со схемами или печатными платами. Для них лучше всего воспользоваться таким ПО, как KiCad (<https://kicad.org>). Ознакомьтесь с курсом видеоматериалов от Packt Publishing под названием *KiCad Like a Pro* автора Пумера Далмариса (Peter Dalmaris).

Для создания печатных плат вы можете использовать оборудование в местных клубах радиолюбителей или воспользоваться услугами мастерских, которые изготовят для вас плату с тонкими дорожками, подписями и цветными паляльными масками (новых терминов будет встречаться еще очень много). Для самодельной печатной платы вы можете выбрать расположение элементов (что позволяет избежать проволочных перемычек), использовать компоненты для поверхностного монтажа, а также добавить любые подписи к контактам – готовая плата будет выглядеть профессионально. Некоторые даже используют эту технологию для изготовления других деталей, включая конструктивные компоненты и лицевые панели.

## ПОДРОБНЕЕ О КОМПЬЮТЕРНОМ ЗРЕНИИ

В главе 13 мы начали говорить о компьютерном зрении. Используя OpenCV, мы научили робота отслеживать цветные объекты, но это лишь малая часть возможностей, которые предлагает данная технология.

### Книги

Для дальнейшего изучения OpenCV я рекомендую книгу от Packt Publishing под названием «*OpenCV with Python By Example*» автора Прадика Джоши (Prateek Joshi). В ней автор рассказывает, как использовать технологии компьютерного зрения для создания инструментов дополненной реальности, а также для идентификации и отслеживания объектов. В книге проводятся различные преобразования и проверки изображений, которые сопровождаются скриншотами. Читать эту книгу очень увлекательно, поскольку она содержит много кода, который можно использовать на практике.

От обычного компьютерного зрения вы можете перейти к трехмерному. Для этого понадобится сенсорный контроллер Kinect для Xbox 360. Хотя Microsoft сняла их производство, на eBay доступно множество вариантов. Вместо Kinect можно использовать более современное устройство Azure Connect, но на момент написания этой книги оно стоит в 20 раз дороже! Также сенсорный контроллер Kinect подходит для работы с Raspberry Pi. Он оснащен системой 3D-датчиков, что как нельзя лучше подходит для робототехнических проектов. В книге от издательства Packt Publishing под названием «*Raspberry Pi Robotic Project*» доктора Ричарда Гриммета (Dr. Richard Grimmett) есть раздел о подключении этого устройства к Raspberry Pi.

### Онлайн-курсы

На веб-сайте PyImageSearch (<https://www.pyimagesearch.com/>) можно найти одни из лучших ресурсов для изучения OpenCV и экспериментов с компьютерным зрением.

По адресу <https://www.packtpub.com/in/application-development/learn-computer-vision-python-and-opencv-video> вы можете найти руководство от *Packt Publishing* под названием *Learn Computer Vision with Python and OpenCV* автора *Катиравана Намараджана (Kathiravan Natarajan)*. В нем подробно рассказывается об отслеживании цветных объектов, обнаружении признаков и анализе видеоданных. Для всех экспериментов с преобразованием изображений автор использует крайне интересный инструмент Jupyter.

На веб-сайте TensorFlow Tutorials по адресу <https://www.tensorflow.org/tutorials/> представлено множество учебных пособий по использованию TensorFlow (среда машинного обучения) в компьютерном зрении. Помните, что для того, чтобы научить систему машинного обучения визуальному распознаванию, уйдет много времени.

У *Packt Publishing* есть интересный видеокурс под названием *Advanced Computer Vision Projects* автора *Мэтью Ревера (Matthew Rever)* (<https://www.packtpub.com/big-data-and-business-intelligence/advanced-computer-vision-projects-video>). В нем рассказывается о различных проектах на основе компьютерного зрения, а также о реализации системы машинного обучения на базе TensorFlow, которая будет анализировать положение тела человека, основываясь на данных камеры.

## Социальные сети

В разделе «Робототехнические интернет-сообщества – форумы и социальные сети» я уже говорил, что по тегам #computervision и #opencv в Twitter можно найти информацию по теме, задать вопрос или поделиться своим опытом.

На YouTube-канале Computerphile есть плейлист, посвященный компьютерному зрению ([https://www.youtube.com/watch?v=C\\_zFhWdM4ic&list=PLzH6n4zXucKoRdljSlM2k35BufTYXNNNeF](https://www.youtube.com/watch?v=C_zFhWdM4ic&list=PLzH6n4zXucKoRdljSlM2k35BufTYXNNNeF)). В видеороликах рассказывается об основных понятиях и теоретических аспектах некоторых алгоритмов обработки видеоданных, но авторы не делают упор на практическую часть.

Итак, вы узнали где можно найти информацию по теме компьютерного зрения, и это подводит нас к еще одной более сложной теме из мира робототехники – машинному обучению.

## ПЕРЕХОД К МАШИННОМУ ОБУЧЕНИЮ

Наиболее «интеллектуальными» считаются роботы, использующие машинное обучение. В этой книге не представлен код для его реализации, и вместо этого используются хорошо известные алгоритмы. Здесь мы использовали **ПИД-регуляторы (пропорционально-интегрально-дифференциальный – Proportional Integral Derivative (PID))** – системы, которые предназначены для коррекции ошибок и выработки управляющего сигнала, но не относятся к технологии машинного обучения. Однако с этим может быть связана оптимизация значений ПИД-регулятора. Также мы использовали каскады Хаара для сценариев обнаружения лиц, и это тоже не является машинным обучением. Однако авторы OpenCV, вероятно, использовали такие системы для создания этих каскадов.

Системы машинного обучения отлично справляются с оптимизацией задач, а также с обнаружением и сопоставлением шаблонов, но все же на их основе нельзя создать полностью сформированный «интеллектуальный» поведенческий сценарий.

Принцип работы большинства систем машинного обучения состоит в том, что системе дается набор начальных примеров и информация о том, какие примеры подходят, а какие нет. Система должна определять или изучать правила, основываясь на совпадениях или несовпадениях. Эти правила могут являться оценками функции пригодности, основанными на правилах обучения, что позволяет максимизировать такую оценку. Это называется обучением системы.

В случае системы ПИД-регуляторов вы можете основывать оценку пригодности на установлении уставки за наименьшее количество шагов с небольшим выбросом или вообще без него, основываясь на обучающих значениях из данных, таких как изменчивость системы, время ответа и скорость.

Для ознакомления с основными аспектами машинного обучения я снова порекомендую YouTube-канал Computerphile и плейлист AI Video (<https://www.youtube.com/watch?v=tLS5Y2vm02c&list=PLzH6n4zXuckquVnQ0KlMDxyT5YE-sA8Ps>). Эта серия видеороликов направлена больше на теорию, чем на практику.

Большую роль в машинном обучении играют данные и статистика, но методы, которые вы изучали в этой книге, могут использоваться для данных от датчиков, что делает их более актуальными для робототехники. Существует множество примеров использования TensorFlow для создания систем распознавания объектов. Решения с генетическими алгоритмами успешно применяются для шагающих роботов, а также для поиска быстрых способов навигации в пространстве.

## Операционная система для роботов

Многие робототехники работают с **операционной системой для роботов (ROS – Robot Operating System)**; подробнее о ней можно узнать по адресу <http://www.ros.org>. Ее используют для создания общих абстракций на разных языках программирования между аппаратным обеспечением робота и поведенческими сценариями. Так создаются общие уровни многократно используемого кода. Системы искусственного интеллекта, построенные на базе этой технологии, можно комбинировать с низкоуровневыми системами. Уровни кода робота / поведенческих сценариев, разработанные в этой книге, также могут использоваться многократно, но в сравнении с ROS они очень просты.

Книга от издательства *Packt Publishing* под названием «*ROS Programming: Building Powerful Robots*» автора *Анила Махтани (Anil Mahtani)*, посвящена связи системы искусственного интеллекта TensorFlow с робототехническими системами на основе ROS.

Для начала вы можете ознакомиться с еще одной книгой от *Packt Publishing* под названием «*Learning Robotics with Python*» автора *Лентина Джозефа (Lentin Joseph)*, где рассказывается о создании робота с искусственным интеллектом, использующим технологию лидар, на основе ROS и Python.

## Выводы

В этой главе вы узнали больше о сообществах робототехников и их проектах, а также о том, как стать частью этих сообществ. Обмен знаниями с другими людьми позволит значительно ускорить ваш прогресс.

Также вы узнали о соревнованиях роботов, о том, где можно попросить совета и как найти материалы для развития навыков, необходимых в робототехнике. Направление задано, и теперь вдохновение должно помочь вам совершенствовать навыки все больше.

В следующей главе мы подведем итог всему, что вы узнали в этой книге, а также поговорим о планах на ваш следующий проект.

## Дополнительные материалы

Предлагаю вам ознакомиться с некоторыми практическими книгами по робототехнике, которые мне нравятся:

- «*Python Robotics Projects*» профессора Дивакара Байиша (Diwakar Vaish) от Packt Publishing. В этой книге представлено множество проектов по созданию роботов на основе Raspberry Pi и Python;
- «*Robot Building for Beginners*» автора Дэвида Кука (David Cook) от Apress. В этой книге рассказывается о создании с нуля робота из ланчбокса (робот sandwich). Упор здесь делается на сборку устройств и электронику, но все же проект достаточно интересный;
- «*Learning Raspberry Pi*» автора Самарта Шаха (Samarth Shah) от Packt Publishing. Эта книга позволяет узнать больше о возможностях Raspberry Pi, а также помогает обрести вдохновение для совершенствования своих проектов;
- «*Robot Builder's Bonanza*» (5th Edition) автора Гордона МакКомба (Gordon McComb) от McGraw-Hill Education TAB. Это книга влиятельного автора, и в ней подробно описывается процесс создания робота. Это лучшая книга для того, чтобы перейти от робототехнических наборов к созданию более крупных и механически сложных роботов.



# Глава 19

## Планирование следующего робототехнического проекта – ПОДВОДИМ ИТОГИ

В этой книге вы узнали, как планировать, проектировать и строить роботов, а также разрабатывать для них различные программы. Мы рассмотрели основные темы на примерах, попрактиковались в их реализации и поговорили о том, как можно улучшить созданные системы. В этой главе я предлагаю вам подумать о следующем проекте и ответить на вопросы: как создать план нового проекта? какие навыки могут понадобиться для исследований и экспериментов? какого робота вы хотите построить?

В этой главе мы рассмотрим следующие темы:

- дизайн нового робота – как он будет выглядеть;
- создание блок-схемы – определяем необходимые входы/выходы и компоненты;
- выбор компонентов – какие аспекты нужно учесть;
- планирование кода – определяем программные уровни и компоненты, выбираем поведенческие сценарии;
- рассказываем миру о проекте – как поделиться своими планами с заинтересованными людьми?

### ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

В этой главе мы создадим несколько блок-схем. Для этого понадобится следующее:

- ручка/карандаш;
- бумага – в идеале блокнот для набросков (или просто бумага в клетку), но подойдет даже оборотная сторона конверта;
- компьютер с доступом к интернету (чтобы пользоваться <https://app.diagrams.net>).

### Представляем нового робота – КАК ОН БУДЕТ ВЫГЛЯДЕТЬ

В начале этой книги (в главе 2) вы узнали, как выглядят схемы роботов. Тогда я предложил вам сделать наброски от руки. Вы находились на самой ранней



стадии создания робота, поэтому не проблема, если наброски были грубыми и не совсем точными. Сначала схемы рисуются от руки, а уже после можно перейти к созданию более детализированных блоков и компоновочных схем.

Каждый проект чем-то вдохновлен. Возможно, вы готовитесь к соревнованиям или захотели повторить поведение другого робота или даже животного (крабы просто очаровательны!). Другим источником вдохновения может стать интерес к новому для вас компоненту или желание освоить новый навык (или поэкспериментировать с ним). Может быть, у вас даже есть список удивительных роботов, которых вы хотите попробовать создать.

Прежде чем создавать робота, составьте краткий список его функций, датчиков/выходов и других важных аспектов. Это позволит вам сосредоточиться на том, что действительно важно. В качестве примера приведу список, который я составил для проекта SpiderBot, описанного в главе 10. В него входит следующее:

- у робота будет 6 «ног» (да, как у обычного насекомого, а не как у паука);
- он будет предназначен для экспериментов с «ногами» и походкой;
- он будет способен избегать столкновений со стенами.

Первичный набросок может выглядеть просто как палочка с шестью «ногами», несколькими квадратами с одной стороны (они обозначают ультразвуковые датчики) и стрелочками, поясняющими, что означает тот или иной элемент. В главе 2 мы обсуждали это подробно. На рис. 19.1 показана простая схема.

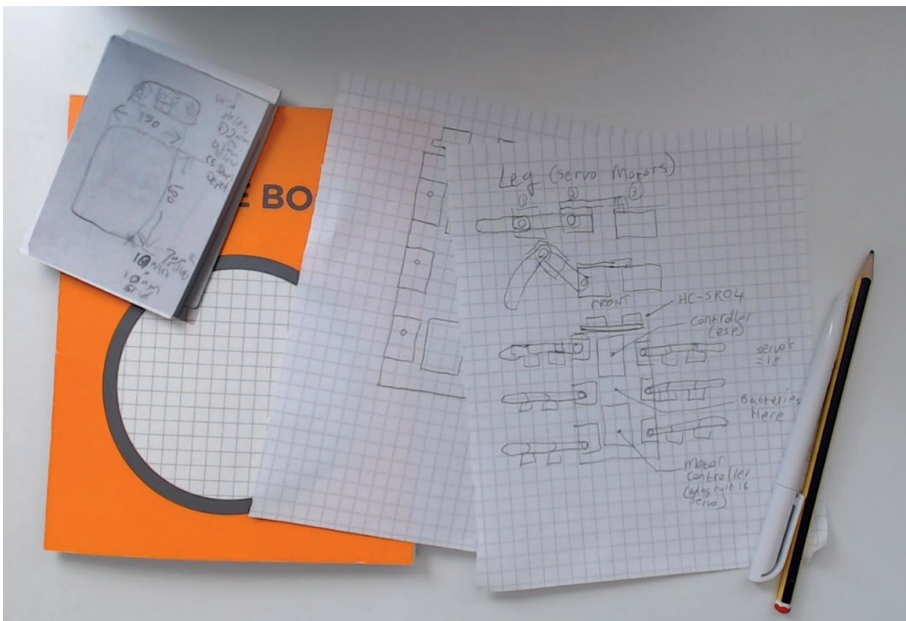


Рис. 19.1. Набросок идеи на бумаге

Как показано на рис. 19.1, первичные наброски я предпочитаю рисовать ручкой на бумаге в клетку (если ее нет под рукой, подойдет любая другая).

Вы можете сделать двумерный или трехмерный набросок, а также изобразить, как выглядит робот в профиль. Ниже представлено несколько советов:

- сначала делайте легкие штрихи, а когда будете уверены, что все правильно, обведите их;
- делайте больше заметок – записывайте все, что приходит на ум;
- не думайте о масштабе и габаритах, не старайтесь сделать набросок идеальным. Он нужен лишь для того, чтобы отразить основные идеи. Позже вы определитесь со всеми тонкостями и учтете их;
- запишите дату создания рисунка и дайте ему рабочее название (в дальнейшем можно придумать название получше);
- не стесняйтесь комбинировать блоки с другими элементами;
- всегда держите при себе ручку/карандаш и блокнот или листы бумаги, чтобы вы могли фиксировать приходящие в голову идеи. Если у вас под рукой есть белая доска для маркеров, то можете записывать все на ней. Делать наброски карандашом – удобно, поскольку всегда можно стереть и нарисовать заново, а ручку просто легко носить в сумке или кармане;
- сначала уделяйте время идеям общего плана, а потом переходите к деталям. Очень легко увязнуть в тонкостях одного аспекта, забыть о других и в итоге не успеть уделить им время. Я рекомендую делать небольшие заметки-напоминания.

Помните, что вы можете вернуться к наброску на любом этапе создания робота, например когда у вас появятся новые идеи, возникнет какая-либо проблема или вы просто захотите усовершенствовать свой проект. Лучше всего сразу записывать основные пункты и делать набросок. Не стоит ждать, пока вы доберетесь до компьютера, или тратить время на создание идеального рисунка – это отвлекает, и вы можете забыть о другой фантастической идее, которая есть в голове.

После того как вы перечислите основные аспекты и сделаете набросок, у вас сформируется целостное представление о том, что будет представлять из себя проект. Теперь можно перейти к созданию более точной блок-схемы.

## Создание блок-схемы

Вспомните блок-схемы, которые мы начали создавать в главе 2, а затем делали это на протяжении всей книги. В таком виде можно представить любого робота. В этой схеме мы отразим блоки для каждого входа и выхода, а затем добавим блоки контроллера и интерфейса. Помните, что схема не должна быть идеальной, главное – чтобы на ней было четко видно, какие части нужно соединить и как это сделать. Вполне вероятно, что в первоначальную блок-схему нужно будет вносить изменения (например, если вы столкнетесь с ограничениями, о которых не могли предположить раньше).

На рис. 19.2 показаны блок-схемы робота SpiderBot на разных этапах его создания.

Как показано на рис. 19.2 слева, сначала я знал только то, что у каждой «ноги» будет по три сервопривода. Я изобразил все это на бумаге, а также добавил дальномер и указал возможность подключения к Wi-Fi.

В правой части рисунка показана блок-схема на более позднем этапе. Я добавил в нее контроллеры для компонентов, а также грубо наметил подключения. Блок-схема не так точна, как принципиальная схема. Я сделал ее на скорую

руку с помощью <https://app.diagrams.net>. Помните, что по мере того, как вы будете больше узнавать о роботе и контроллерах, схема может измениться.

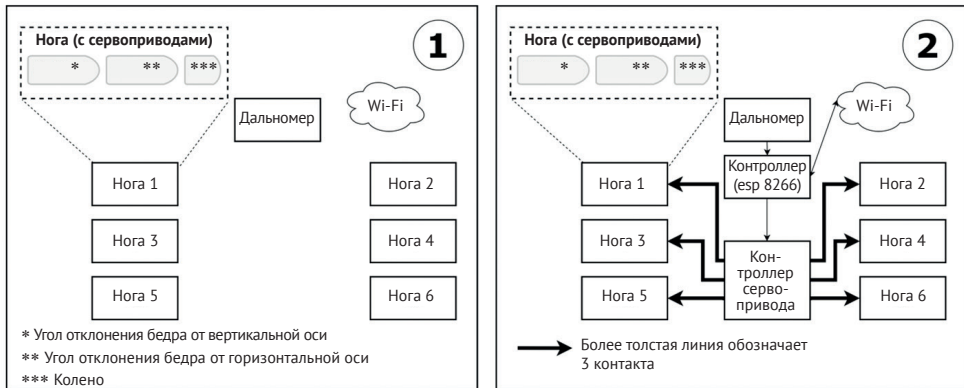


Рис. 19.2. Блок-схемы на разных этапах создания робота

Также вы можете создавать блок-схемы для дополнительных устройств и компонентов. Я рекомендую делать так для всех частей, которые кажутся вам сложными. Переносить подобные идеи на бумагу очень важно, поскольку это позволяет представить их яснее, выявить недостатки и вообще не забыть о той или иной части.

Следующую блок-схему мы создадим для программной части в разделе «Планирование кода».

Теперь, когда у вас есть простой набросок робота и блок-схема, можно приступить к выбору компонентов.

## ВЫБОР КОМПОНЕНТОВ

В этой книге мы уже рассмотрели плюсы и минусы разных типов датчиков, наборов для шасси, контроллеров и т. д. В главе 6 мы определили, какие компоненты подходят лучше всего по весу, сложности, доступности (важно, чтобы любой компонент можно было легко заменить) и стоимости.

Если вы вдохновились набором для создания определенного робота (например, идея SpiderBot пришла, когда я увидел набор для создания гексапода, а вас может заинтересовать другой проект, например робот-манипулятор или робот на гусеницах), это значительно облегчит выбор компонентов. Моему роботу требовалось 18 сервоприводов, но контроллер позволял подключить только 16, поэтому я решил использовать два контакта **ввода/вывода** для оставшихся сервоприводов. К слову, это значительно усложнило программную часть.

Еще одним сложным решением был выбор главного контроллера. Мне хотелось, чтобы SpiderBot поддерживал Wi-Fi, но выполнять обработку видеоданных от него не требовалось, поэтому я выбрал небольшой и недорогой контроллер ESP8266 с малым потреблением мощности.

Что касается питания, я знал, что для всех сервоприводов потребуется серьезная система питания, но большой вес робот поднять не сможет. В резуль-

тате я выбрал специализированную литий-полимерную аккумуляторную батарею с зарядным устройством и схемой защиты.

Важнейшая часть при выборе компонентов – это проверка их размещения в системе.

### **Схема для проверки размещения компонентов**

При выборе компонентов учитывайте, как они будут сочетаться друг с другом. Четко определите, как контроллер двигателя будет взаимодействовать с главным контроллером. Будут ли эти два компонента использоваться вместе, или вы готовы создать новый сложный интерфейс? Подумайте, какие компоненты вы, вероятнее всего, приобретете, и составьте из них схему для проверки размещения, учитывая габариты. Такую схему вы уже создавали в главе 6. Я настоятельно рекомендую вам сделать это перед покупкой.

### **Приобретение компонентов**

Следующий вопрос – выбор места, где приобретать компоненты. У меня есть любимые магазины (это [coolcomponents.co.uk](http://coolcomponents.co.uk), [shop.pimoroni.com](http://shop.pimoroni.com) и [thepihut.com](http://thepihut.com)), но вы можете выбрать те, которые доступны в вашем регионе. Узнайте, где можно приобрести товары от Pimoroni, SparkFun, Raspberry Pi и Adafruit.

Модули можно заказать на веб-сайтах Amazon, Aliexpress или eBay (при этом четко опишите, что именно вам нужно, и уточните условия поддержки покупателей). Отдельные компоненты можно найти в крупных магазинах, таких как Element14, Mouser, RS и Digi-Key. Готовых модулей у них не так много, но это проверенные магазины с большим ассортиментом.

Почти все доступно для приобретения онлайн, но можно посетить и обычные магазины электронных и механических компонентов.

В дальнейшем вы можете использовать те компоненты, которые будут оставаться от предыдущих проектов. В качестве шасси можно использовать детские игрушки. Часто робототехники дают вторую жизнь двигателям и датчикам от старых принтеров и электромеханических систем (в этом случае будьте осторожны). При использовании таких компонентов не забудьте составить схему размещения и проверить, подходят ли они.

### **Сборка робота**

Итак, вы готовы к сборке. Инструкции для этого процесса можно найти в главах 6 и 7. Также можно обратиться к руководству по пайке из главы 12. Не забывайте и о дополнительных материалах и навыках, описанных в главе 18, – это значительно расширит ваши возможности.

После того как вы приобрели нужные компоненты и начали строить робота, самое время перейти к разработке кода.

## **ПЛАНИРОВАНИЕ КОДА**

В главе 2 мы начали планировать уровни кода, а затем рассмотрели это подробнее в разделе «Создание объекта Robot» из главы 7.

Давайте вспомним, как мы планировали структуру кода и его уровни.

## Уровни системы

Основная идея заключается в создании уровней кода в системе, как показано на рис. 19.3.

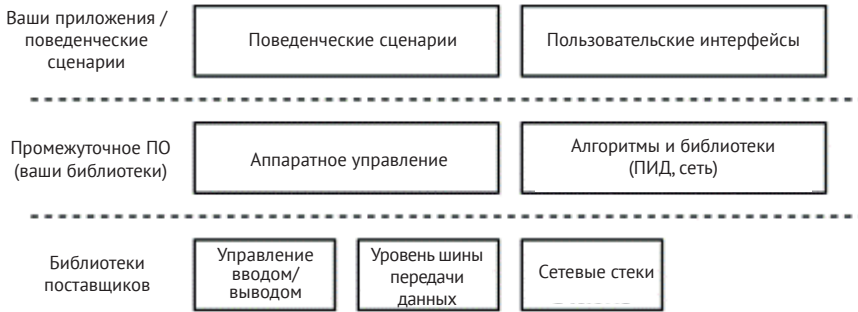


Рис. 19.3. Уровни программного обеспечения робота

На рис. 19.3 показаны возможные уровни системы:

- в нижней части стека находятся **библиотеки поставщиков**. Они, как следует из названия, обычно предоставляются поставщиками аппаратных компонентов, третьими лицами или сообществами. Среди таких библиотек можно отметить используемые в этой книге `gpiozero`, а также библиотеки `Arduino` для этой экосистемы. Нижний уровень включает в себя управление вводом/выводом, шины и сетевые стеки;
- следующий уровень – **библиотеки и промежуточное ПО**. Такое ПО может предоставляться сторонними производителями. Например, это могут быть библиотеки более высокого уровня для взаимодействия со специальными аппаратными компонентами. Уровень промежуточного ПО включает в себя разработанные вами абстракции, например те, которые заставляют всех двухколесных роботов двигаться одинаково, даже если у них разные сторонние библиотеки. Также на этом уровне находятся алгоритмы и библиотеки. Сообществом предоставлена библиотека `OpenCV`. Здесь же могут быть ваши **ПИД-алгоритм** и конвейеры сценариев распознавания объектов. Этот уровень включает в себя компоненты для создания приложений и поведенческих сценариев;
- верхний уровень связывает два предыдущих. С помощью алгоритмов он преобразует входные данные датчиков в выходные сигналы двигателей и позволяет создавать удобные интерфейсы управления или информационные панели. На этом уровне создаются поведенческие сценарии и приложения для робота.

У простых роботов компонентами уровней являются простые функции и классы. В более сложных проектах это могут быть различные программные компоненты, взаимодействующие через общую программную шину (например, очередь сообщений или подключенные службы). Разработанная нами библиотека подойдет (на среднем уровне) для многих роботов, которые передвигаются с помощью небольших колес. По мере приобретения опыта библиотеку

можно доработать. Также, если сценарий не имеет каких-либо аппаратных ограничений, вы можете адаптировать его для других датчиков и выводов.

Чтобы увидеть границы уровней и блоков, изобразите их в виде схемы. Подумайте, какой код будет в тех модулях и блоках, которые вы хотите связать. Не углубляйтесь в то, как работает передача данных через шину **SPI (Serial Peripheral Interface – последовательный периферийный интерфейс)**, – все это остается на аппаратном уровне поставщика; лучше подумайте о создании шаблонов для поведенческих сценариев, использующих светодиоды.

Теперь, когда вы узнали об основных уровнях и некоторых компонентах, поговорим о том, как между ними передается информация. Для этого обратимся к схемам потоков данных.

### Схемы потоков данных

Вы можете использовать схемы для представления потока данных поведенческого сценария; например, в главе 13 мы создавали такие схемы для ПИД-алгоритма и обратной связи. С их помощью можно проследить процесс преобразования изображений в сценариях отслеживания цветных объектов и лиц. Разумеется, одной схемы, скорее всего, будет мало – чтобы отразить все аспекты поведенческого сценария, их потребуется несколько.

Уделите больше времени сложным аспектам, таким как математические операции, которые необходимы для сложных датчиков/движений. С первого раза может не получиться, но позже вы разберетесь и поймете, почему система вела себя не так, как ожидалось. На этом этапе можно поискать похожие проекты в интернете или выбрать одну из рекомендованных книг по теме. В большинстве случаев настойчивость того стоит.

### Формальные схемы

У многих схем есть формальные представления, например у блок-схем или схем **унифицированного языка моделирования (UML – Unified Modeling Language)**. Их стоит изучить и использовать в дальнейшем. У [app.diagrams.net](http://app.diagrams.net) есть хорошая библиотека схем. Главная задача схемы – передавать информацию, поэтому при ее создании вы должны выразить свои мысли так, чтобы вы или другие члены команды (если она есть) смогли понять идею даже через полгода.

Простые поведенческие сценарии в дальнейшем могут послужить компонентами для более сложных и интересных вариантов. Например, наш поведенческий сценарий движения по прямой был основой для сценария движения по квадратной траектории.

### Разработка программ для робота

Теперь можно перейти к разработке программ, но будьте готовы, что для этого нужно будет провести не один цикл планирования, реализации, тестирования и изучения. Не расстраивайтесь, если что-то не получилось с первого раза, – напротив, это отличная возможность развить навыки. Больше всего времени на изучение затрачивается на этапах планирования и проверки (которая может закончиться неудачей). Если все получится с первого раза, то вы можете не задерживаться и идти дальше. При разработке каждого сценария в этой книге



возникало множество проблем, которые я старался исправить. Помните, что наша тактика – это метод проб и ошибок.

Итак, от планирования кода вы перешли к завершению построения робота. Как рассказать о своем проекте миру и в случае необходимости попросить о помощи?

## РАССКАЗЫВАЕМ МИРУ О ПРОЕКТЕ

На этом этапе у вас наверняка возникнут вопросы о том, как действовать дальше, и о том, как решать возникающие проблемы. Возможно, вы уже сталкивались с этим и на более ранних этапах. Если у вас есть вопросы или вы чувствуете, что знаете недостаточно, самое время выйти в интернет и обратиться к робототехническому сообществу, как описано в разделе «*Робототехнические интернет-сообщества*» в главе 18.

В Twitter и на Stack Overflow можно спрашивать и отвечать на вопросы других робототехников. Поделитесь своим опытом на YouTube и посмотрите другие каналы. Не откладывайте общение, ваше творение не обязано быть идеальным. Подробно расскажите обо всех этапах и проблемах, с которыми вы столкнулись, а также о своих неудачах. Истории про неудачи всегда самые лучшие, поскольку они могут послужить мотивацией для тех, кто столкнулся со сложностями и хочет сдаться.

Для развития новых навыков можно смотреть видеоролики на YouTube, изучать проекты на веб-сайте Instructables и читать различные блоги, но лучше всего посетить площадку для совместной работы или мероприятия, такие как Coder Dojo или Raspberry Jam. Там вы сможете объединиться с другими робототехниками, в том числе и начинающими.

Быть робототехником – значит постоянно учиться. В этой области всегда появляется что-то новое, поскольку она активно исследуется и развивается. Исследователи постоянно расширяют границы человеческих знаний. Вы можете попробовать себя в качестве наставника или помощника, чтобы поделиться знаниями с другими людьми. Возможно, когда-нибудь вы создадите собственный робототехнический набор, разработаете модуль или код, который облегчит входной барьер для новичков, найдете робототехнике новое применение в других областях или изобретете совершенно новый датчик. Как бы то ни было, общаться с участниками робототехнических сообществ крайне увлекательно. Это помогает освежить мысли и найти новые решения.

Проверить робота в действии можно и в мастерской, но самые строгие испытания происходят вне ее, на соревнованиях и демонстрациях. Вы обнаружите новые ошибки, столкнетесь с рядом проблем, которые нужно решить, и найдете много друзей и единомышленников. Существует стереотип, что робототехника – это уединенное хобби или профессия, но на самом деле все совсем не так, ведь этим может заниматься каждый желающий. Так будем же заниматься вместе!

Работа в команде – это полезный и одновременно непростой опыт. Вместе можно создать более амбициозные проекты, нежели чем в одиночку. Единомышленников вы можете найти в местных сообществах.

Теперь вы знаете больше о робототехническом сообществе, где можно найти помощь, вдохновение и достойных конкурентов. Вы узнали, что робототехни-



ка может быть совместным хобби. Возможно, вы даже захотели создать свой блог или YouTube-канал. С нетерпением жду вас в нашем сообществе!

## Выводы

В этой книге вы увидели, как построить своего первого робота и разработать для него программы. Также вы узнали, где найти больше информации и как развивать свои навыки. В последней главе мы обобщили все полученные знания и варианты их применения для планирования и построения следующего робота, а также разработки программ. Вы узнали, как стать частью сообщества и показать свой проект миру.

Книга рассказала, как спроектировать, спланировать и построить робота, разработать для него программы и проверить систему в действии. Вы начали осваивать новые навыки по части аппаратной сборки (например, пайку), научились создавать простое ПО (например, чтобы заставить робота двигаться) и слегка затронули сложные области, таких как компьютерное зрение и инерциальные измерения. Теперь вы умеете исправлять ошибки, находить компромиссы при выборе компонентов, осуществлять точную настройку системы и сохранять резервные копии кода. Вы разработали пользовательские интерфейсы, интеллектуальные поведенческие сценарии и реализовали управление роботом через смартфон.

Вы дошли до конца этой книги, но я надеюсь, что эта точка – лишь начало вашего пути в робототехнике.

# Предметный указатель

## А

абсолютный энкодер 237  
Автоматизированные транспортные средства 352  
адресуемые RGB светодиоды 181  
аккумуляторов 37  
акселерометр 274  
аналого-цифровой преобразователь (АЦП) 273

## Б

Бесколлекторный (бесщеточный) двигатель 41  
библиотека интерфейсов 133

## В

ведущих колеса 37  
вектор 290  
визуальный символ 354  
время пролета 154  
вход 22  
входной файл 95  
выбег интегрального компонента 331  
выход 22  
выходной файл 95

## Г

гироскоп 273  
голосовой помощник 381  
графические примитивы 409

## Д

датчик температуры 273  
датчик Холла 236  
Двигатели 37  
диалог 382  
дисплей 43  
драйвер двигателя 105  
драйверы двигателей 212

## З

Значение цвета 192

## И

инерциальный измерительный модуль 272  
инкрементальный энкодер 237  
интегральное изображение 340  
интерфейс ввода/вывода общего назначения 46  
интерфейс записи и воспроизведения 239  
историю коммитов 87  
итераторы 317

## К

кадры 223  
калибровка 415  
каскады Хаара 340  
Качалка сервопривода 206  
кодовые датчики 29  
контроллер 22  
Контроллер двигателя 37  
корпус 37  
кортеж (tuple) 186

## М

магнитометр 274  
маяк 354  
метод 144  
метод Виолы-Джонса 340  
механизм поворота и наклона 25, 213  
механизм с задними управляемыми колесами 139

## Н

навык 382  
намерение 382  
Напряжение 155  
Насыщенность 192

## О

обнаружение с помощью камеры 354  
обнаружение с помощью оптических датчиков 353

обхода препятствий 23  
объект 143  
объектно-ориентированное  
    программирование 151  
одноплатный компьютер 58  
одометрия 235  
оптический энкодер 236

## П

переменный резистор 236  
печатная плата 491  
Платы расширения HAT 61  
поведенческий уровень 53  
Поворотное колесо 37  
полнофункциональный HAT-  
    контроллер шагового  
    двигателя (Full Function  
    Stepper Motor HAT) 108  
пользовательский интерфейс (UI) 447  
полярная диаграмма 226  
постоянного тока 40  
предел разрешающей способности 46  
предотвращать столкновения 23  
преобразование речи в текст 381  
привода 40  
принцип Аккермана в рулевом  
    управлении 139  
пропорционально-интегрально-  
    дифференцирующий регулятор 254

## Р

размер блока данных 95  
реечный механизм управления 138  
реплика 382  
Рефакторинг 260  
робот 21  
рулевая рейка 138

## С

Светоизлучающие диоды 181  
селектор 465  
Сенсоры 37  
Серводвигатель 41  
сервомеханизм 41  
Сервоприводы, 202  
система автоматизированного  
    проектирования (САПР) 493  
системабортовогоповорота 140  
система координат осей корпуса 289  
система управления с обратной  
    связью 203

следование по визуальным линиям 353  
следование по магнитным линиям 354  
словарь 382  
слово для пробуждения 381  
степень свободы 274  
строка документации 220

## Т

таблица суммированных площадей  
    340  
тахометр 235  
твердое железо 427

## У

унифицированный язык  
    моделирования (UML) 508  
Устройства отладки 37

## Ф

фиксированные колеса 138  
Формирователь ШИМ 212

## Ц

цветовой тон 192  
цикл REPL 188

## Ч

ЧПУ 491

## Ш

Шасси 37  
широотно-импульсной модуляции  
    (ШИМ) 47

## Э

Электродвигатель постоянного тока 40

## А

ALSA 387  
API 380  
Arduino Leonardo 48

## В

balenaEtcher 88  
BEC 109

## С

CSI 60  
CSS 459

## Д

Debian 62

**G**

git-репозиторий 87

GND 60

**H**

HSL 192

HSV 191

**I**

I2S 47

**J**

JavaScript Object Notation (JSON) 475

Jinja2 450

**M**

micro:bit 48

Mycroft 381

**N**

NodeMCU 48

NumPy 314

**O**

Open Computer Vision (OpenCV) 313

**P**

PCBA 497

PCM 45

PIC 49

**R**

Raspberry Pi 48

Raspberry Pi 3A+ 50

Raspberry Pi 4B 50

Raspberry Pi OS Lite 62

Raspberry Pi Zero W 50

RGB 181

RGB-значения 183

**S**

Serial 60

SFTP 84

SPI 47

SVG 463

systemd 479

**V**

Visual Python (VPython) 283

Книги издательства «ДМК Пресс» можно заказать  
в торгово-издательском холдинге «КТК Галактика» наложенным платежом,  
выслав открытку или письмо по почтовому адресу:  
115487, г. Москва, пр. Андропова д. 38 оф. 10.  
При оформлении заказа следует указать адрес (полностью),  
по которому должны быть высланы книги;  
фамилию, имя и отчество получателя.  
Желательно также указать свой телефон и электронный адрес.  
Эти книги вы можете заказать и в интернет-магазине: **www.galaktika-dmk.com**.  
Оптовые закупки: тел. (499) 782-38-89.  
Электронный адрес: **books@alians-kniga.ru**.

## **Дэнни Стейпл**

### **Устройство и программирование автономных роботов**

Главный редактор	<i>Мовчан Д. А.</i>
	<i>dmkpress@gmail.com</i>
Зам. главного редактора	<i>Сенченкова Е. А.</i>
Перевод	<i>Шевчук Е. В.</i>
Научный редактор	<i>Яценков В. С.</i>
Корректор	<i>Абросимова Л. А.</i>
Верстка	<i>Луценко С. В.</i>
Дизайн обложки	<i>Бурмистрова Е. А.</i>

Формат 70×100 1/16.  
Гарнитура «PT Serif». Печать цифровая.  
Усл. печ. л. 42,25. Тираж 200 экз.

Веб-сайт издательства: [www.dmkpress.com](http://www.dmkpress.com)

# Устройство и программирование автономных роботов

Мы живем в эпоху, когда самые сложные для человека задачи уже автоматизированы. В наше время необходимы «умные» и интеллектуальные роботы, выполняющие задачи точно и эффективно. Создать такого робота можно с помощью Python и Raspberry Pi.

Эта книга начинается с ознакомления со структурой робота и рассказывает, как планировать, строить роботизированные системы и разрабатывать для них код. По мере прочтения вы узнаете, как добавить роботу различные выходы и датчики, изучить новые навыки аппаратной сборки и разработать интересные поведенческие скрипты, основывающиеся на данных от датчиков. Далее вы настроите подключение робота к сети и Wi-Fi, а затем реализуете управление им через смартфон.

К концу книги вы создадите «умного» робота, способного выполнять базовые операции искусственного интеллекта (ИИ).

## Рассматриваемые темы:

- настройка Raspberry Pi для использования в роботизированной системе;
- взаимодействие Raspberry Pi с двигателями и датчиками;
- разработка кода для интересных и интеллектуальных поведенческих сценариев;
- первые шаги в области задач искусственного интеллекта (распознавание речи и обработка видеоданных);
- управление интеллектуальным роботом через Wi-Fi;
- выбор компонентов.

Интернет-магазин:  
[www.dmkpress.com](http://www.dmkpress.com)

Оптовая продажа:  
КТК «Галактика»  
[books@aliants-kniga.ru](mailto:books@aliants-kniga.ru)

**Packt**  
  
ИЗДАТЕЛЬСТВО  
[www.dmk.pf](http://www.dmk.pf)

ISBN 978-5-97060-989-7



9 785970 609897 >