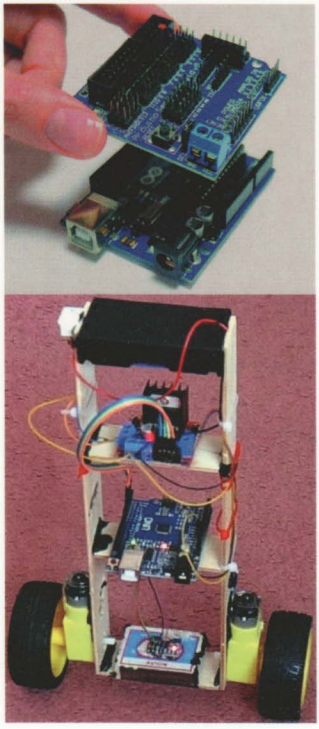
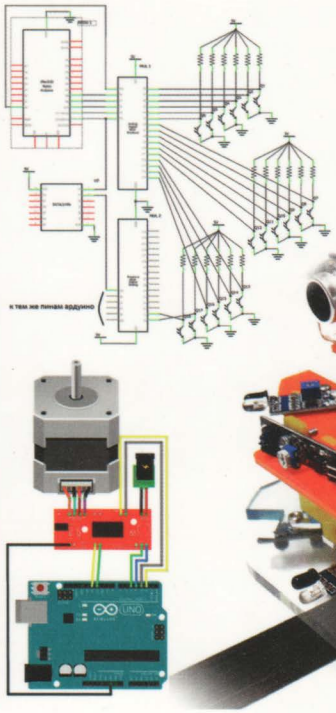
Электроника

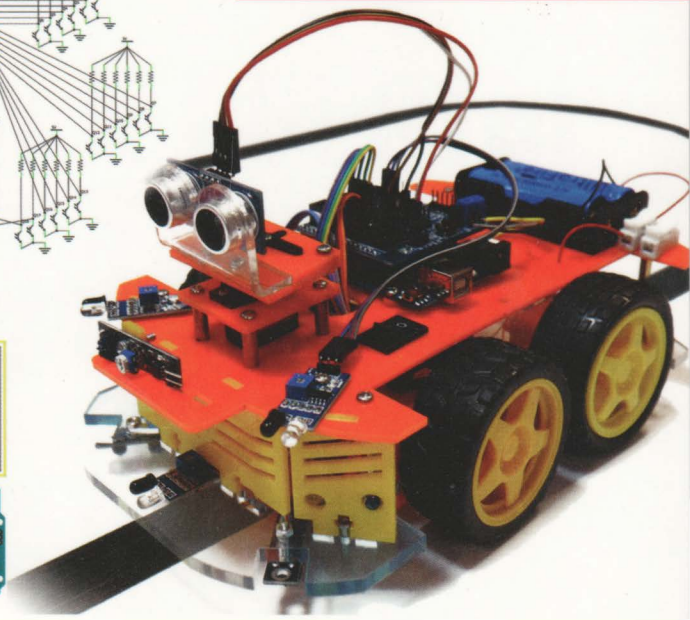
Михаил Момот

bhv®





Мобильные роботы любой сложности легко и быстро!



МОБИЛЬНЫЕ РОБОТЫ НА БАЗЕ Arduino

2-е издание

+ Материалы на www.bhv.ru

Михаил Момот

МОБИЛЬНЫЕ РОБОТЫ НА БАЗЕ Arduino

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2018

УДК 007.52
ББК 32.816
М76

Момот М. В.

М76 Мобильные роботы на базе Arduino. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2018. — 336 с.: ил. — (Электроника)

ISBN 978-5-9775-3861-9

Руководство для начинающих конструкторов написано в форме практических проектов по построению мобильных роботов. Для их реализации выбрана популярная платформа Arduino и единая базовая четырехколесная конструкция. Сложность проектов возрастает от простой, управляемой с пульта «машинки» до интеллектуального говорящего робота. Рассказано, как управлять моторами, осуществлять сборку механики и электроники, программировать основные функции и управлять роботом. Роботы смогут обходить препятствия, выбираться из запутанных лабиринтов, искать кегли и определять их цвета, ориентироваться по электронному компасу, гироскопу и даже балансировать на двух колесах.

Во 2-м издании обновлены все алгоритмы, добавлены проекты гироскоп-акселерометр, говорящий робот, голосовое управление роботом, механическая «рука» и др.

Электронный архив, находящийся на сайте издательства, содержит детали робота для печати на 3d-принтере, векторные рисунки для резки лазером, листинги, дополнительные библиотеки и программы.

Для читателей, интересующихся электроникой и робототехникой

УДК 007.52
ББК 32.816

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Марины Дамбиевой*

Подписано в печать 31.10.17.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 27,09.
Тираж 2000 экз. Заказ № 5452.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ООО "Печатное дело",
142300, МО, г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-3861-9

© ООО "БХВ", 2018
© Оформление. ООО "БХВ-Петербург", 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Здравствуй, дорогой читатель! Хотя я не уверен, что правильно называть того, кто читает эту книгу, читателем, скорее, он Конструктор или Изобретатель. Потому, что чтение только лишь книги без сопутствующей сборки и программирования роботов не столь интересно.

Эту книгу я составлял как руководство для начинающих Конструкторов, т. е. людей, которым нравится *конструировать*. А за основу взял конструирование несложных роботов на весьма популярной в настоящее время платформе Arduino. Arduino же выбрал потому, что проекты, выполненные на ее основе, весьма простые и функциональные. Платформа Arduino открытая, а это значит, что изготавливать дополнительные модули для нее может любой человек или организация, то же относится и к программам.

Прочитав книгу, вы научитесь программировать на платформе Arduino и обращаться с электронными компонентами, из которых состоят роботы. Поймете принцип действия датчиков, при помощи которых роботы следят за внешним миром. Будете собирать своих оригинальных роботов и удаленно управлять их работой.

Книга разбита на главы.

- *Главы с первой по четвертую* являются вводными — они дают основу, подготавливают вас к конструированию робота. Так, *первая глава* посвящена составным частям, из которых может быть сделан робот. *Вторая глава* учит выполнять электрические соединения. *Третья* — рассказывает об источниках электрического питания, их стабилизации и выборе подходящего источника питания для робота. В *четвертой главе* даны основы программирования контроллеров Arduino, поскольку начинать программировать можно еще до установки Arduino в робота.
- *Главы с пятой по одиннадцатую* посвящены сборке базовой конструкции робота. Так, *пятая глава* содержит обзор ходовых частей и учит основам управления двигателями. *Шестая глава* дает практические навыки по сборке ходовой части. *Седьмая глава* дает основы управления движениями робота, в ней разрабатыва-

ется первая — базовая программа, содержащая блоки управления его движениями. *Восьмая глава* посвящена способам дистанционного управления роботом со смартфона на Android или с пульта дистанционного управления, подобного телевизионному пульту. В *девятой главе* описано применение одного из самых простых датчиков — датчика освещенности, помогающего роботу при движении по черной линии. *Десятая глава* дает знания и практический навык по применению ультразвукового дальномера, расположенного на вращающейся «голове» робота и помогающего ему обходить препятствия. *Одиннадцатая глава* научит настраивать робота и проводить отладку программы — здесь вы создадите робота, который сможет объезжать препятствия.

- *Главы с двенадцатой по семнадцатую* вводят в проектирование конкретных роботов, которые должны, выполняя определенные действия, добиваться поставленных перед ними разнообразных целей. Так, *глава двенадцатая* познакомит вас с различными детекторами препятствий, даст понимание программной логики, которая позволяет роботу не заблудиться в запутанном лабиринте. *Глава тринадцатая* научит пользоваться электронным компасом и использовать его для ориентирования робота в пространстве по сторонам света. *Четырнадцатая глава* посвящена ориентированию робота в пространстве при помощи специальных электронных приборов — гироскопа и акселерометра, робот научится ехать прямо, не сбиваясь с указанного направления и точно поворачивать на заданный угол. *Глава пятнадцатая* даст знания о том, как действует робот при решении задачи поиска стоящей поодаль кегли и определения ее цвета. В *главе шестнадцатой* мы научим робота говорить — сообщать о различных событиях, которые с ним произошли. В *семнадцатой главе* рассказано, как сконструировать двухколесного робота с высоким центром тяжести и научить его удерживать равновесие.
- *Восемнадцатая глава* посвящена дальнейшим улучшениям и модернизации рассмотренных в книге конструкций. В ней предлагаются несколько способов, позволяющих увеличить количество портов ввода/вывода на используемых управляющих платах, обсуждаются достоинства и недостатки шаговых двигателей. Приведена программа голосового управления роботом — вы сможете установить вместо головы робота смартфон и давать через него роботу голосом различные команды, которые он будет выполнять. Для захвата мяча и других предметов приведен вариант конструкции «руки» робота, которую несложно вырезать самостоятельно из плоских материалов: оргстекла или фанеры.

Книга поможет вам разобраться в работе ряда датчиков: ультразвукового датчика расстояния, детектора препятствия, датчика цвета (черный или белый), электронного компаса и гироскопа — чтобы вы смогли обеспечить своим роботам требуемый функционал. В книге также приведены советы по самостоятельному изготовлению некоторых датчиков.

В электронном архиве, сопровождающем книгу, содержатся все программные модули, которые приведены в книге, и даже некоторые не приведенные или приведенные не полностью. Кроме них в архиве содержатся файлы с векторными рисунками и 3D-моделью «руки» робота. По этим рисункам можно либо самостоятельно

вырезать ее из оргстекла/фанеры, либо заказать подобную резку, например, при помощи лазерного резака.

Электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538619.zip** или со страницы книги на сайте **www.bhv.ru**. Все необходимые листинги расположены там в папках с номерами, соответствующими их номеру в тексте книги (см. *приложение 2*).

Я уверен, что изучение книги и конструирование роботов будет вам не только полезно с точки зрения получения новых знаний и умений, но и станет увлекательным и интересным занятием.

С уважением, Михаил Момот

ГЛАВА 1

ОСНОВНЫЕ СОСТАВНЫЕ ЧАСТИ РОБОТА

Что мы имеем в виду, когда произносим слово «робот»? В основном, подразумевается некое техническое творение, которое либо самостоятельно, либо посредством удаленного управления может совершать определенные действия.

Слово «робот» впервые применил для обозначения искусственных людей чешский писатель Карел Чапек в своей пьесе «Россумские универсальные роботы». К. Чапек называл так искусственные создания, внешне не отличающиеся от обычных людей, но безгранично преданные человечеству и поэтому безмерно эксплуатируемые. Современные роботы, это, конечно, не искусственные люди, но довольно продвинутые автоматические механизмы, способные на очень многое благодаря наличию у них развитой структуры компонентов, отвечающих за разнообразные действия.

Внутренняя начинка роботов не сильно отличается от обычного персонального компьютера. Но, имея другие цели, роботы выглядят иначе. Любого робота можно условно разделить на следующие части:

- информационно-измерительная система;
- система принятия решений (система внутреннего управления);
- система связи;
- исполнительная система;
- система энергоснабжения;
- механика робота (скелет).

Информационно-измерительная система

Информационно-измерительная система — это совокупность органов чувств робота. Она отвечает за восприятие роботом внешнего мира. Элементы информационно-измерительной системы называют *датчиками* или *сенсорами*. Самым простым

из них является датчик касания, который срабатывает от замыкания контактов. Сюда также относятся дальномеры и датчики расстояния, датчики и измерители освещенности, датчики шума, различные магнитные датчики, а также системы машинного зрения, гирокомпасы, акселерометры, температурные датчики и многие другие. Далее мы рассмотрим основные электронные и электрические элементы робота и их схематические обозначения.

Программа Fritzing

При подготовке книги для рисования электрических схем использовалась программа Fritzing, специально разработанная для среды Arduino. Поэтому большинство схем, приведенных в книге, нарисованы с использованием этой программы. Fritzing является программой свободно распространяемой и может быть получена с одноименного сайта.

Датчик касания

Рассмотрим наиболее распространенные органы чувств робота более подробно. На рис. 1.1 изображены датчик касания и его принципиальная схема. Датчик реализован в виде нефиксируемой кнопки с рычажком, при нажатии на который одна пара контактов замыкается, а другая размыкается.

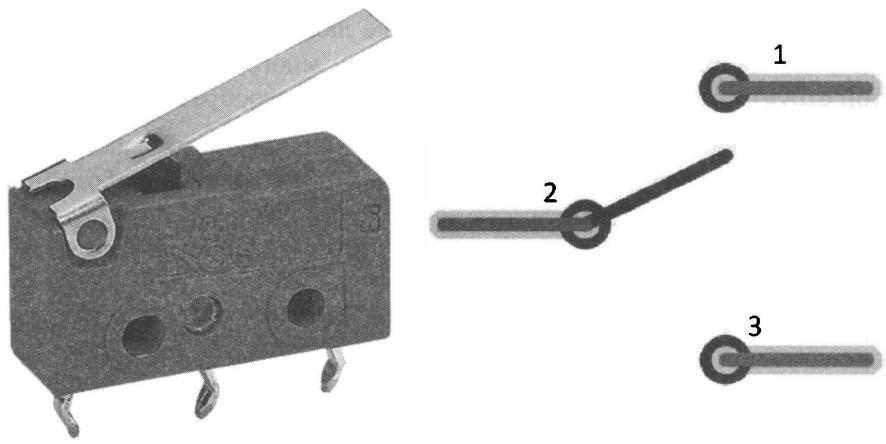


Рис. 1.1. Датчик касания

Датчик температуры

На рис. 1.2 показан датчик температуры — принцип его работы основан на изменении сопротивления при изменении температуры, что приводит к изменению выходного напряжения, которое и служит для замера температуры.

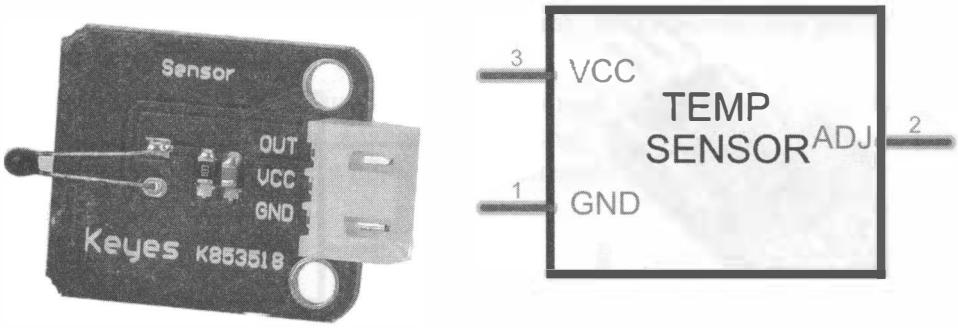


Рис. 1.2. Датчик температуры

Датчик освещенности

На рис. 1.3 изображен пороговый датчик освещенности, который срабатывает по достижении освещенностью определенного значения. Пороговое значение срабатывания датчика можно регулировать. Основой датчика служит фоторезистор, сопротивление которого изменяется при изменении освещенности.

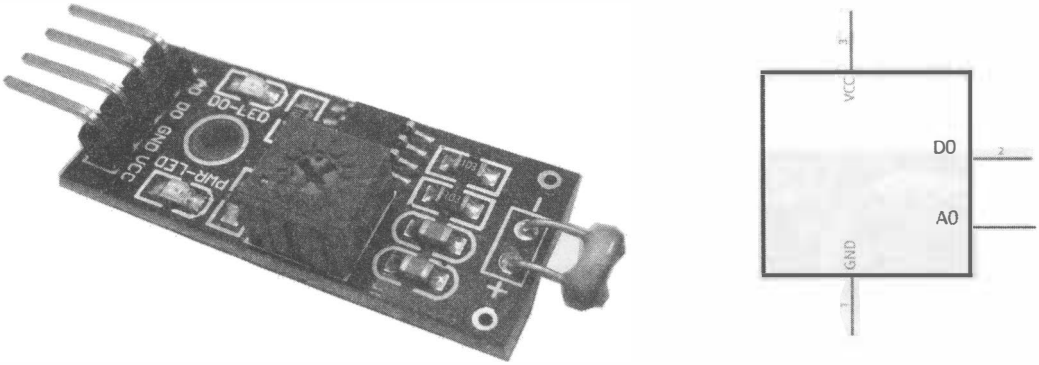
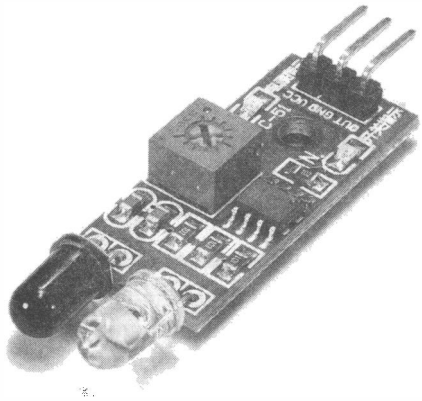


Рис. 1.3. Датчик освещенности

Датчик препятствия

Датчик препятствия (рис. 1.4) состоит из инфракрасного светодиода, излучающего в невидимом человеку инфракрасном диапазоне, и фототранзистора. Светодиод светит на препятствие, фототранзистор реагирует на отраженный свет, величина тока фототранзистора зависит от уровня освещенности. За счет подстройки можно изменять пороговый уровень срабатывания датчика и тем самым менять расстояние срабатывания. Недостатком датчика является реакция на сторонние источники инфракрасного света.



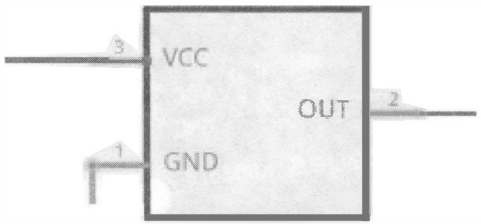
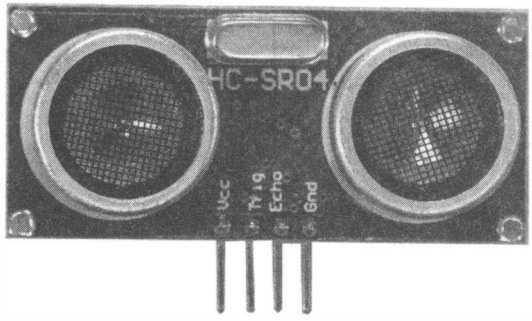


Рис. 1.4. Датчик препятствия

Ультразвуковой датчик расстояния

Более сложным, но и более точным, чем инфракрасный датчик препятствия, является ультразвуковой датчик расстояния (рис. 1.5). Он излучает ультразвуковую волну и принимает отраженный сигнал, расстояние определяется по времени распространения сигнала до препятствия и обратно. При этом точность датчика составляет несколько миллиметров. Однако на определение расстояния требуется некоторое время, которое робот должен провести неподвижно.



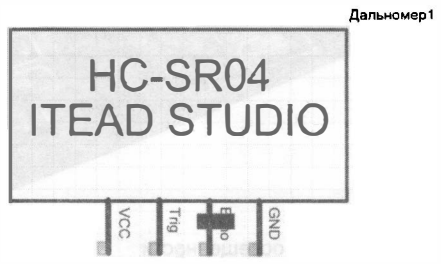


Рис. 1.5. Ультразвуковой датчик расстояния

Оптический рефлекторный датчик расстояния

Интересным устройством является оптический рефлекторный датчик расстояния, иногда также называемый *инфракрасным датчиком расстояния* (рис. 1.6). В основе его работы лежит триангуляционный принцип измерений.

Датчик состоит из передатчика, излучающего прямо вперед, и приемника, отстоящего от него на известном (базовом) расстоянии. Вместе с отражающим объектом они образуют прямоугольный треугольник. Чем дальше объект, тем больше угол

в основании прямоугольника. Зная угол и базу, легко вычислить расстояние тригонометрически.

Следует заметить, что датчик имеет нелинейный выход — при изменении расстояния до объекта сигнал на аналоговом выходе датчика изменяется непропорционально, поэтому для расчета применяется специальная формула.

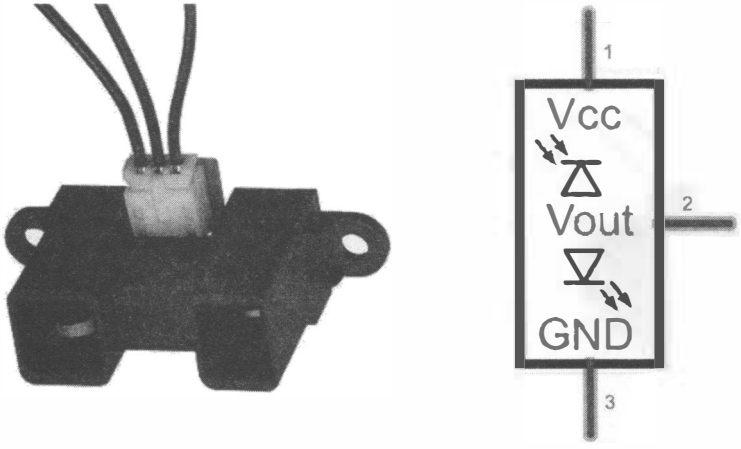


Рис. 1.6. Оптический рефлекторный датчик расстояния

Детектор шума

На рис. 1.7 показан детектор шума на конденсаторном микрофоне, срабатывающий на заданный уровень звука. Порог его срабатывания можно плавно регулировать.

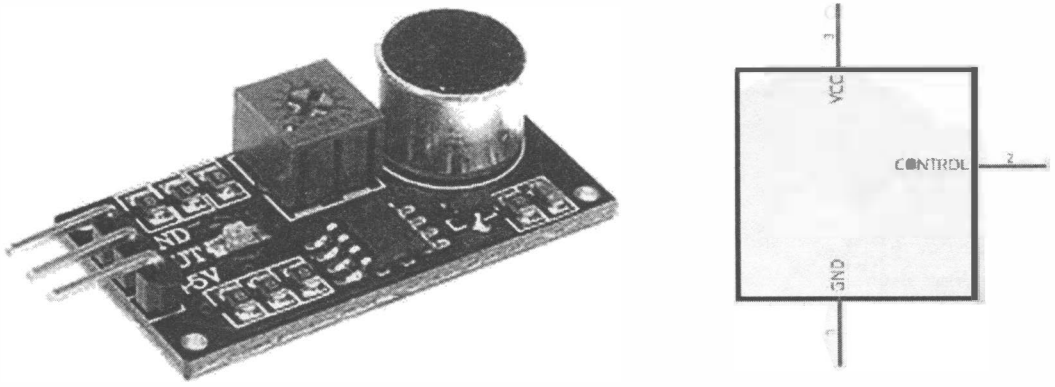
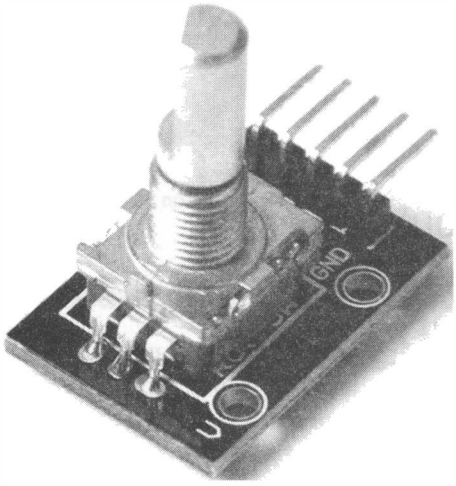
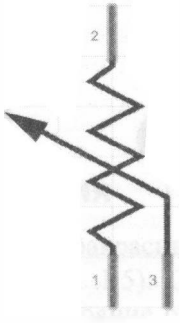


Рис. 1.7. Детектор шума

Энкодер

Энкодером называют датчик угла поворота — устройство, предназначенное для преобразования угла поворота вращающегося вала в электрические сигналы, позволяющие определить величину этого угла. Энкодер, изображенный на рис. 1.8, представляет собой модифицированный переменный резистор.





R1
100kΩ

Рис. 1.8. Энкодер

Датчик движения

Датчик движения (рис. 1.9) состоит из приемника, реагирующего на инфракрасный свет, и линзы Френеля, обеспечивающей большие углы обзора. Конструкция датчика такова, что он не срабатывает на излучение от неподвижных объектов, но срабатывает, когда объект начинает двигаться.

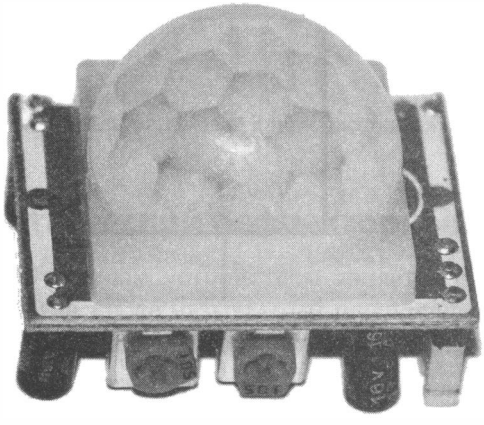


Рис. 1.9. Датчик движения с линзой Френеля

Датчик газа

Датчик газа (рис. 1.10) реагирует на увеличение концентрации углекислого и угарного газов. Его можно использовать при разработке пожарной сигнализации. Если в помещении увеличивается концентрация угарного газа, человек этого не почувствует, а датчик сработает и предупредит о грозящей опасности.

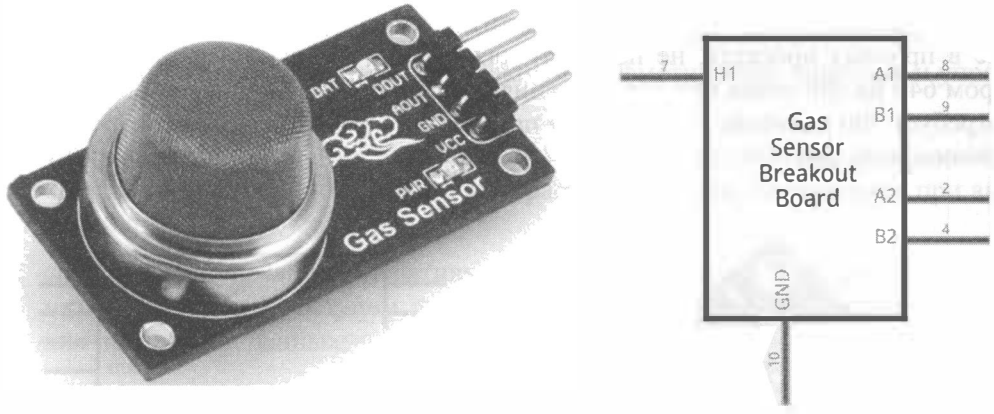


Рис. 1.10. Датчик угарного газа

Датчик влажности

Датчик влажности грунта основан на изменении электрического сопротивления почвы в зависимости от уровня ее влажности. Такой датчик можно с успехом применять в системах полива растений. Полив растений будет проходить тогда, когда влажность почвы уменьшится до определенного предела.

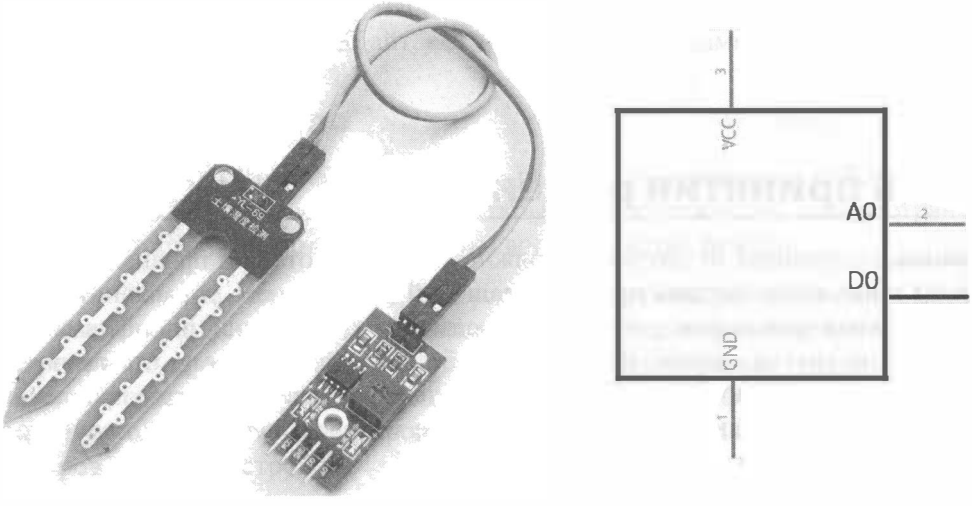


Рис. 1.11. Датчик влажности

Видеокамера

Информационно-измерительная система роботов может включать видеокамеру, которая состоит из проецирующей линзы и фоточувствительной матрицы (рис. 1.12). Изображение проецируется на матрицу, каждый элемент матрицы генерирует сигнал, соответствующий цвету и освещенности определенной точки объекта. Полученная информация, закодированная числами (в случае цифрового сигнала), передается на обработку. Следует отметить, что контроллеры Arduino, используемые в простых проектах, не подходят для обработки видео и фото. Фотокадр размером 640 на 480 точек при 256 значениях освещенности (для черно-белого кадра) потребует 300 килобайт оперативной памяти, чего в большинстве контроллеров Arduino попросту нет.

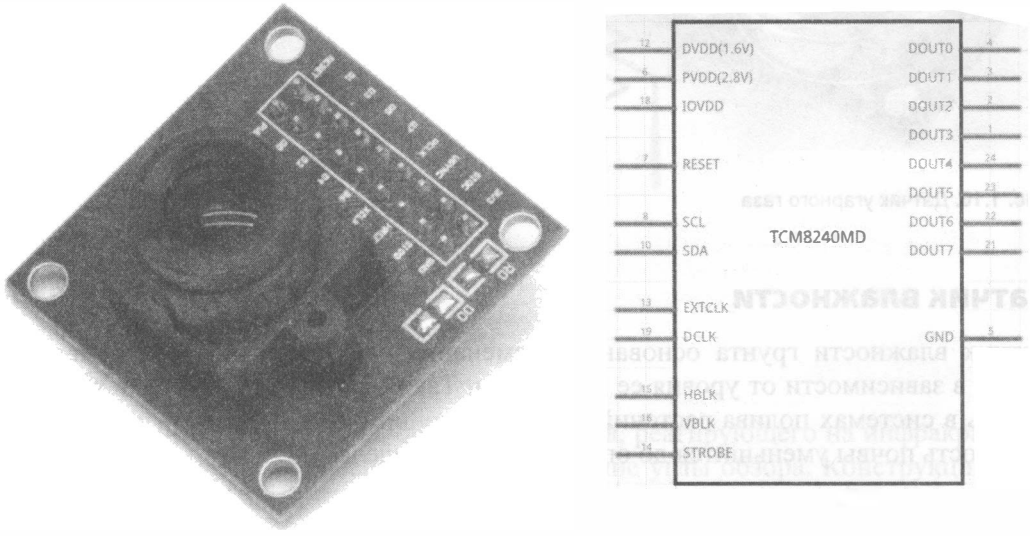


Рис. 1.12. Видеокамера

Система принятия решений

Информация, полученная от датчиков, должна быть обработана и проанализирована — этим занимается система принятия решений (внутреннего управления). Она содержит правила поведения робота в зависимости от времени и информации, поступающей от других систем. Включение робота приводит к тому, что он начинает действовать согласно этим правилам. Под действием понимается управление своими механизмами, включение или отключение моторов, передача и получение информации по системам связи.

Правила поведения роботов содержатся в его памяти в машинных кодах. Но перед этим они создаются людьми сначала в виде алгоритмов, а затем в виде программы

на языке программирования. Роботы сами не способны изменять правила своего поведения.

Система принятия решений является «компьютером» робота и может быть реализована на схожих с ним компонентах. Эту систему можно также назвать центральной нервной системой робота.

Функции системы принятия решений в роботах выполняют микроконтроллеры. Микроконтроллер — это микросхема, включающая в себя несколько устройств:

- *процессор* осуществляет все логические и арифметические операции и управляет выполнением программы робота;
- *постоянная память* (постоянное запоминающее устройство) выполняет роль «жесткого диска» робота, хранит программы и данные, не стирается при выключении питания;
- *оперативная память* (оперативное запоминающее устройство) является памятью с быстрым доступом, используется контроллером при работе для хранения программ и данных;
- *аналого-цифровые преобразователи* преобразуют уровни напряжения в числовую форму;
- *широтно-импульсные генераторы* предназначены для генерации электрических импульсов с определенной частотой и шириной и служат для управления внешними устройствами — например, скоростью вращения двигателя постоянного тока.

Микроконтроллер

Микроконтроллеры функционально схожи с современными компьютерами, но гораздо слабее их (рис. 1.13). Частота работы микроконтроллера: 8–32 МГц, размер оперативной памяти: 1–16 Кбайт, емкость постоянной памяти: 1–32 Кбайт.

Контроллер Arduino

Проект, который существенно упростил разработку несложных роботов, зародился в Италии в 2005 году и, благодаря открытой документации, распространился по всему миру. Его разработчики назвали свое устройство Arduino.

Arduino — это не только плата, содержащая микроконтроллер, стабилизаторы питания и удобно расположенные контактные площадки для подключения внешних устройств. В проект Arduino входит простая и оригинальная программная оболочка, позволяющая относительно легко создавать и модифицировать программы, а также огромное количество программных библиотек для подключения датчиков, моторов и других устройств.

Фактически, Arduino — это электронный конструктор. На рис. 1.14 представлен контроллер Arduino UNO. Номерами 0–13 обозначены цифровые порты, A0–A5 —

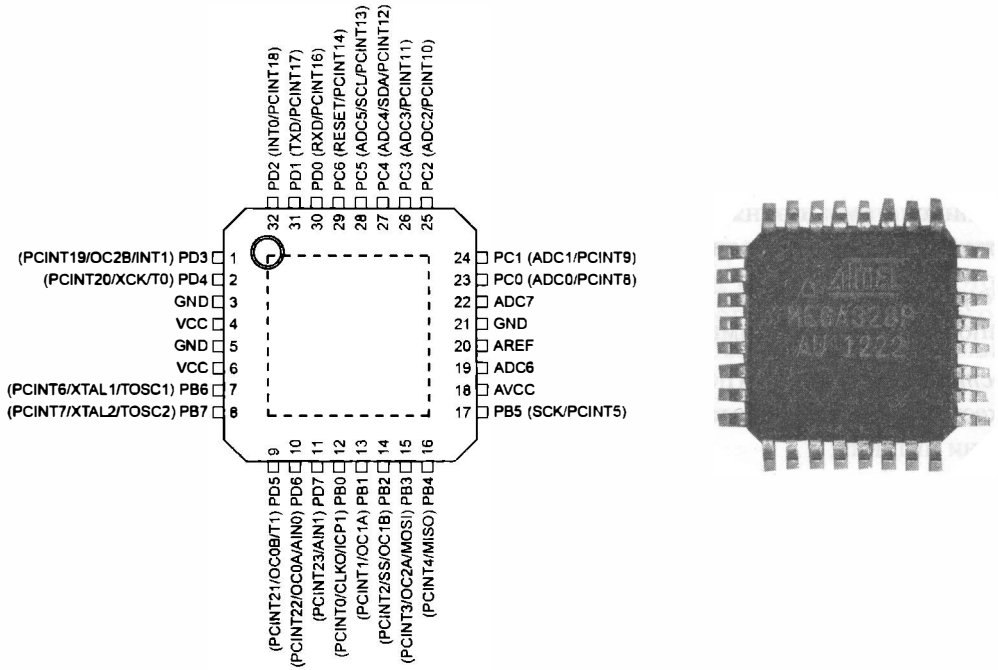


Рис. 1.13. Микроконтроллер ATmega328

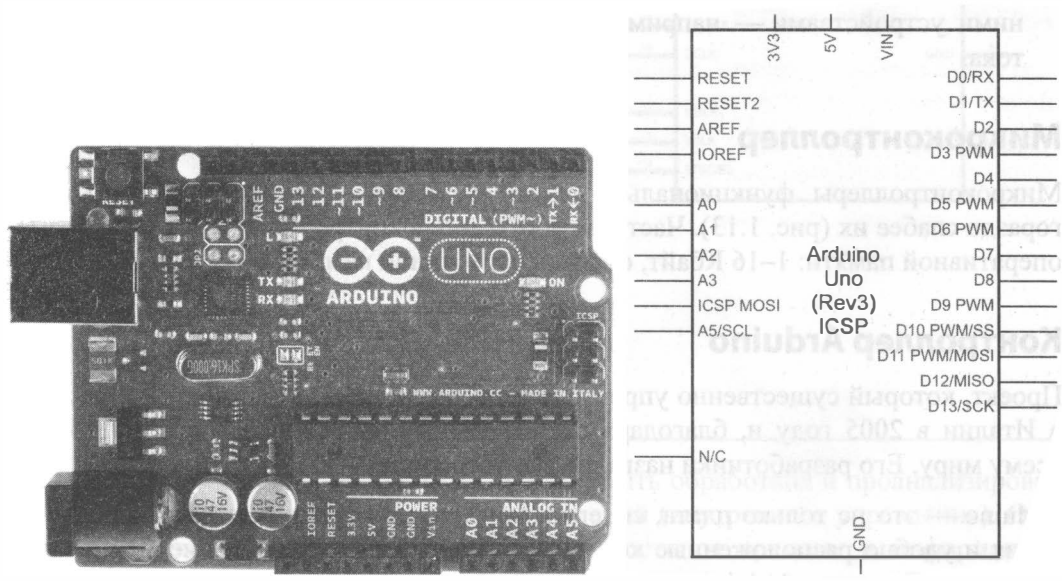


Рис. 1.14. Плата контроллера Arduino UNO на основе микроконтроллера ATmega328P

порты, которые могут анализировать уровень сигнала и преобразовывать его в числовую форму. На плату может быть подано напряжение от 6 до 12 вольт, при этом стабилизаторы питания преобразуют его в требуемые уровни.

Система связи

Когда требуется скорректировать работу робота либо получить дополнительную информацию о его состоянии, следует использовать системы связи.

Система связи служит для организации взаимодействия робота с другими устройствами и «хозяином». Вмешательство «хозяина» требуется, когда автономная (самостоятельная) работа невозможна или нецелесообразна. Так, робот может попасть в условия, правила действия в которых не описаны в его программе, или условия требуют принятия решения человеком, или робот жестко запрограммирован так, что любые действия совершаются только после получения команды от «хозяина».

Системы связи могут использовать следующие каналы:

- проводные каналы, когда робот соединен с пультом управления проводами;
- радиоканалы, когда информация передается радиоволнами;
- инфракрасные каналы, когда информация передается световыми импульсами;
- аудиоканалы, когда робот напрямую воспринимает речевые команды человека и сам может сообщить ему некоторую информацию;
- визуальные каналы, когда сообщения выводятся на различные информационные мониторы робота.

Инфракрасный приемник

Управляющие команды можно посылать на инфракрасный (IR) приемник, установленный на роботе (рис. 1.15). Это удобно, когда команд немного, и между вами и роботом нет преград. Так, например, можно управлять движением робота.

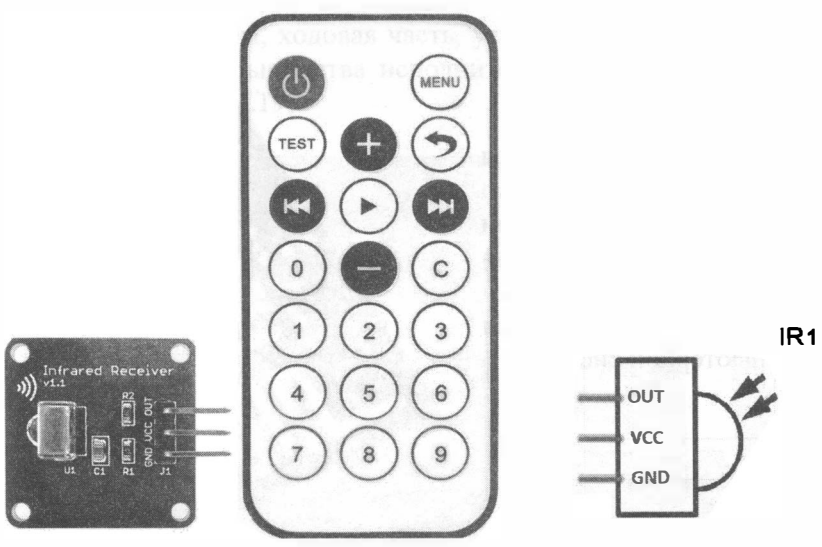


Рис. 1.15. IR-приемник на плате (*слева*); IR-пульт (*в центре*); схема выводов (*справа*)

Канал Bluetooth

Удобный способ общения с роботом предоставляет канал Bluetooth. Существует масса программ для управления роботами по этому каналу с помощью смартфонов на операционной системе Android. Модули Bluetooth для роботов (рис. 1.16) также не являются редкостью. В одной из практических глав мы рассмотрим этот вопрос подробнее.

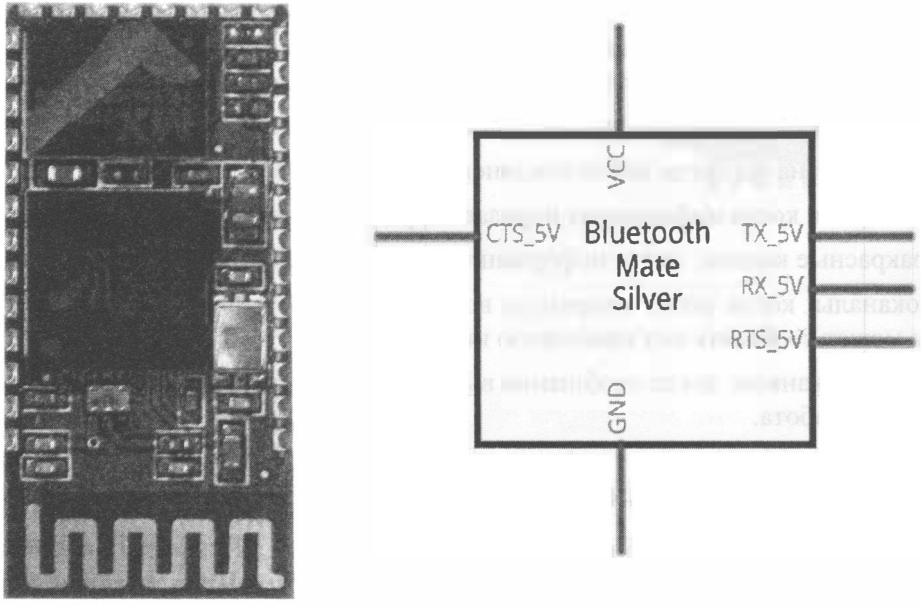


Рис. 1.16. Приемопередатчик Bluetooth

Канал Wi-Fi

Радиоканалы связи не ограничиваются только Bluetooth, существуют и широко применяются модули Wi-Fi (рис. 1.17), при помощи которых роботами можно управлять посредством беспроводных компьютерных сетей. Роботы при этом становятся частью большой компьютерной системы.

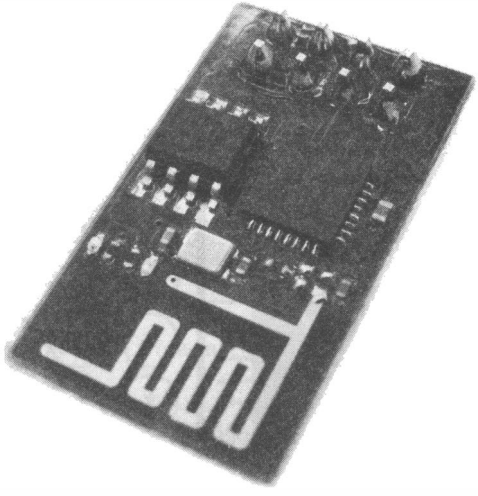


Рис. 1.17. Приемопередатчик Wi-Fi

Дисплей

К средствам связи можно также отнести и различные виды дисплеев для вывода информации. Основная проблема подключения обычных дисплеев к Arduino заключается в большом количестве задействованных портов ввода/вывода, но современные дисплеи (рис. 1.18), использующие I^2C-шину, этой проблемы лишены.

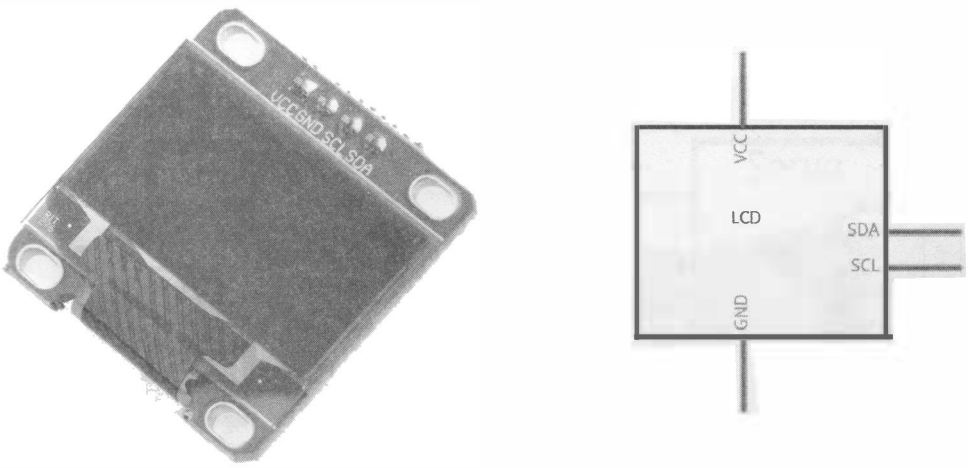


Рис. 1.18. Дисплей с управлением по шине I^2C

Исполнительная система

Исполнительная система отвечает за все движения робота и определяет возможности робота выполнять другие физические действия. К исполнительной системе относятся манипуляторы, ходовая часть, устройства позиционирования рабочих элементов. Основой большинства исполнительных механизмов являются электрические двигатели (рис. 1.19).

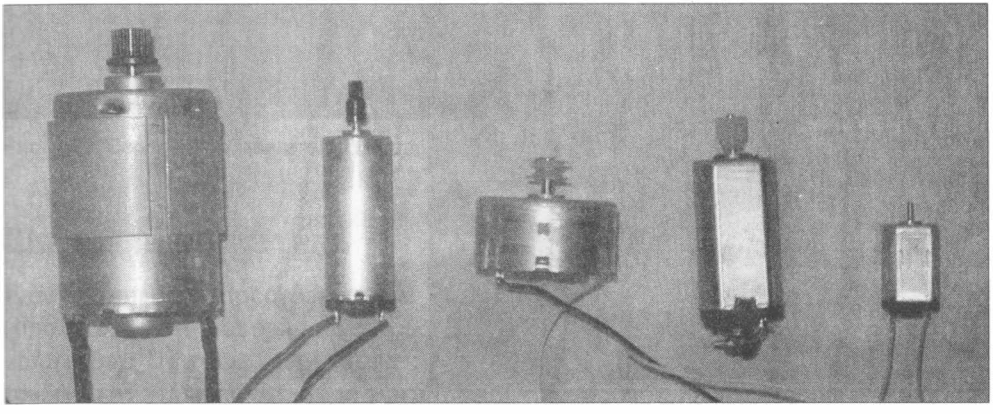


Рис. 1.19. Электрические двигатели

Электрический двигатель постоянного тока

Электрические двигатели различаются мощностью и количеством оборотов в минуту. Выбирать электрический двигатель нужно исходя из конкретной задачи. Если мощность мала, применяют редукторные передачи, которые снижают количество оборотов на выходном валу, но увеличивают мощность. На рис. 1.20 изображен двигатель постоянного тока с установленным на нем редуктором с передаточным числом 48.

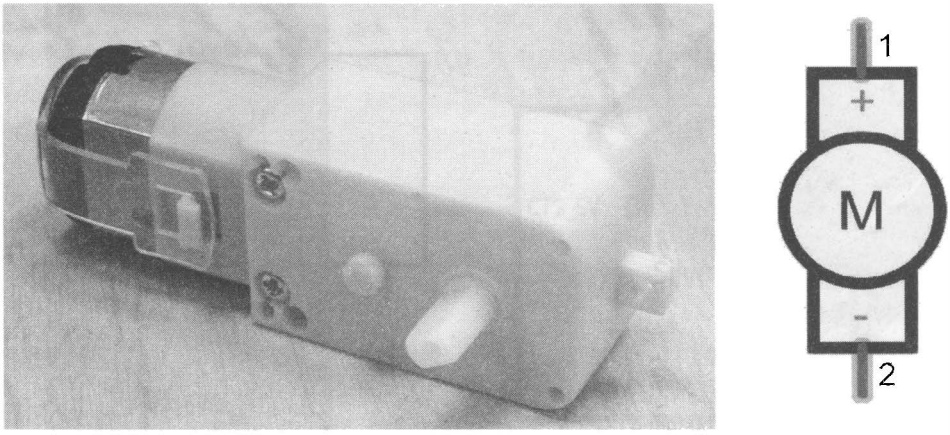


Рис. 1.20. Двигатель постоянного тока с редуктором

Редукторные передачи этого двигателя показаны на рис. 1.21. Вал двигателя вращается быстро, а колеса робота — в 48 раз медленнее, но в 48 раз мощнее. Это позволяет роботу преодолевать препятствия и двигаться в гору. Конечно, размер колеса также имеет значение.

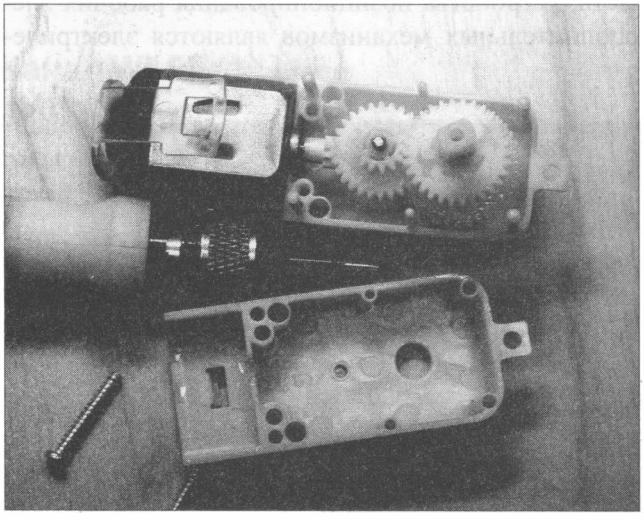


Рис. 1.21. Редуктор двигателя

Сервомотор

Другим распространенным видом привода для робота является сервомотор (рис. 1.22). Его отличие от двигателя постоянного тока с установленным редуктором в том, что основная задача сервомотора — обеспечить определенный угол поворота вала, который задается управляющим сигналом. Сервомоторы применяются в манипуляторах, поворотных механизмах, при создании шагающих роботов — везде, где требуется точность позиционирования угла поворота.

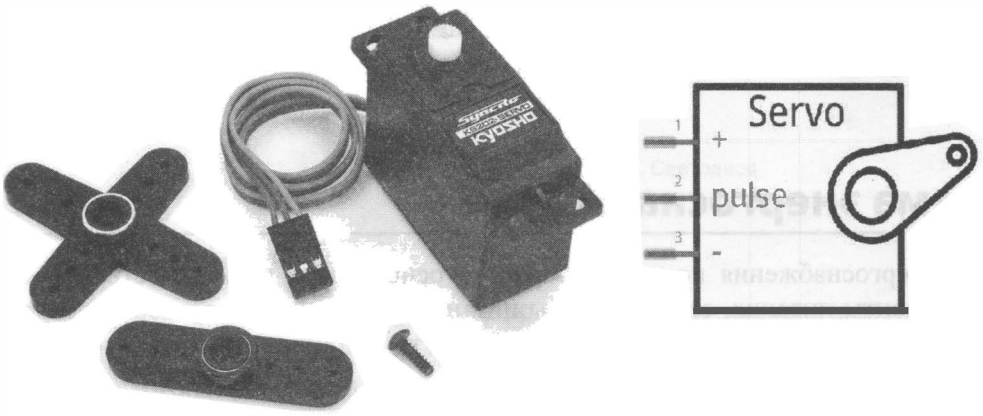


Рис. 1.22. Сервомотор

Внутри сервомотора, как правило, установлен двигатель постоянного тока и понижающий редуктор. Там же могут находиться схема управления и энкодер. На рис. 1.23 изображен разобранный сервомотор (сервопривод).

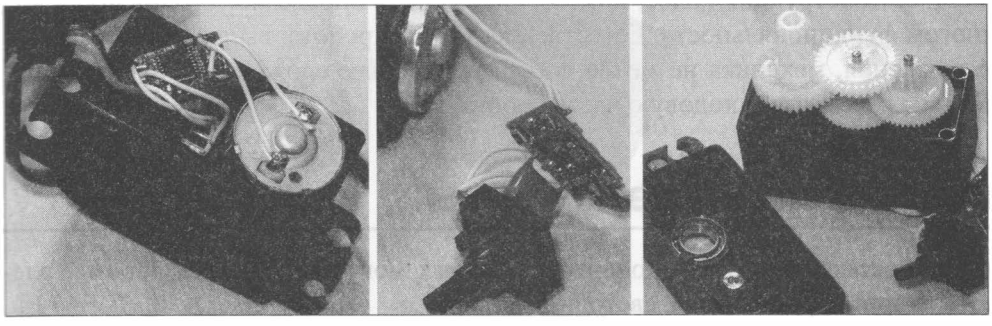


Рис. 1.23. Сервомотор: двигатель постоянного тока (*слева*); энкодер (*в центре*); редуктор (*справа*)

Шаговый двигатель

Еще одним распространенным видом приводов является шаговый двигатель (рис. 1.24). Он поворачивает свой вал небольшими дискретными значениями — «шагами». Шаговые двигатели широко используются в станках с числовым программным управлением для организации точного движения обрабатывающего инструмента.

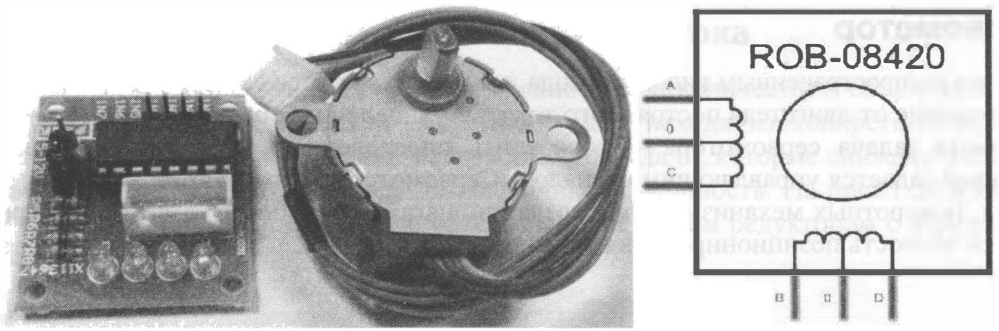


Рис. 1.24. Шаговый двигатель: плата управления (*слева*); собственно шаговый двигатель (*в центре*); схема (*справа*)

Система энергоснабжения

Система энергоснабжения несложного робота основана на электричестве. В нее входят элементы питания, стабилизаторы питания, различные генераторы электроэнергии — например, солнечные батареи. Электропитанию роботов посвящена *глава 3* книги.

Механика робота

Механика робота, ее скелет, представляет собой совокупность элементов конструкции, к которым крепятся и с которыми соединяются все рассмотренные ранее составляющие. Конструкция колес, ног, корпуса робота определяет его внешний вид и во многом функциональность. Естественно, что для робота, выполняющего физические движения, механика не менее, а нередко и более сложна, чем электроника. В *главе 6* мы рассмотрим ходовую часть робота.

Вспомогательные элементы

Кроме уже рассмотренных компонентов, в схемах могут встречаться простые элементы — например, резисторы, светодиоды и пр.

Резистор

Резистор (рис. 1.25) играет роль ограничителя тока и роль дополнительной нагрузки.

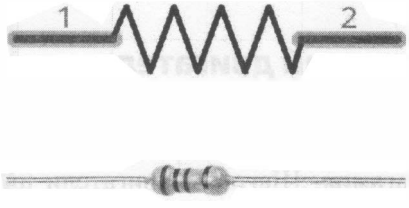


Рис. 1.25. Резистор

Светодиод

Светодиод (рис. 1.26) служит для подсветки объектов и индикации.

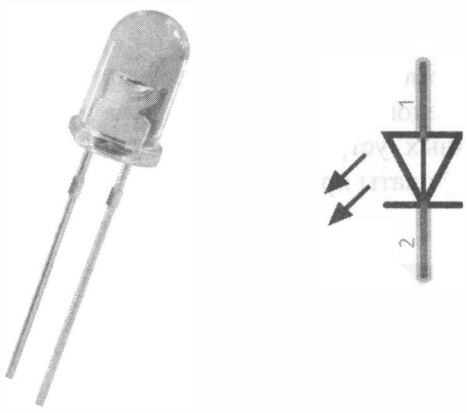


Рис. 1.26. Светодиод

Выключатель

Выключатель, или тумблер (рис. 1.27), служит для включения/выключения робота. Выключатели ставятся в цепь электропитания робота, разрывая соединение положительного контакта с элементом питания.

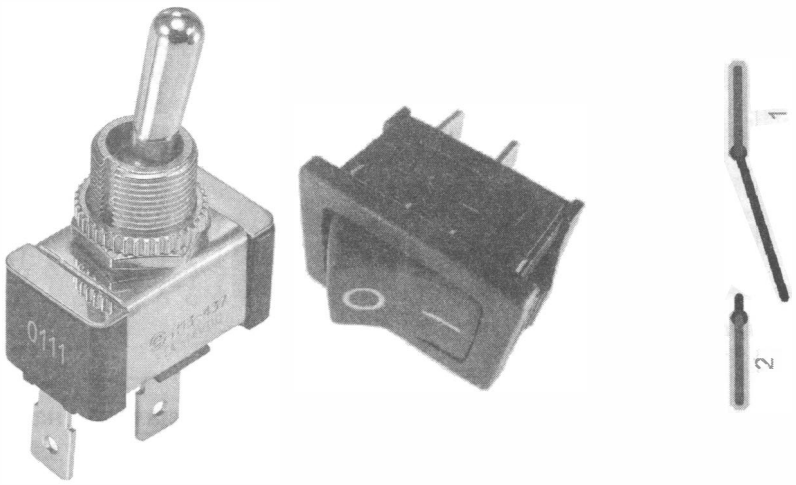


Рис. 1.27. Выключатель

Сервисные платы

Помимо основных элементов в состав робота могут входить части, роль которых заключается в создании удобного соединения электронных компонентов робота между собой. Примером подобной платы в arduino-проектах является Arduino Sensor Shield v5.0 (рис. 1.28). Эта плата устанавливается сверху в разъемы платы

Arduino UNO или Arduino MEGA и обеспечивает удобное подключение таких устройств, как: дисплей 128×64 с последовательным (SPI) или параллельным портом, сервомоторы, модуль Bluetooth, модуль беспроводной связи APC220, устройств с последовательным интерфейсом, устройств с интерфейсом I^2C. Списком перечисленных устройств возможности платы не ограничиваются (подробное описание этой платы приведено в *приложении 1*).

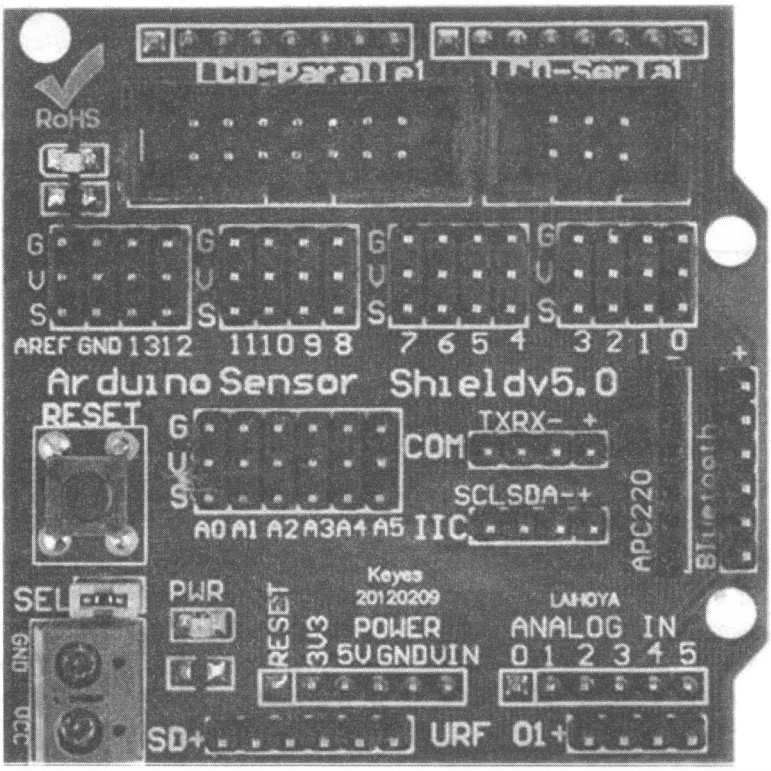
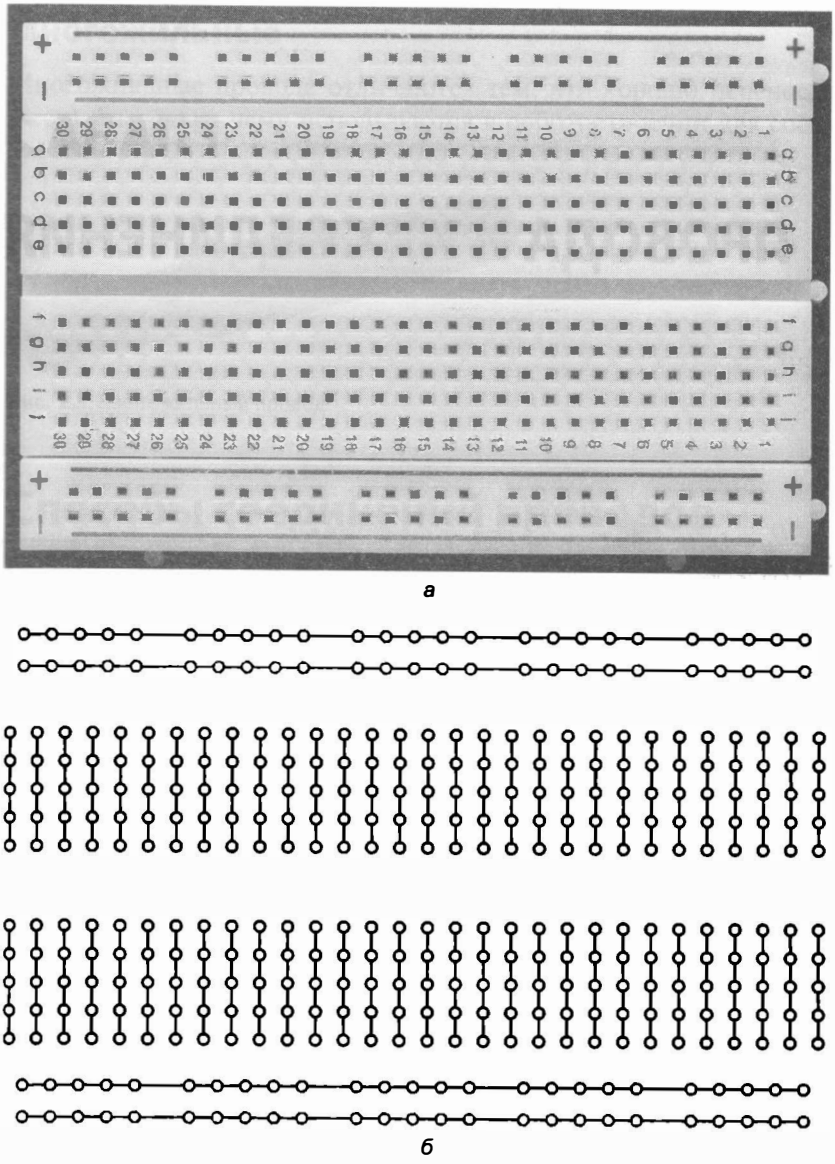


Рис. 1.28. Сервисная плата Arduino Sensor Shield v5.0

Плата не имеет активных элементов (транзисторов, стабилизаторов напряжения) в своем составе, но может использовать внешний источник питания для подключаемых устройств, что часто очень важно.

Удобным для временного монтажа может стать применение макетной платы (рис. 1.29, *а*). Такая плата предназначена для быстрого монтажа экспериментальных схем. Она позволяет, буквально на коленках, собрать электронное устройство, имеющее множество соединений. На рис. 1.29, *б* показано соединение контактов внутри этой платы.



а

б

Рис. 1.29. Макетная плата (*а*) и схема ее внутренних соединений (*б*)

Выводы

В краткой форме мы рассмотрели здесь основные системы робота: информационно-измерительную, принятия решений, связи, исполнительную, энергоснабжения, а также механику робота и вспомогательные элементы. Особенности работы с ними будут раскрыты более подробно в процессе выполнения соответствующих проектов. Следующая же глава посвящена проводам робота и их соединению.

ГЛАВА 2
ПРОВОДА И ИХ СОЕДИНЕНИЯ

Виды проводов

Электрические элементы робота соединяются друг с другом посредством проводников электрического тока. Если элементы расположены на специальной подложке — плате, то проводниками являются ее медные «дорожки». Если элементы разнесены, используют провода. Провода различают по типу металла, из которого они состоят, по толщине (сечению) и количеству жил. Для роботов используют медные провода.

Медные провода являются основой соединений в бытовой электронике из-за малого сопротивления электрическому току, хорошей теплопроводности и гибкости, а также возможности качественной пайки.

Одножильные

Одножильные провода (рис. 2.1) могут применяться в соединениях, которые не будут подвергаться смещениям относительно друг друга. Если такие провода перегибать, то они быстро ломаются и перестают работать.

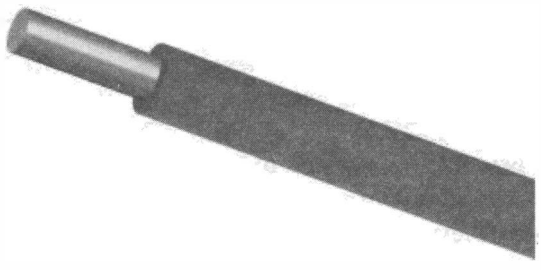


Рис. 2.1. Одножильный провод

Многожильные

Многожильные провода отличаются тем, что хорошо переносят изгибы (не ломаются), благодаря чему используются в роботостроении для соединения подвижных частей.

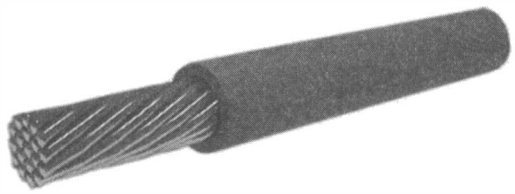


Рис. 2.2. Многожильный провод

Способы соединений проводов

Так как создание робота подразумевает самостоятельное соединение проводов, то разобраться, как это можно сделать качественно, будет не лишним. Рассмотрим основные способы электрических соединений, которые можно применять при конструировании роботов.

Скрутка

Соединение скруткой (рис. 2.3) применяется тогда, когда требуется создать контакт временно, или условия, в которых вы работаете, не позволяют применить другой

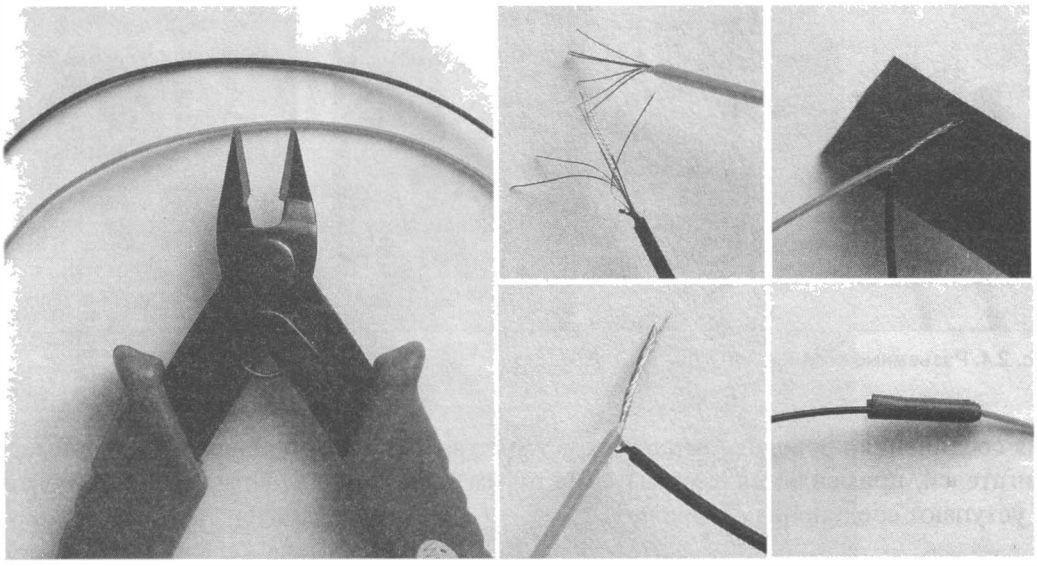


Рис. 2.3. Этапы соединения скруткой: провода и бокорезы (*слева*); зачищенные и распушенные провода (*в центре сверху*); скрутка (*в центре внизу*); изолирование скрутки (*справа вверху*); изолированная скрутка (*справа внизу*)

тип соединений. Для соединения скруткой потребуются бокорезы или нож. Без нажима зажимаем бокорезами изоляцию проводов в месте, по которое нужно снять изоляцию, и дергаем. Если все сделано правильно, изоляция оторвется, а жилы проводов останутся нетронутыми. Распушим жилы и скрутим провода друг с другом. Место скрутки обязательно нужно заизолировать. Следует помнить, что медь очень быстро окисляется, покрываясь непроводящей электрический ток пленкой, что через некоторое время сделает соединение неработоспособным. При первой возможности соединение скруткой нужно пропаять.

Разъемные соединения

Разъемные соединения широко применяются при быстром монтаже радиодеталей, но провода с разъемными клеммами высокого качества стоят дорого, а дешевые не лишены недостатков, что, впрочем, компенсируется скоростью монтажа. Недостатки разъемных соединений заключаются в окислении контактов со временем и потере соединения, а также в разрыве соединения при небольшом случайном усилии. Платы Arduino содержат или комплектуются клеммными разъемами, подобными разъемами комплектуется и большинство дополнительных электронных модулей (датчики, драйверы двигателей). Эти типы клемм называются *Dupont* (рис. 2.4).

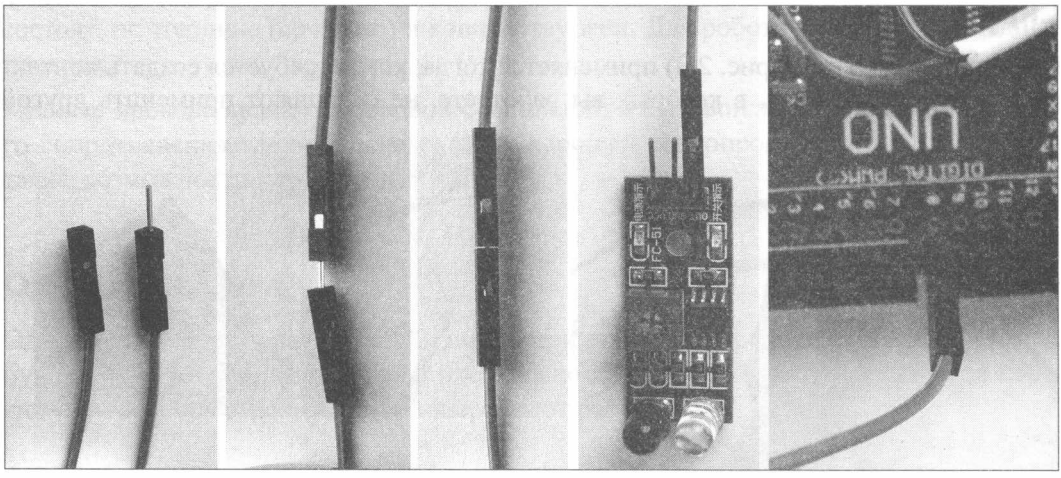


Рис. 2.4. Разъемные соединения

Для соединения проводов, питающих устройства большой мощности, — например, двигателей, применяются клеммы с болтовым зажимом, подобные разъемы почти не уступают соединениям пайкой.

Часто разъемные соединения применяют совместно с платами расширения (рис. 2.5) или с макетной платой (рис. 2.6). Это позволяет быстро создать прототип и проверить его работоспособность, что очень важно при конструировании.

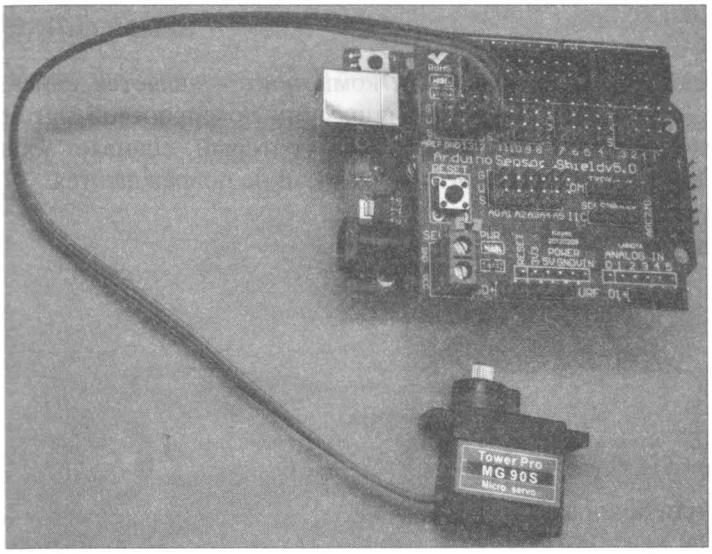
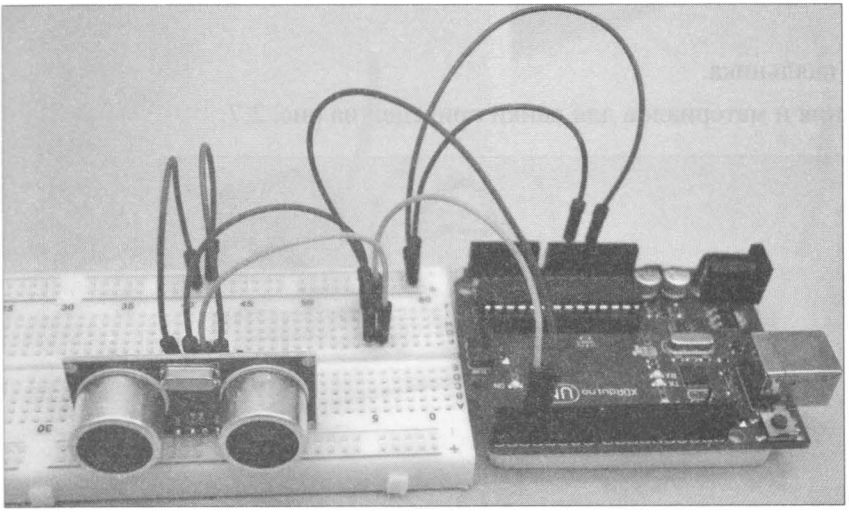
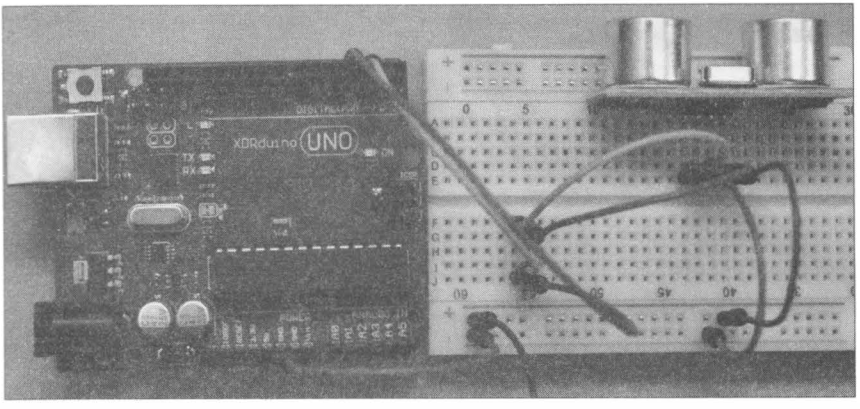


Рис. 2.5. Сервомотор соединен с платой расширения при помощи разъема



а



б

Рис. 2.6. Использование макетной платы для сборки прототипа: *а* — общий вид; *б* — вид сверху

Пайка и ее основы

Наиболее качественным соединением электронных компонентов является пайка. *Пайкой* называется процесс соединения двух металлических поверхностей путем введения между ними низкотемпературного расплава, который спаивает обе поверхности между собой. При этом спаиваемые поверхности не повреждаются.

Оборудование и материалы

Нам потребуются:

- паяльник;
- припой;
- паяльный флюс или канифоль.

Для комфортной работы еще будут нужны:

- острый перочинный нож или бокорезы;
- стальная губка для быстрой очистки жала паяльника от гари;
- пинцет;
- подставка для паяльника.

Набор оборудования и материалов для пайки приведен на рис. 2.7.

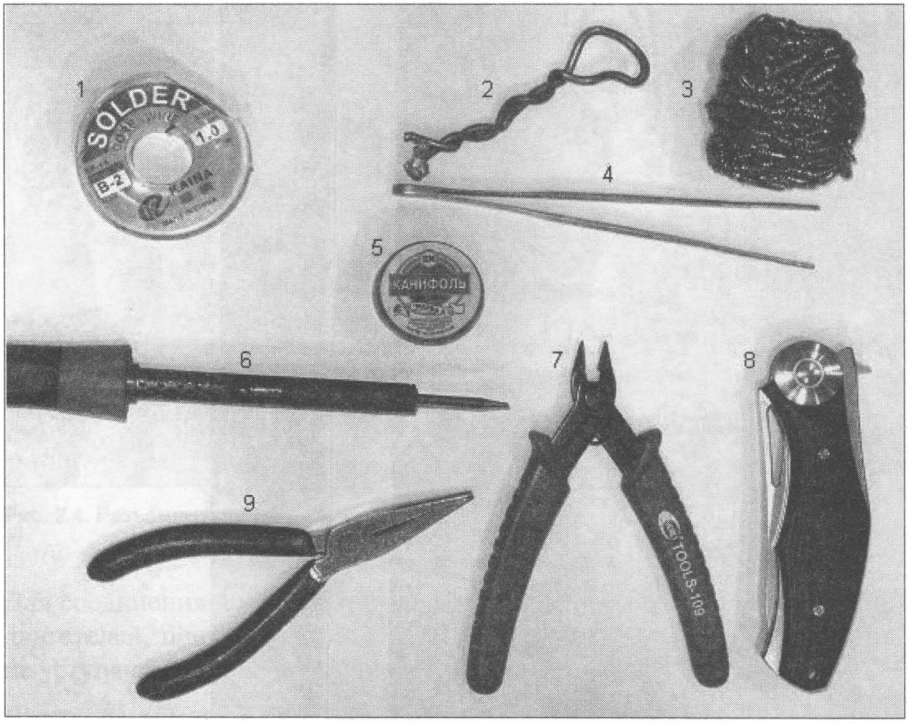


Рис. 2.7. Принадлежности для пайки: катушка с припоем (1); проволочный припой с сердцевиной из канифоли (2); губка для очистки жала паяльника (3); пинцет (4); канифоль (5); паяльник (6); бокорезы (7); острый нож (8); плоскогубцы (9)

Этапы пайки

Пайка проводов (рис. 2.8) состоит из следующих этапов:

1. Зачищаем бокорезами кончики проводов от изоляции. Для многожильного провода скручиваем жилы в одну косичку.

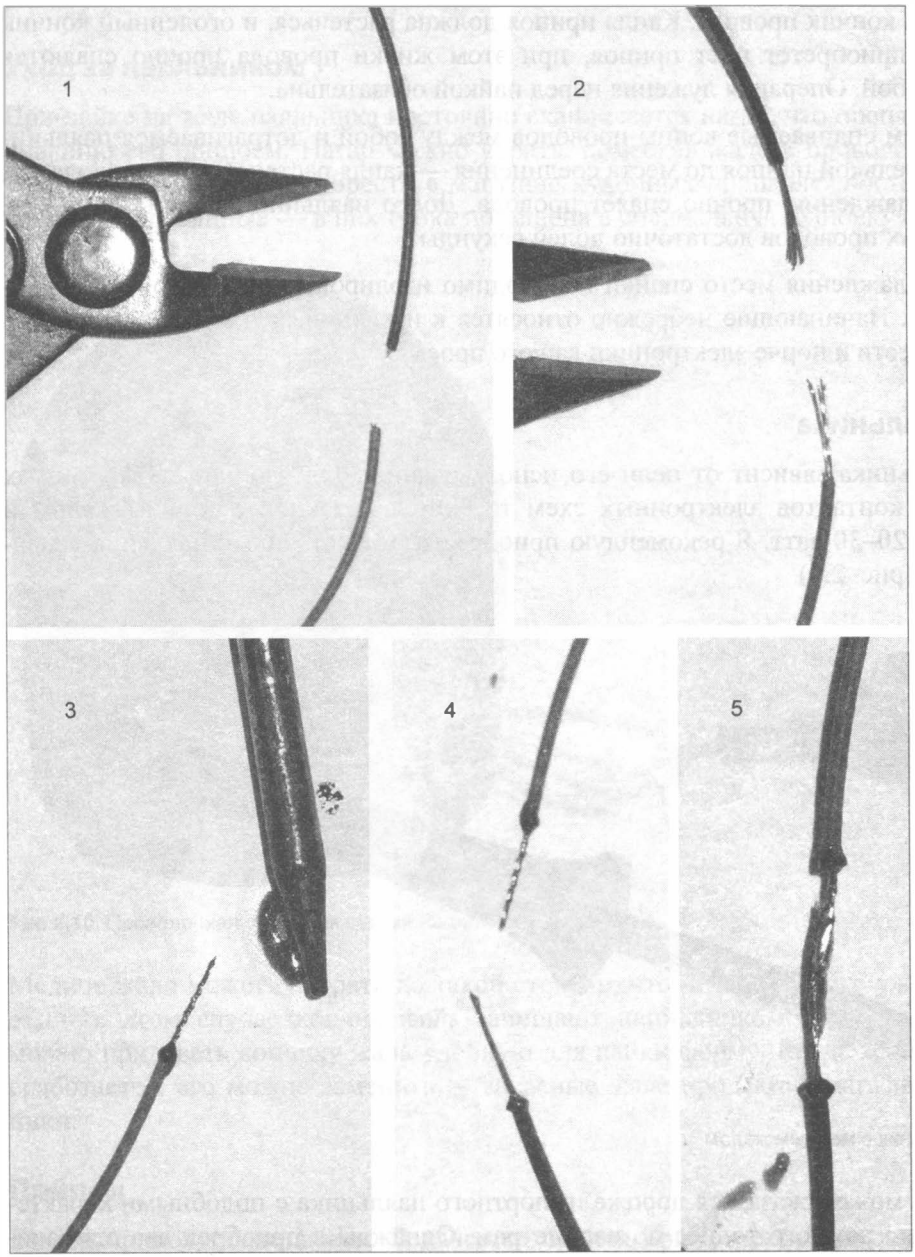


Рис. 2.8. Процесс пайки: обрезка края проводов (1); зачистка (2); лужение (3); готовые к пайке концы (4); спаянные провода (5)

2. Лужение провода. Если попробовать спаять нелуженые провода, то, скорее всего, пайка будет некачественной, потому что на поверхности медных жил находится окись, препятствующая прилипанию припоя. Для лужения помещаем кончик жала паяльника в канифоль и, пока расплавленная канифоль не испарилась с жала, переносим ее на зачищенный кончик провода. Теперь повторяем ту же операцию с припоем: берем на кончик паяльника каплю припоя, а затем переносим ее на кончик провода. Капля припоя должна растечься, и оголенный кончик провода приобретет цвет припоя, при этом жилки провода прочно спаяются между собой. Операция лужения перед пайкой обязательна.
3. Соединяем спаиваемые концы проводов между собой и дотрагиваемся паяльником с капелькой припоя до места соединения — капля растечется по проводам и, после охлаждения, прочно спаяет провода. Долго паяльник держать не нужно, для тонких проводов достаточно долей секунды.
4. После охлаждения место спайки необходимо изолировать изолентой или трубкой ПВХ. Начинающие небрежно относятся к изоляции, что в дальнейшем может привести к порче электроники вашего проекта.

Выбор паяльника

Выбор паяльника зависит от цели его использования. Для удобной пайки тонких проводов и контактов электронных схем требуется паяльник с тонким жалом и мощностью 20–30 ватт. Я рекомендую приобрести отечественный паяльник с медным жалом (рис. 2.9).

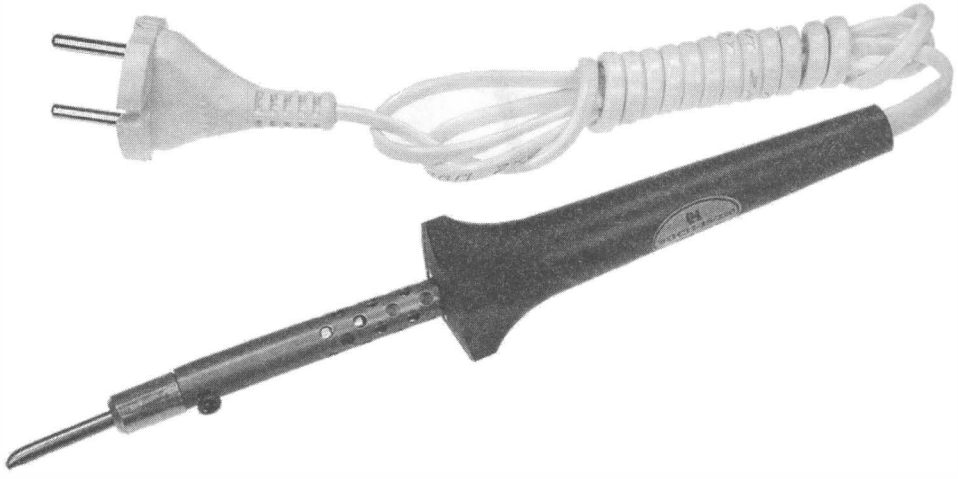


Рис. 2.9. Паяльник с медным жалом

Конечно, он может оказаться дороже импортного паяльника с подобными характеристиками, но намного точнее по параметрам. Однажды я приобрел два одинаковых импортных 20-ваттных паяльника, но один из них имел мощность, соответствующую номиналу, а другой постоянно перегревался, и им невозможно было паять.

Впрочем, высококачественные паяльники зарубежного производства с регулировкой температуры, не обгорающими жалами и набором сменных жал разного вида стоят достаточно дорого.

Если вы планируете паять толстые провода, то потребуется паяльник мощностью 40–60 ватт, имейте только в виду, что тонкие провода и микросхемы паять им неудобно.

Уход за паяльником

При пайке на жале паяльника постоянно скапливается нагар, что препятствует смачиванию его припоем. Нагар можно убрать, поместив жало в проволочную губку. Такую губку можно приобрести в магазине кухонных принадлежностей, но есть и специализированные — в них губка помещена в специальную баночку (рис. 2.10).

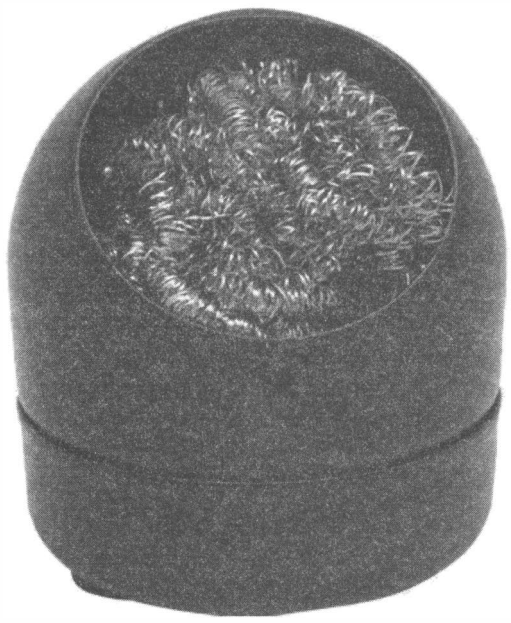


Рис 2.10. Проволочная губка для снятия нагара

Медное жало может обгорать до такой степени, что чистка в губке уже не помогает, — в этом случае кончик жала зачищают напильником. Напильником также можно придавать кончику жала удобную для пайки форму. Когда жало полностью сработается, его можно заменить — запасные жала продают в магазинах электроники.

Припои

Выбор припоя также важен. В качестве припоя применяется легкоплавкий сплав металлов, температура плавления которого может колебаться от 200 до 350 градусов. При монтаже электроники используются оловянно-свинцовые припои. Соглас-

но отечественной маркировке, оловянно-свинцовый пропой обозначается буквами ПОС, после которых идет число, обозначающее процентное содержание олова. Температуры плавления различных оловянно-свинцовых припоев в градусах Цельсия: ПОС30 — 240°, ПОС40 — 210°, ПОС61 — 180°, ПОС90 — 310°.

Проволочный припой может содержать внутри себя сердцевину из канифоли для лучшего качества спайки.

Флюсы

Флюс служит для удаления слоя окисла со спаиваемых поверхностей, предохраняет металл от окисления при пайке, а также способствует равномерному растеканию припоя. Флюсы делятся на активные и неактивные. Активные флюсы содержат активные кислоты и применяются для пайки различных металлов, а не только меди. Остатки активных флюсов должны обязательно удаляться после пайки смывкой, иначе они начинают разъедать место пайки. Неактивные флюсы в удалении не нуждаются, но справляются далеко не со всеми окислами металлов.

К неактивным флюсам относится канифоль и ее растворы. Для большинства электронных контактов она подойдет — смачиваем контакт канифолью, а затем паяем припоем.

Но в случаях, когда спаять требуется сталь или алюминий, канифоль бесполезна. Так, если надо припаять провод к контакту аккумулятора, без активного флюса не обойтись. Активные флюсы (рис. 2.11) бывают жидкими и желеобразными. Желеобразными флюсами пользоваться удобнее — они не стекают после нанесения на поверхность. Самым распространенным активным паяльным флюсом является паяльная кислота, и ее обязательно нужно смывать после пайки. Хорошо зарекомендовал себя флюс для алюминия, с ним можно паять почти все металлы.

Рис. 2.11. Активные флюсы

Выводы

Мы рассмотрели здесь виды проводов и соединений, а также — подробно — этапы и особенности пайки, ведь пайка является наиболее качественным соединением для электрических проводов. Если вопросы все же остались, можно обратиться к YouTube — там есть несколько каналов, которые очень доходчиво объясняют и показывают, как паять правильно.

Следующая глава будет посвящена электрическому питанию роботов, выбору источников электропитания и их особенностям.

ГЛАВА 3
ЭЛЕКТРОПИТАНИЕ

Прежде чем начать обсуждать источники электрического питания, которые можно использовать в роботах, следует вспомнить немного теории.

Закон Ома

Электрический ток измеряется в амперах (А), и в формулах он обозначается буквой *I*. Электрическое напряжение измеряется в вольтах (В), его принято обозначать буквой *U*. Сопротивление измеряется в омах (Ом) и обозначается буквой *R*.

Связь между ними записывается в виде формулы (закона) Ома:

$$I = \frac{U}{R}$$

Электрическая мощность

При обсуждении источников электрического тока будет важна еще одна характеристика — электрическая мощность:

$$P = U \cdot I$$

Мощность измеряется в ваттах (Вт) и показывает работу, выполненную за 1 секунду электрическим током. Для нас будет важна потребляемая роботом мощность. На основании значений потребляемой мощности должен подбираться источник электрического питания.

Характеристики элементов питания

Номинальное напряжение

Номинальное напряжение — напряжение, которое будет на клеммах элемента питания при работе его в нормальных условиях.

Номинальный ток

Номинальный ток — ток, при котором элемент питания будет работать в соответствии с указанными в документации на него параметрами.

Емкость

Для аккумуляторной батареи важна ее емкость, измеряемая в ампер-часах. Обозначают ее на аккумуляторах в миллиампер-часах или mAh.

Двигатели, которые будут применяться в рассматриваемых моделях роботов, требуют напряжения питания от 4 до 7 В и потребляют ток 0,1 А. Максимальный ток при четырех одновременно работающих моторах будет тогда 0,4 А. Электронный контроллер Arduino потребляет порядка 0,05 А. Так что при питании от источника 7 вольт будет потребляться мощность около 3 Вт, а общий ток достигнет 0,45 А. Если емкость используемой батареи 2 ампер-часа (2000 mAh), то робот проработает около 4,5 часа без остановки. Когда робот прекращает движение, потребление им энергии снижается до потребления только контроллером.

Форм-фактор

Самыми распространенными типоразмерами элементов питания являются стандарты AA и AAA с размерами 14,5×50×5 мм и 10,5×44,5 мм соответственно. С появлением литий-ионных аккумуляторов стал распространяться типоразмер 18650 (18×66,5 мм). Указанные элементы питания представляют собой цилиндры с полюсами на концах. Существуют и другие форм-факторы элементов питания, но при построении несложных роботов с использованием контроллеров Arduino чаще всего используются батареи AA и 18650.

Типы элементов электрического питания

Элементы питания можно поделить на перезаряжаемые и не перезаряжаемые. Не перезаряжаемые используются от начала эксплуатации до окончания нормальной работы (обеспечение номинального напряжения и тока), после чего утилизируются. Перезаряжаемые элементы принято называть *аккумуляторами*, они заряжаются специальными зарядными устройствами, накапливают электрический заряд, затем используются, а после использования и разряда снова заряжаются. Так как мотор-

ные роботы потребляют относительно много электрической энергии, использование в них аккумуляторных батарей является предпочтительным.

Солевые батареи

Солевые батареи — это самые недорогие не перезаряжаемые источники питания. Они могут быть использованы только в тех роботах, где отсутствуют мощные потребители электроэнергии, — такие как двигатели, сильное освещение, работа по радиоканалу. Они имеют малую емкость и нестабильное напряжение. Номинальное напряжение для формата AA — 1,5 В.

Алкалиновые батареи

Алкалиновые батареи (также называются *щелочными*) имеют увеличенный по сравнению с солевыми срок службы и хранения. Маркируются надписью «Alkaline». Могут быть использованы в роботах с мощными потребителями электроэнергии. Номинальное напряжение для формата AA — 1,5 В.

Никель-металлогидридные аккумуляторы

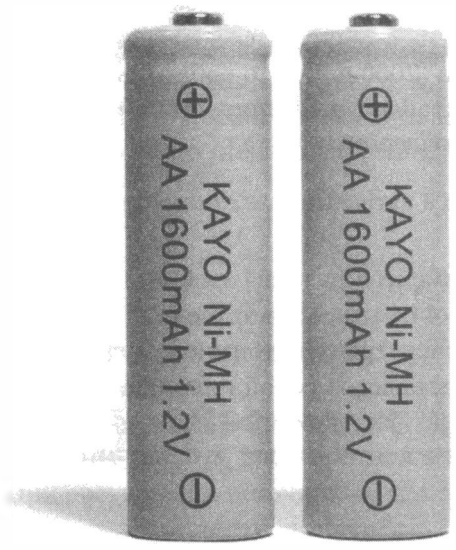


Рис. 3.1. Никель-металлогидридные аккумуляторы формата AA

Самым распространенным типом аккумуляторов являются никель-металлогидридные (рис. 3.1), их номинальное напряжение — 1,2 В (формат AA), и емкость от 900 до 3000 mAh. Для электропитания робота на основе платы Arduino потребуется шесть подобных аккумуляторов, соединенных последовательно, что даст на выходе напряжение 7,2 В. Этого будет достаточно как для питания платы Arduino (6–12 В), так и для моторной части робота. Следует учитывать, что при последовательном соединении емкости батарей не складываются, поэтому для продления их жизни следует применять совместно аккумуляторы с одинаковой емкостью.

Литий-ионные аккумуляторы

Если в моменты пиковой нагрузки робот останавливается и перезагружается, значит, были выбраны элементы питания недостаточной мощности, которые не в состоянии выдавать требуемый роботу ток. При этом напряжение питания падает, что

и приводит к отключению электроники. Этого можно избежать, если параллельно примененному источнику питания подключить другой, что увеличит максимальную отдаваемую мощность. Можно также запитать электронику робота от отдельного источника, не забыв в этом случае объединить отрицательные полюса обоих источников питания.

Еще одним выходом из подобной ситуации является использование более мощных батарей — например, литий-ионных (рис. 3.2). Для них стандартным номинальным напряжением является значение 3,7 В. Производятся литий-ионные аккумуляторы как в виде круглых батарей формата 18650, так и в виде пластин, широко применяемых в планшетах и сотовых телефонах. Емкость аккумуляторов формата 18650 составляет 1500–4000 mAh.

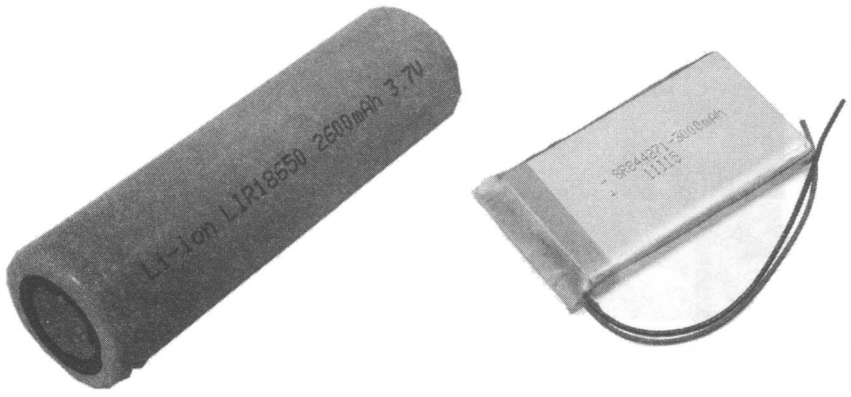


Рис. 3.2. Литий-ионные аккумуляторы: *слева* — цилиндрический; *справа* — плоский

Стабилизация электропитания

Важным моментом в электротехнике является наличие прочного контакта в цепи источников питания — особенно это касается подвижных роботов, рывки и толчки которых могут приводить к разрыву питающей цепи. Для того чтобы разрывов питающей цепи не происходило, следует использовать подпружиненные контакты и специальные боксы с подобными контактами (рис. 3.3).

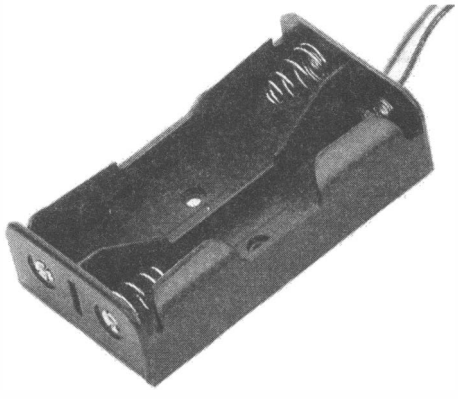


Рис. 3.3. Бокс для элементов питания

Электронные компоненты — такие как контроллеры, датчики, приемопередающие устройства — очень чувствительны к скачкам напряжения в цепи электропитания. Аккумуляторы хоть и промаркированы номинальным напряжением, но реальное напряжение на них колеблется в зависимо-

сти от уровня заряда в широких пределах. Решением вопроса стабилизации электрического питания занимаются специальные приборы — стабилизаторы питания.

Стабилизация напряжения

Самым доступным понижающим стабилизатором напряжения питания является микросхема КР142ЕН5А или ее аналог L7805CV. Схема подключения L7805CV показана на рис. 3.4: на вход поступает нестабилизированное напряжение 7 В или выше, а на выходе получаем стабильное постоянное напряжение 5 В.

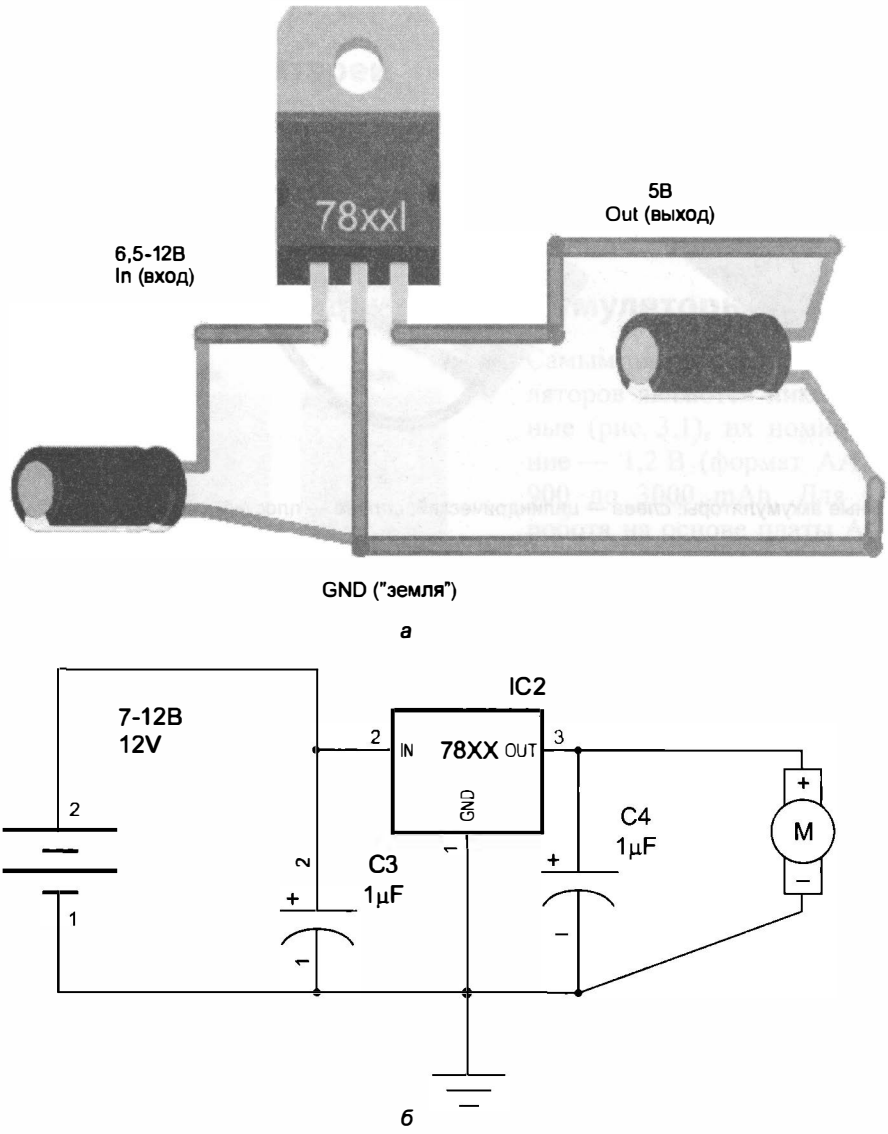


Рис 3.4. Стабилизация питания при помощи микросхемы L7805CV:
а — схема соединений; *б* — электрическая схема

Существуют подобные стабилизаторы и на другое напряжение, а также настраиваемые стабилизаторы. Платы Arduino UNO и Nano включают в себя, как минимум, один стабилизатор напряжения на 5 В и на 3,3 В.

Широкое применение получили портативные импульсные стабилизаторы. Они имеют высокий коэффициент полезного действия, достигающий 95%.

В последнее время стали популярны импульсные понижающие стабилизаторы китайского производства с настройкой выходного напряжения на основе микросхемы LM2596 (рис. 3.5).

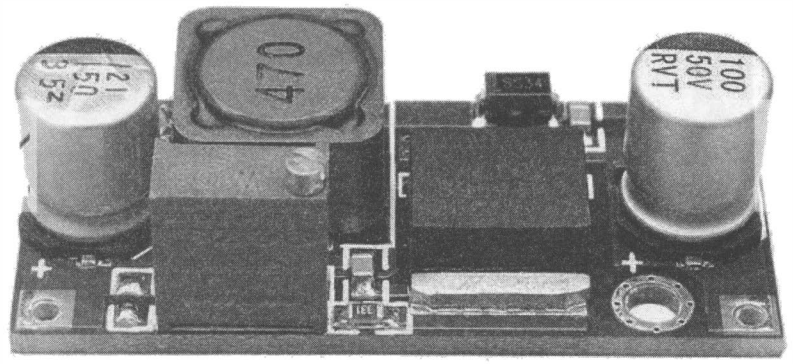


Рис. 3.5. Понижающий импульсный стабилизатор на основе микросхемы LM2596

Существуют также повышающие импульсные стабилизаторы. Они могут, имея на входе 3,7 В, преобразовать это напряжение в нужные для питания электроники робота повышенные значения. Визуально они мало отличаются от понижающих стабилизаторов напряжения, но работают на микросхеме XL6009 (рис. 3.6).

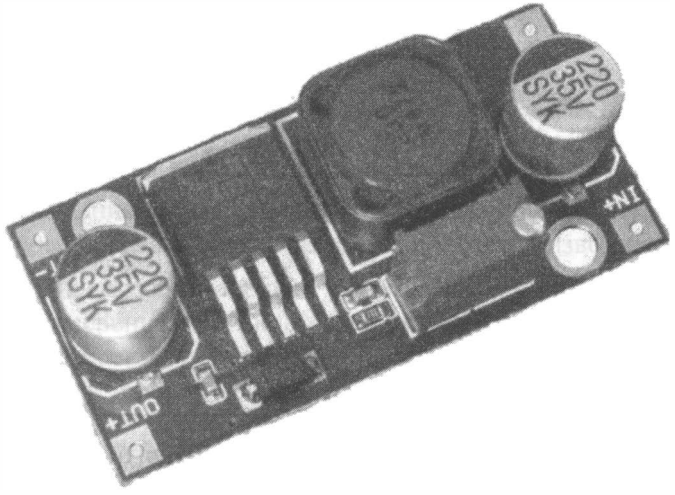


Рис. 3.6. Повышающий импульсный стабилизатор на микросхеме XL6009

Хорошо зарекомендовал себя в работе повышающий импульсный стабилизатор со входным напряжением от 2 В и стабилизированным выходным напряжением 5,1–5,2 В (рис. 3.7). Такие стабилизаторы удобны, если требуется сэкономить на массе элементов питания и размерах создаваемого прибора, применяя низковольтную батарею, — например, один литий-ионный аккумулятор на 3,7 В. Таким образом, если для электропитания нашей схемы необходимо 5 В и небольшой ток (не более 0,5 А), можно получить довольно компактное устройство.

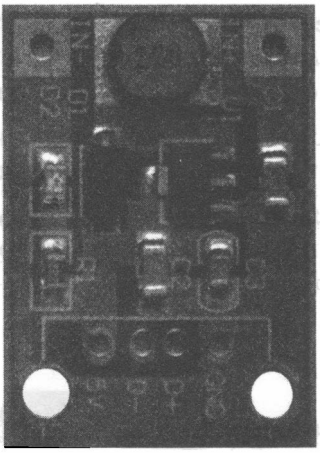


Рис. 3.7. Повышающий импульсный стабилизатор на заданное напряжение 5,2 В

Стабилизация электрического тока

В некоторых случаях может потребоваться получить не стабильное напряжение, а стабильный ток, — например, при подключении светодиодов. В этом случае используются стабилизаторы тока. Стабилизатор тока несложно построить на микросхеме LM317T (рис. 3.8). Значение стабилизируемого тока зависит от величины сопротивления R1.

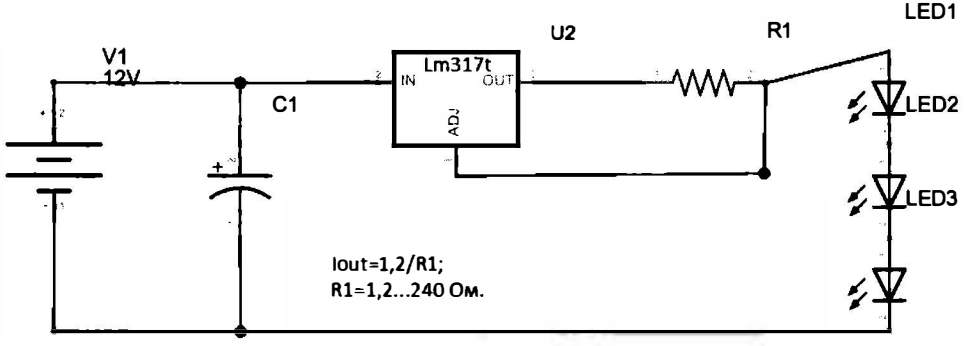


Рис. 3.8. Источник тока на базе LM317T

Измерение электрического тока, напряжения и сопротивления

Для того чтобы точно знать параметры вашего источника электрического питания, следует уметь производить измерения. Для этого применяются специальные приборы: амперметр измеряет величину тока, вольтметр — напряжение, омметр слу-

жит для измерения сопротивления электрической цепи и ее элементов. Существуют приборы, которые могут измерять все указанные величины, — такие приборы называются *мультиметрами* (рис. 3.9).

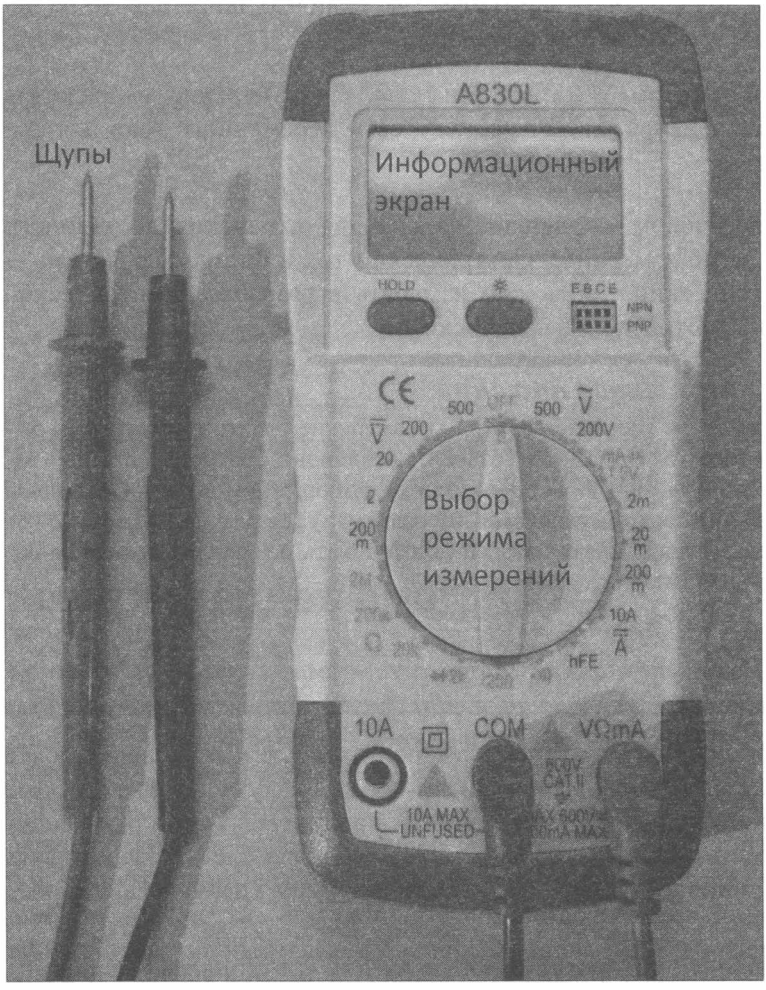


Рис. 3.9. Мультиметр

Измерения осуществляются при помощи щупов, которыми шунтируется место проведения замеров. Вращением ручки производится выбор режима измерения.

- Шкала, обозначенная значком **Ω**, применяется для измерения сопротивления, измерения производятся в величинах, кратных обозначениям на шкале. Перед измерением сопротивления требуется обязательно обесточить измеряемый элемент, иначе мультиметр может быть испорчен.
- Если у мультиметра есть выбор режима  , то возможно тестирование наличия соединения между участками цепи, — это полезно, когда нужно установить место разрыва.

- Шкала, обозначенная значком  , применяется для измерения напряжения постоянного тока. Может быть измерено напряжение питающей батареи, напряжение на различных элементах собранной схемы. Чтобы измерить напряжение для элементов схемы, они должны быть запитаны. Так, если это двигатель, то на него должно быть подано электрическое питание.
- Шкала, обозначенная значком  , предназначена для измерения напряжения переменного тока. Такое напряжение имеется в сети переменного тока для бытовых электроприборов.
- Шкала, обозначенная значком  , предназначена для измерения постоянного тока. Ни в коем случае не измеряйте электрический ток в бытовых розетках — это ОПАСНО! Электрический ток можно измерять в разрывах электрической цепи, чтобы определить потребление вашей схемы.

> ***Измерение тока элементов питания***
>
> Возможно кратковременное (не более секунды) измерение электрического тока химических элементов питания (литиевых батарей, алкалиновых батареек) с целью замера их остаточного заряда, при этом производится выбор режима **10A**, а активный щуп (обычно красный) перекоммутируется из разъема **VΩmA** в разъем **10A**. Ток нормально заряженных алкалиновых батарей должен быть не менее 1 ампера, а литиевых аккумуляторов 18650 — не менее 2 ампер.

Выводы

Мы рассмотрели здесь различные источники электропитания, в том числе наиболее эффективные никель-металлогидридные или литий-ионные аккумуляторы, которые и рекомендуется использовать для питания роботов. Рассмотрен также ряд наиболее распространенных схем стабилизаторов питания — от них следует запитывать контроллеры, датчики, приемопередающие устройства и другие чувствительные к питанию устройства.

Далее будут рассмотрены основы программирования для контроллера Arduino, на котором строится система принятия решений робота, его «компьютер».

ГЛАВА 4

ОСНОВЫ ПРОГРАММИРОВАНИЯ ARDUINO

Для того чтобы робот начал решать какую-либо задачу, собрать его недостаточно, потребуется создать и поместить в него специальную программу (план действий). Подобные программы называются *компьютерными.*

Компьютерная программа

Компьютерная программа — это четко формализованный план, состоящий из команд для контроллера (системы принятия решений). Контроллер поочередно читает команды и исполняет их. Стоит указать, что команды внутри любого цифрового устройства, коим и является контроллер Arduino, закодированы нулями и единицами, которые называются *двоичным представлением чисел*, т. е. вся информация перед поступлением в контроллер перекодируется из привычной для нас десятичной системы счисления в двоичную. Таким образом, при выполнении логических или арифметических операций контроллер сравнивает, делит, вычитает, выполняет прочие действия именно над двоичными числами. Эти числа хранятся в последовательных ячейках памяти, имеющих определенные адреса.

При поступлении на контроллер Arduino электрического питания автоматически начинается выполнение той программы, которая была в него загружена, если же программа отсутствует или написана некорректно, то происходит сбой, который либо останавливает выполнение команд, либо приводит к *зависанию* программы (переходу ее в бесконечный цикл). Номер выполняемой команды хранится в специальной ячейке памяти, которая называется *счетчиком команд*. Этот номер изменяется на следующий при выполнении арифметических операций, но может измениться и на любой адрес, если выполнялась логическая команда, результатом которой стал переход на некоторый пункт плана, отличный от следующего по порядку.

Алгоритм

Для визуального представления компьютерных программ используют их графическую запись, называемую *блок-схемой алгоритма*. Алгоритм, не имеющий логических блоков и выполняемый строго по пунктам от начала и до конца, называется *линейным* (рис. 4.1). Если алгоритм после достижения последнего пункта подразумевает бесконечное повторение, его называют *циклическим* (рис. 4.2). При наличии в алгоритме анализа некоторого значения и принятия решения в результате такого анализа, этот алгоритм называется *разветвляющимся* (рис. 4.3). Алгоритмы легко анализировать и создавать — для этого нужен только лист бумаги и карандаш. При помощи алгоритмов можно описывать не только компьютерные программы, но и любые планы — например, планировать бизнес-процесс.

Рис. 4.1. Линейный алгоритм

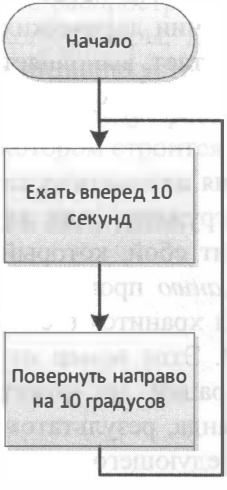


Рис. 4.2. Циклический алгоритм

Рис. 4.3. Разветвляющийся алгоритм

Алгоритмы можно записывать и в простой текстовой форме:

1. Ехать вперед 10 секунд.
2. Повернуть направо на 10 градусов.

А для разветвляющегося и циклического алгоритмов:

1. Получение информации о наличии препятствия с датчика.
2. Если препятствие есть, то перейти на пункт 5.
3. Ехать вперед 10 секунд.
4. Перейти на пункт 6.
5. Повернуть направо на 10 градусов.
6. Перейти на пункт 1.

Текстовая форма алгоритма более похожа на компьютерную программу, но визуально воспринимается хуже, в ней труднее найти логические ошибки, особенно если переходов много.

Для того чтобы создать компьютерную программу, требуется ясно понять, что она должна делать, и главное, как она это будет делать. Если некоторые вопросы вами еще не проработаны, а вы уже сели за написание серьезной программы, то, скорее всего, она не будет работать так, как вы рассчитываете. Поэтому нужно идти от простого к сложному, разбивать программы на маленькие простые блоки, добиваться их работоспособности, а только затем собирать из них полную программу для робота.

Среда разработки Arduino IDE

Перейдем к практике. В основу языка программирования, используемого в проектах Arduino, положен язык C++ — один из самых широко используемых языков программирования, поддерживающий как работу с низкоуровневыми командами, так и построение сложных объектов. Программирование контроллеров Arduino удобно осуществлять в специальной среде Arduino IDE, поскольку в нее включен основной функционал для работы с ними.

Установка Arduino IDE

Arduino IDE можно скачать в Интернете по адресу основного сайта проекта: **www.arduino.cc/en/Main/Software**, программное обеспечение бесплатное, но если есть желание помочь проекту, это можно сделать, нажав кнопку **CONTRIBUTE & DOWNLOAD**, для простого же скачивания нажмите кнопку **JUST DOWNLOAD** (рис. 4.4).

Выберите вариант Windows и скачайте на компьютер программу инсталляции Arduino IDE — в настоящий момент это файл arduino-1.8.1-windows.exe. Его надо запустить на выполнение с административными полномочиями, принять

условия лицензии GNU LESSER GENERAL PUBLIC LICENSE и согласиться с предложенным вариантом установки.

На все предупреждения Windows в процессе установки следует отвечать утвердительно (продолжать установку). Когда установщик предложит установить драйверы порта, также ответить утвердительно.

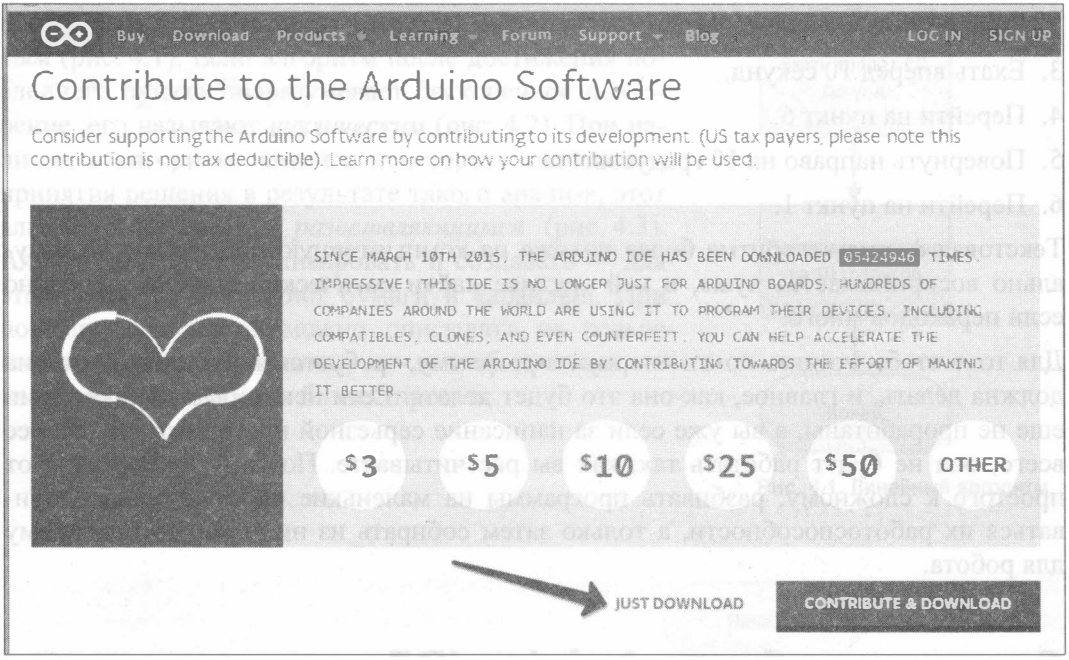


Рис. 4.4. Окно скачивания программы Arduino IDE

Начало работы с Arduino IDE

При запуске установленной Arduino IDE откроется окно (рис. 4.5), в котором уже содержится заготовка программы. Она состоит из двух функций: `setup` и `loop`. Функция `setup` содержит команды, выполняемые один раз при включении Arduino, — это установка номеров портов ввода/вывода для управления моторами и установка скорости обмена данными между Arduino и компьютером. Функция `loop` выполняется бесконечное число раз — до тех пор, пока мы не отключим питание, фактически она зациклена, алгоритмически это изображено на рис. 4.6.

Подключение контроллера Arduino к ПК

Теперь можно подключить контроллер Arduino к компьютеру. В зависимости от того, какой контроллер Arduino используется, могут потребоваться разные кабели:

- для Arduino UNO R3 и Mega — это кабель с разъемом под USB-принтер с одной стороны и стандартным USB-разъемом с другой (рис. 4.7);

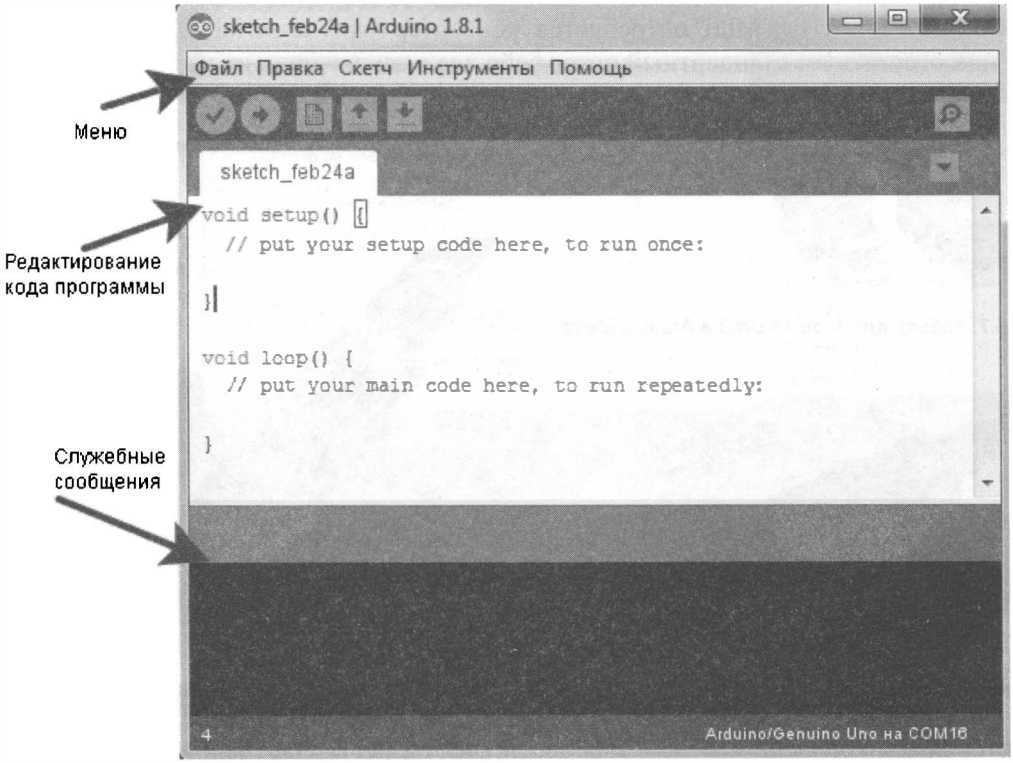


Рис. 4.5. Окно Arduino IDE (заготовка программы)

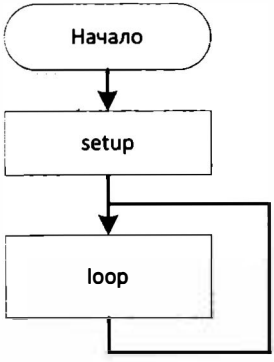


Рис. 4.6. Типовой алгоритм программы в Arduino IDE

- для контроллера Nano требуется кабель с разъемом mini-USB (рис. 4.8);
- для Arduino Micro — это micro-USB, а для программирования контроллера Arduino Mini и Pro Mini потребуется конвертер USB-to-serial (рис. 4.9), т. к. у них отсутствует стандартный интерфейс для подключения их к компьютеру.

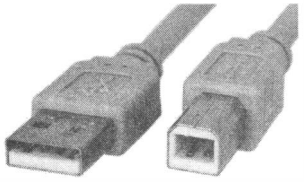


Рис. 4.7. Кабель для Arduino UNO и Arduino Mega

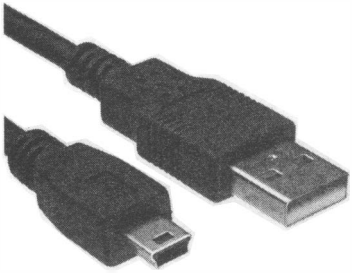


Рис. 4.8. Кабель для Arduino Nano

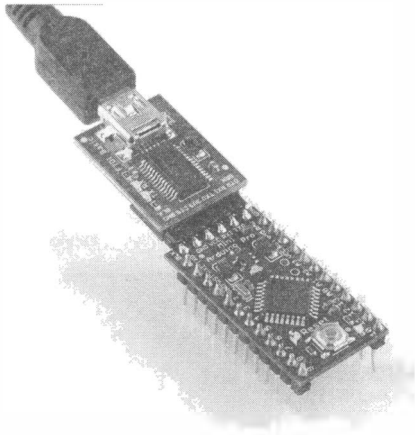


Рис. 4.9. Конвертер USB-to-serial и Arduino Mini

Некоторые контроллеры требуют для своей работы нестандартный драйвер. Так, я встречал контроллеры Arduino, которые требуют установки отдельного драйвера, не входящего в комплект Arduino IDE, — его название ch341ser.exe. Найти его в Интернете не трудно, имеется этот драйвер и в сопровождающем книгу электронном архиве (см. *приложение 2*).

Физически подключив контроллер к ПК, следует установить связь между ним и оболочкой Arduino IDE. Для этого нужно задать номер порта, к которому подключен контроллер (рис. 4.10). Если портов много и найти нужный сложно, рекомендуется запомнить все имеющиеся, а затем физически отсоединить Arduino от кабеля, и снова проанализировать список портов, — тот, который исчез, и есть нужный. Подключайте снова контроллер и выбирайте появившийся порт — для этого нужно установить соответствующий ему флажок. Иногда для появления порта в списке требуется некоторое время, за которое операционная система компьютера анализирует и проверяет подключенное устройство, так что подождите немного до завершения этого процесса

У пользователей Windows XP может возникнуть проблема, а именно — драйвер порта контроллера Arduino не установится автоматически. Решение этой проблемы описано на сайте Arduino по адресу: **http://arduino.ru/Guide/Windows**.

На следующем шаге выберем тип контроллера Arduino. На рис. 4.11 можно видеть, что выбран **Arduino Nano**, если же у вас Arduino Uno, то выбирайте пункт **Arduino/Genuino Uno**.

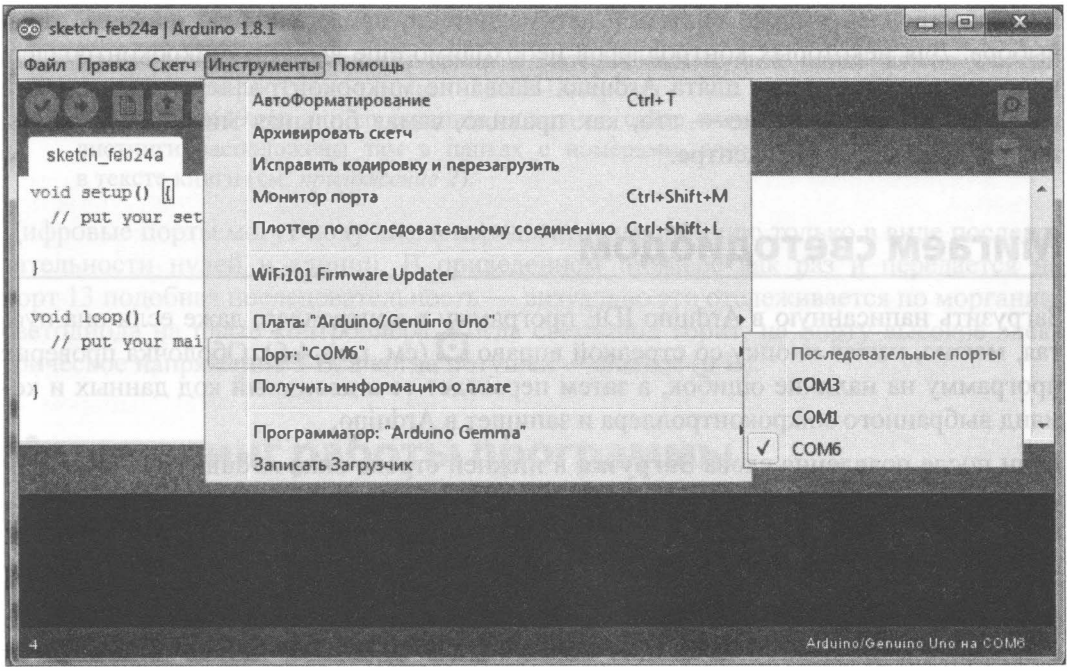


Рис. 4.10. Выбор порта

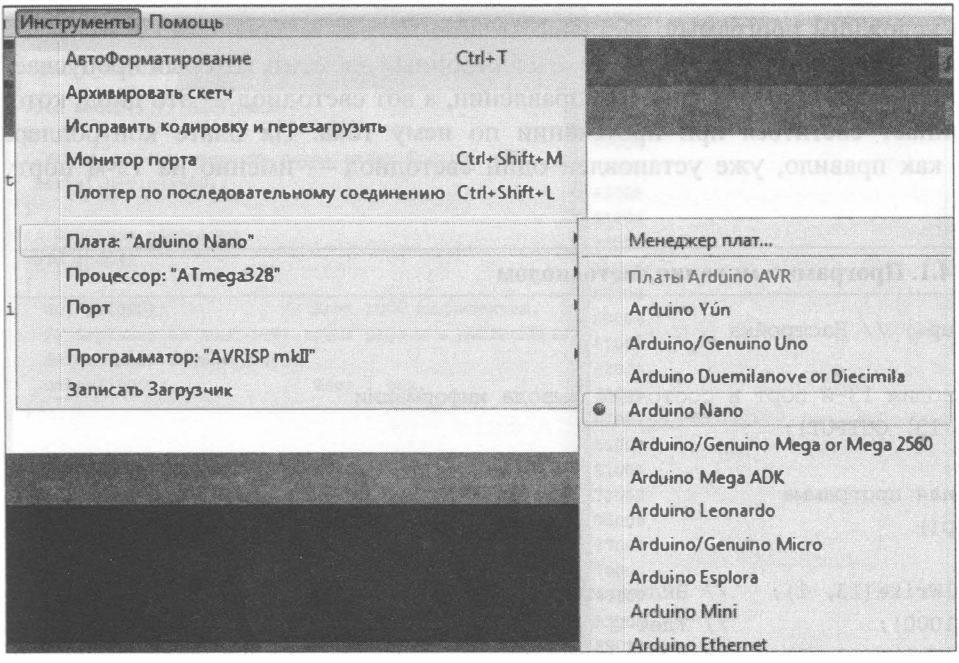


Рис. 4.11. Выбор контроллера Arduino

Если контроллер выбран системой автоматически, проверьте правильность этого выбора. Для некоторых контроллеров необходимо еще выбрать микроконтроллер, на котором реализована плата Arduino. Название микроконтроллера можно найти на самой его микросхеме — это, как правило, самая большая микросхема платы и расположена она в ее центре.

Мигаем светодиодом

Загрузить написанную в Arduino IDE программу в контроллер, даже если она пустая, можно, нажав кнопку со стрелкой вправо  (см. рис. 4.5). Оболочка проверит программу на наличие ошибок, а затем переведет ее в двоичный код данных и команд выбранного микроконтроллера и запишет в Arduino.

Если после появления слова **Загрузка** в нижней строке окна Arduino IDE замигали светодиоды TX и RX на плате Arduino, то загрузка программы в плату началась успешно. Если после этого фраза **Загрузка завершена** не появилась, то, скорее всего, не правильно выбран тип платы Arduino. Если же диоды TX и RX не замигали вообще, то проблемы заключаются в выборе порта платы Arduino.

Если же все пойдет штатно, появится надпись **Загрузка завершена**, и программа автоматически начнет выполняться. Если загружена пустая программа, то ничего и не произойдет, — пустой код будет бесконечно повторяться, пока к плате подключено электропитание.

Немного усложним программу, заставив Arduino мигать встроенным светодиодом на 13-м порту (листинг 4.1). Диод — это электронный элемент, который пропускает электрический ток только в одном направлении, а вот светодиод — это диод, который начинает светиться при протекании по нему тока. На плате контроллера Arduino, как правило, уже установлен один светодиод — именно на 13-м порту (ножке).

Листинг 4.1. Программа мигания светодиодом

```
void setup() // Настройка
{
 // Переводим 13-й порт в состояние вывода информации
 pinMode(13, OUTPUT);
}
// Основная программа
void loop()
{
  digitalWrite(13, 1);   // Включает светодиод на плате.
  delay(1000);           // Ждет 1 сек.
  digitalWrite(13, 0);   // Выключает светодиод.
  delay(1000);           // Ждет 1 сек.
}
```

Электронный архив

Напомню, что электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538619.zip** или со страницы книги на сайте **www.bhv.ru**. Все необходимые листинги расположены там в папках с номерами, соответствующими их номеру в тексте книги (см. *приложение 2*).

Цифровые порты могут получать и передавать информацию только в виде последовательности нулей и единиц. В приведенном примере как раз и передается на порт 13 подобная последовательность — визуально это отслеживается по морганию светодиода на плате контроллера. Когда светодиод горит, на порту высокое электрическое напряжение 5 В, а когда потушен — низкое (0 В).

Мониторинг работы программы

Научим контроллер общаться с компьютером во время выполнения программы и запишем для этого в функции `setup` команду `Serial.begin(9600)`, которая указывает, с какой скоростью происходит обмен информацией с ПК. Для обмена информацией требуется, чтобы приемник и передатчик осуществляли обмен на одинаковой

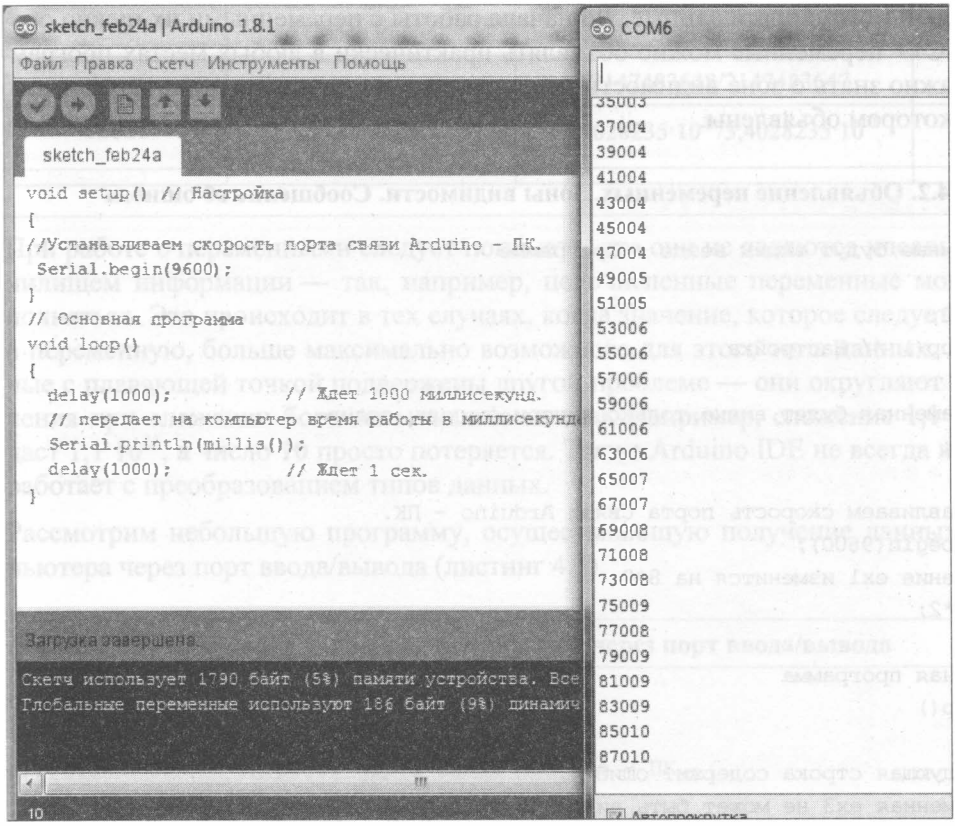


Рис. 4.12. Программа в окне Arduino IDE (*слева*) и результат ее работы в окне монитора порта (*справа*)

скорости, а так как контроллер Arduino достаточно медленный, то наиболее подходящей для обмена с ним является скорость 9600 бод. Обмен информацией на более высоких скоростях также возможен, но задействует большее количество ресурсов контроллера, при этом чаще происходят прерывания текущей программы, и основной код может выполняться медленнее.

Команда `Serial.println()` передает с контроллера на ПК значение, указанное в скобках. Здесь это значение, возвращаемое функцией `millis()`, которая возвращает время работы в миллисекундах. Программа и результат ее работы приведены на рис. 4.12. Для получения на экране компьютера полученных результатов после запуска программы потребуется еще открыть вкладку **Инструменты | Монитор порта** и, возможно, скорректировать скорость порта (в программе это 9600 бод) — в окне **Монитор порта** скорость задается в правом нижнем углу выбором из всплывающего списка.

Переменные

Важной составляющей синтаксиса языков программирования являются переменные. Переменные — это поименованная память, в которую можно вносить данные и из которой можно брать данные. До начала работы с переменными их нужно объявить. В C++ переменные можно объявлять практически в любых местах программы, но важно знать о зоне видимости переменных, — они видны лишь внутри того блока, в котором объявлены.

Листинг 4.2. Объявление переменных. Зоны видимости. Сообщение об ошибке

```
//переменные будут видны везде в программе
int ex1;
float ex2;
void setup() // Настройка
{
  // переменная будет видна только внутри функции setup.
 int ex3;
 ex3=8;
 //Устанавливаем скорость порта связи Arduino - ПК.
 Serial.begin(9600);
 // значение ex1 изменится на 8*2.
 ex1=ex3*2;
}
// Основная программа
void loop()
{
  // Следующая строка содержит ошибку,
  //переменная ex3 не может быть видна за пределами функции setup.
  ex3=ex1+5;
```

```
  delay(ex1*100);            // Ждет 8*2*100 миллисекунд.
  // передает на компьютер время работы в миллисекундах.
  Serial.println(millis());
  delay(1000);               // Ждет 1 сек.
}
```

Листинг 4.2 в демонстрационных целях содержит специально добавленную в него ошибку объявления переменной, и при попытке его скомпилировать об этой ошибке будет выведено сообщение: **'ex3' was not declared in this scope**.

Переменные могут отличаться по типу данных, для хранения которых созданы (табл. 4.1).

Таблица 4.1. *Типы данных*

Обозначение в программе	Название	Принимаемые значения	Размер в памяти
boolean	Логический	True/false, 1/0	1 байт
char	Символьный	–128/+127	1 байт
byte	Короткое беззнаковое целое	0–255	1 байт
int	Целое число	–32768/32767	2 байта
long	Длинное целое число	–2147483648/2147483647	4 байта
float	Число с плавающей точкой	$-3{,}4028235 \cdot 10^{38}/3{,}4028235 \cdot 10^{38}$	4 байта

При работе с переменными следует понимать, что они не являются идеальным хранилищем информации — так, например, целочисленные переменные могут переполняться. Это происходит в тех случаях, когда значение, которое следует записать в переменную, больше максимально возможного для этого типа данных. Переменные с плавающей точкой подвержены другой проблеме — они округляют свои значения при сложении большого числа с малым. Например, сложение $1{,}1 \cdot 10^{25}$ и 10 даст $1{,}1 \cdot 10^{25}$, а число 10 просто потеряется. Также Arduino IDE не всегда корректно работает с преобразованием типов данных.

Рассмотрим небольшую программу, осуществляющую получение данных от компьютера через порт ввода/вывода (листинг 4.3).

Листинг 4.3. Получение данных от компьютера через порт ввода/вывода

```
void setup() // Настройка
{
 //Устанавливаем скорость порта связи Arduino - ПК.
 Serial.begin(9600);
}
```

```
// Основная программа
void loop()
{
  char char1;
 // если поступили данные.
 if (Serial.available() > 0)
 {
  // считываем символ.
  char1 = Serial.read();
  // отсылаем то, что получили, обратно на ПК.
  Serial.println(char1);
 }
}
```

Скомпилируйте и загрузите эту программу. Введите в верхней части окна **Монитор порта** слово `ПРИВЕТ!` и нажмите кнопку **Отправить**. Программа отправит это слово обратно на ПК посимвольно. Результат будет виден в нижней части окна:

П
Р
И
В
Е
Т
!

Условные операторы

Условные операторы мы уже немного использовали, а теперь рассмотрим их подробно.

Оператор *if... else*

Оператор `If ... else` графически, в виде блок-схемы алгоритма, представлен на рис. 4.13, а в виде фрагмента кода — в листинге 4.4.

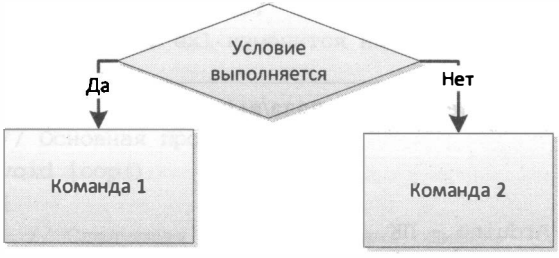


Рис. 4.13. Представление условного оператора

Листинг 4.4. Примененне оператора *if... else* при управлении движением робота

```
if (distance < 10)
 { //Если дистанция до препятствия меньше 10 см.
   Povorot(Right,10); //Поворот вправо на 10 градусов.
 }
else // Иначе, если расстояние больше 10 см.
{
  Vpered(); //Едем вперед 10 секунд.
  delay(10000);
}
```

Как можно видеть, в этом фрагменте не раскрывается суть измерения расстояния, или как наш робот будет двигаться, — это скрыто во внешних функциях.

Создадим программу (листинг 4.5), которая изменяет частоту моргания встроенного светодиода на 13-м порту (рис. 4.14) в зависимости от посланной с компьютера команды. Блок-схема алгоритма этой программы представлена на рис. 4.15.

Программа не завершается, пока есть электропитание, в ней присутствуют три условных блока: один проверяет наличие данных на порту ввода/вывода, а последующие два сравнивают полученный символ с условиями, в зависимости от которых изменяют величину переменной `time_pick`, задающую время задержки при мигании светодиода.

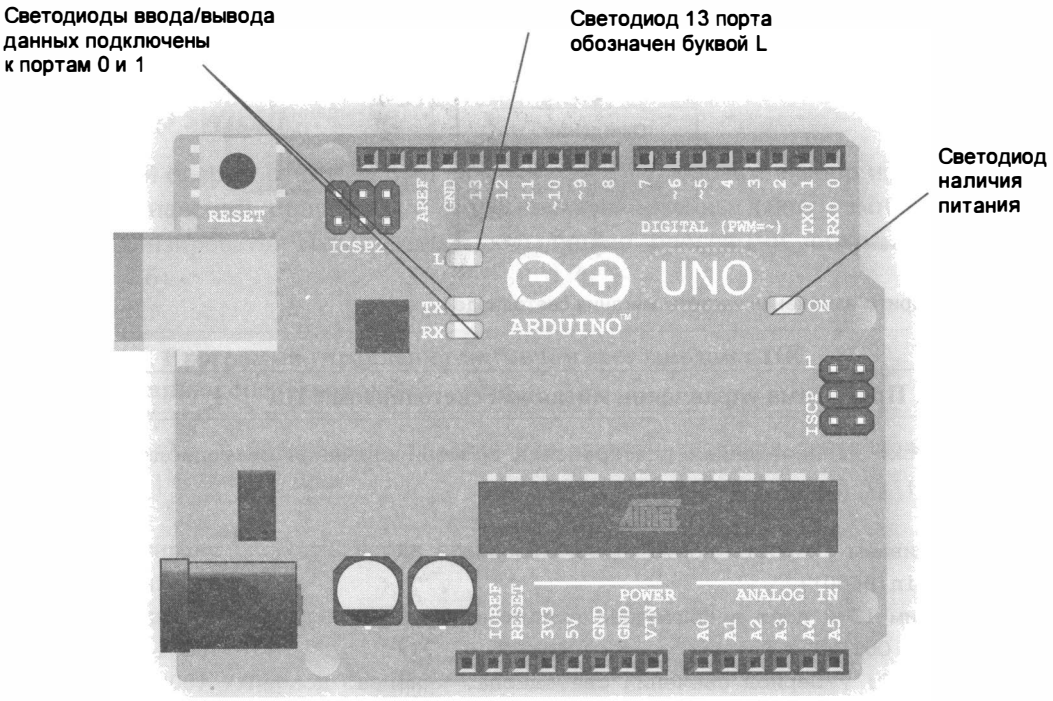


Рис. 4.14. Светодиоды на плате Arduino UNO

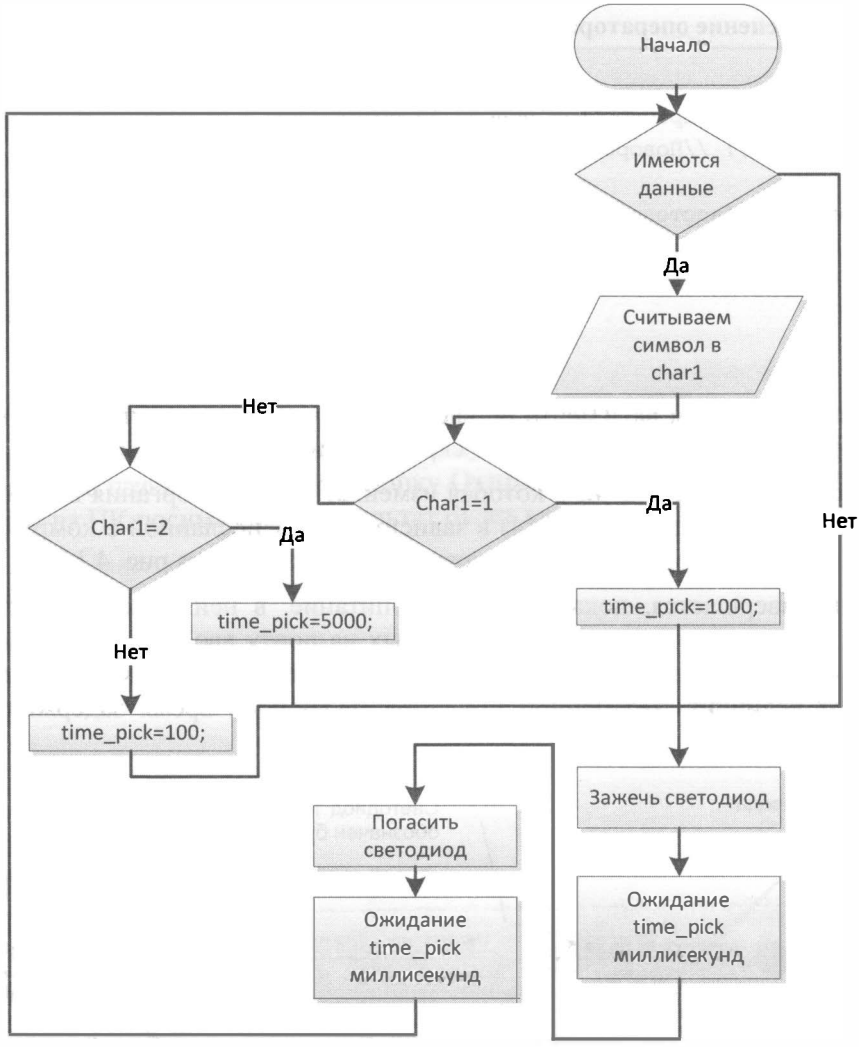


Рис. 4.15. Алгоритм изменения частоты мигания светодиода

Листинг 4.5. Программа управления миганием светодиодом с ПК

```
int time_pick; // Переменная для хранения времени свечения светодиода.
void setup() // Настройка
{
 //Устанавливаем скорость порта связи Arduino - ПК.
 Serial.begin(9600);
 // Переводим 13-й порт в состояние вывода информации
 pinMode(13, OUTPUT);
 time_pick=200; //Период свечения светодиода.
}
```

```
// Основная программа.
void loop()
{
  char char1;
 // если поступили данные.
 if (Serial.available() > 0)
 {
  // считываем символ.
  char1 = Serial.read();
    if(char1=='1') //Если нажата "1".
  {time_pick=1000;} //Задержка мигания 1 сек

  else
  { //Если нажата "2". // Задержка мигания 0,5 сек
    if(char1=='2')
    {
      time_pick=500;}
      // Если нажато что-то другое.
      else {time_pick=100;}} // Задержка мигания 0,1 сек
 }
   digitalWrite(13, 1);   // Включает светодиод на плате.
  delay(time_pick);            // Ждет
  digitalWrite(13, 0);   // Выключает светодиод.
  delay(time_pick);            // Ждет
}
```

Оператор *switch ... case*

Следующий оператор: `switch ... case` — позволяет сделать выбор из набора определенных значений, программа с его использованием выглядит намного проще, ее легче анализировать. Перестроим предыдущую программу, сохранив ее смысл (листинг 4.6).

Листинг 4.6. Программа управления миганием светодиодом с ПК с использованием оператора *switch ... case*

```
int time_pick; // Переменная для хранения времени свечения светодиода.
void setup() // Настройка
{
 //Устанавливаем скорость порта связи Arduino - ПК.
 Serial.begin(9600);
 // Переводим 13-й порт в состояние вывода информации
 pinMode(13, OUTPUT);
 time_pick=200; //Период свечения светодиода.
}
```

```
// Основная программа.
void loop()
{
  char char1;
 // если поступили данные.
 if (Serial.available() > 0)
 {
  // считываем символ.
  char1 = Serial.read();
    switch (char1) {
    case '1':
      time_pick=1000;
      break;
    case '2':
      time_pick=500;
      break;
    default:
      time_pick=100;
  }
 }
   digitalWrite(13, 1);   // Включает светодиод на плате.
  delay(time_pick);            // Ждет
  digitalWrite(13, 0);   // Выключает светодиод.
  delay(time_pick);            // Ждет
}
```

Операторы циклов *while* и *for*

Очень удобными также являются операторы организации циклов. Рассмотрим два оператора: `while` и `for`.

Цикл `while` продемонстрируем на примере программы из листинга 4.1, заменив функцию ожидания `delay()` на оператор цикла по условию (листинг 4.7). Воспользуемся при этом функцией `millis()`, возвращающей количество миллисекунд от старта программы. В результате работы этой программы светодиод включается на 1 сек и выключается на 1 сек.

Листинг 4.7. Программа управления миганием светодиодом с ПК с использованием оператора цикла *while*

```
void setup() // Настройка.
{
 // Переводим 13-й порт в состояние вывода информации.
 pinMode(13, OUTPUT);
}
// Основная программа.
void loop()
```

```
{
  unsigned long time_1;
  unsigned long time_2;
  unsigned long time_3;
time_1 = millis();    //Начальное значение - количество миллисекунд
                      // от включения программы.
time_2=time_1+1000; // +1 секунду.
time_3=time_1+2000; // +2 секунды.
  while(time_1<time_2)
  {
    digitalWrite(13, 1);   // Включает светодиод на плате.
    time_1 = millis(); //Текущее время в time_1.
  }
  while(time_1<time_3)
  {
    digitalWrite(13, 0);   // Выключает светодиод на плате.
    time_1 = millis(); //Текущее время в time_1.
  }
}
```

Оператор цикла `for` хорошо подходит тогда, когда требуется выполнить определенное количество повторений. При построении алгоритмов в виде блок-схем для него есть отдельная фигура (рис. 4.16).

Для примера листинг 4.8 демонстрирует программу, управляющую миганием встроенного в плату светодиода на 13-м порту, при этом светодиод моргает 300 раз с увеличением времени свечения на 1 миллисекунду каждый повтор, а частота мигания уменьшается по мере приближения к концу цикла. Затем все начинается сначала.

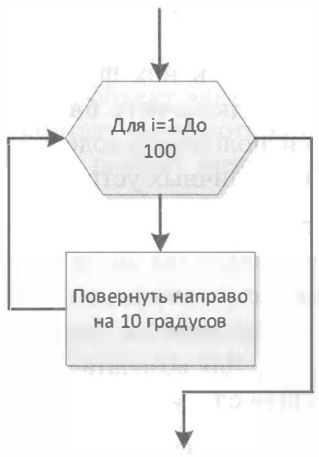


Рис. 4.16. Представление оператора `for`

Листинг 4.8. Пример использования оператора цикла *for* (на 300 повторений)

```
void setup() // Настройка
{
 // Переводим 13-й порт в состояние вывода информации.
 pinMode(13, OUTPUT);
}
// Основная программа.
void loop()
```

```
{
  int i;
  for(i=0;i<300;i++)
  {
  digitalWrite(13, 1);   // Включает светодиод на плате.
  delay(i);           // Ждет i миллисекунд.
  digitalWrite(13, 0);   // Выключает светодиод.
  delay(i);           // Ждет i миллисекунд.
  }
}
```

Функции

Функции являются поименованными блоками команд, которые по их имени можно вызывать из любого места программы. Функции нужно применять там, где существует необходимость одинаковый программный код повторять много раз, или если нагромождение команд в основной программе делает ее нечитаемой.

Существуют встроенные функции — они уже присутствуют в библиотеке функций Arduino IDE, и функции, созданные в рамках текущей программы. Также существуют библиотеки, которые можно подключать к текущей программе и пользоваться имеющимися в них функциями. Подключаются библиотеки из пункта оболочки **Скетч | Подключить библиотеку**. После подключения той или иной библиотеки можно использовать содержащиеся в ней готовые программные решения, написанные для различных устройств и датчиков.

В листинге 4.9 приведен пример написания и использования простой функции — она сравнивает два числа и возвращает наибольшее из сравниваемых. Для функций, которые могут возвращать значения, при их описании в строке перед именем функции устанавливается тип возвращаемого значения, в нашем случае — это целое число: `int`. Для возврата значения из функции используется ключевое слово `return`, за которым следует возвращаемое значение.

При описании функций, не возвращающих значения, в строке перед их именем должно стоять слово `void`.

Действия, производимые функцией, заключаются в фигурные скобки и называются *телом функции*.

Листинг 4.9. Пример написания и использования функции

```
void setup() {
}
void loop() {
  int x=10;
  int y=25;
```

```
// Вызов функции sravnenie().
  int s=sravnenie(x,y);
}
// Определение функции sravnenie ().
// Указываем тип возвращаемого значения, имя функции и информацию о формальных
аргументах.
int sravnenie(int a, int b)
// Далее идет тело функции
{
  if(a<b) return b;
  else return a;
}
```

Элементы объектно-ориентированного программирования

Язык C++, который составляет основу для программирования в проекте Arduino, является объектно-ориентированным. Объекты — это сложные структуры, которые имеют оригинальное название, могут включать в себя как переменные, так и функции. Фактически, объекты созданы для удобного обращения к сложным структурам. Например, когда мы программно подключаем сервомотор, то подключаем сначала библиотеку (`#include <Servo.h>`), а затем создаем объект `servomotor` (листинг 4.10). Естественно, что при этом управляющий контакт сервомотора должен быть физически подключен к порту Arduino, и на мотор подано электропитание (рис. 4.17).

Листинг 4.10. Пример управления сервомотором

```
//Подключаем библиотеку для управления сервомоторами.
#include <Servo.h>
// Создаем объект сервомотор.
Servo servomotor;
void setup()
{
  // Присоединяем сервомотор к 12-му пину Arduino.
  servomotor.attach(12);
}
void loop()
{
 servomotor.write(10); // Поворачиваем вал сервомотора в положение 10°.
// Угол сервомотора увеличивается по часовой стрелке.
 delay(500);// Задержка нужна, чтобы сервомотор успел повернуть.
 servomotor.write(150);//Поворачиваем вал сервомотора в положение 150°.
 delay(500);// Задержка на поворот.
}
```

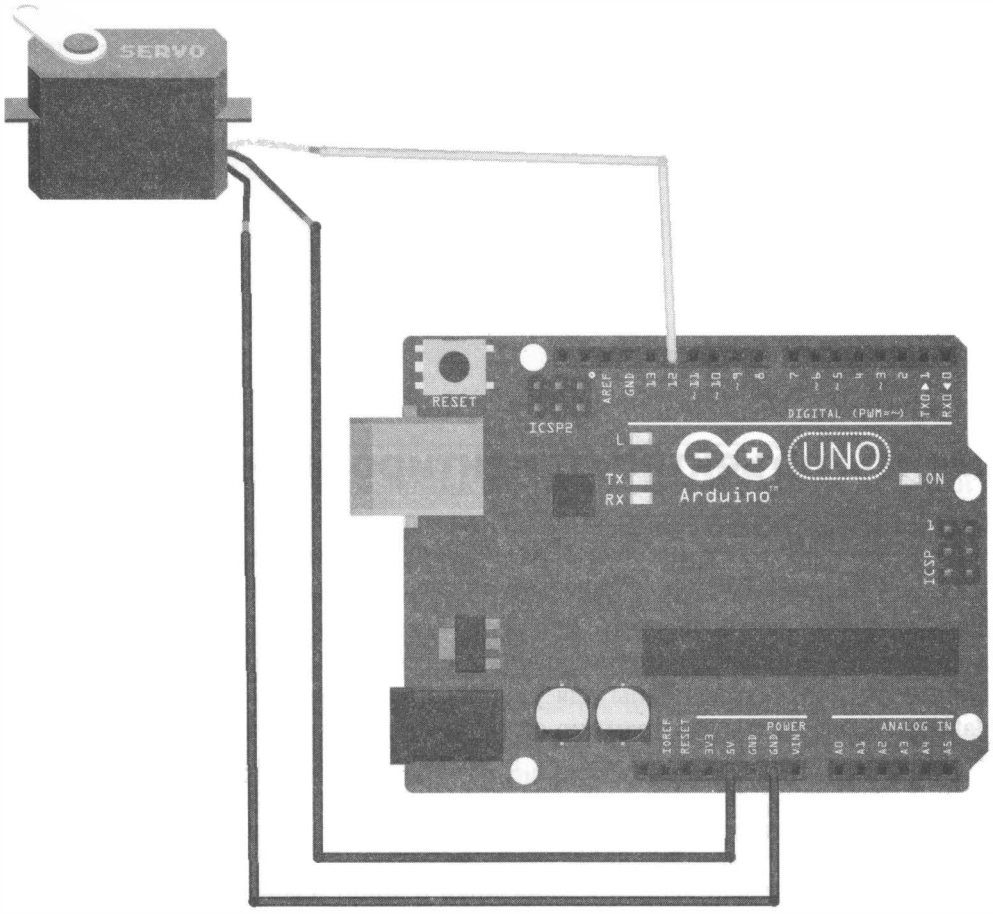


Рис. 4.17. Подключение сервомотора к контроллеру Arduino UNO

Разделение программы (внутренние библиотеки)

Несмотря на то, что в каталоге с программой для Arduino IDE может находиться только одна программа с расширением ino, одноименная с каталогом, программу не обязательно «укладывать» в один файл. В рабочем каталоге с программой можно дополнительно создавать файлы с расширением h, а затем подключать их к программе инструкцией `#include "new_file.h"`, где `new_file.h` — имя созданного дополнительного файла (оно может быть другим, но должно иметь расширение h) (рис. 4.18).

При этом содержимое подключенного файла при сборке программы включается в тело основного файла программы.

Подключенные подобным образом файлы при открытии основной программы автоматически открываются на редактирование в среде Arduino IDE в отдельных

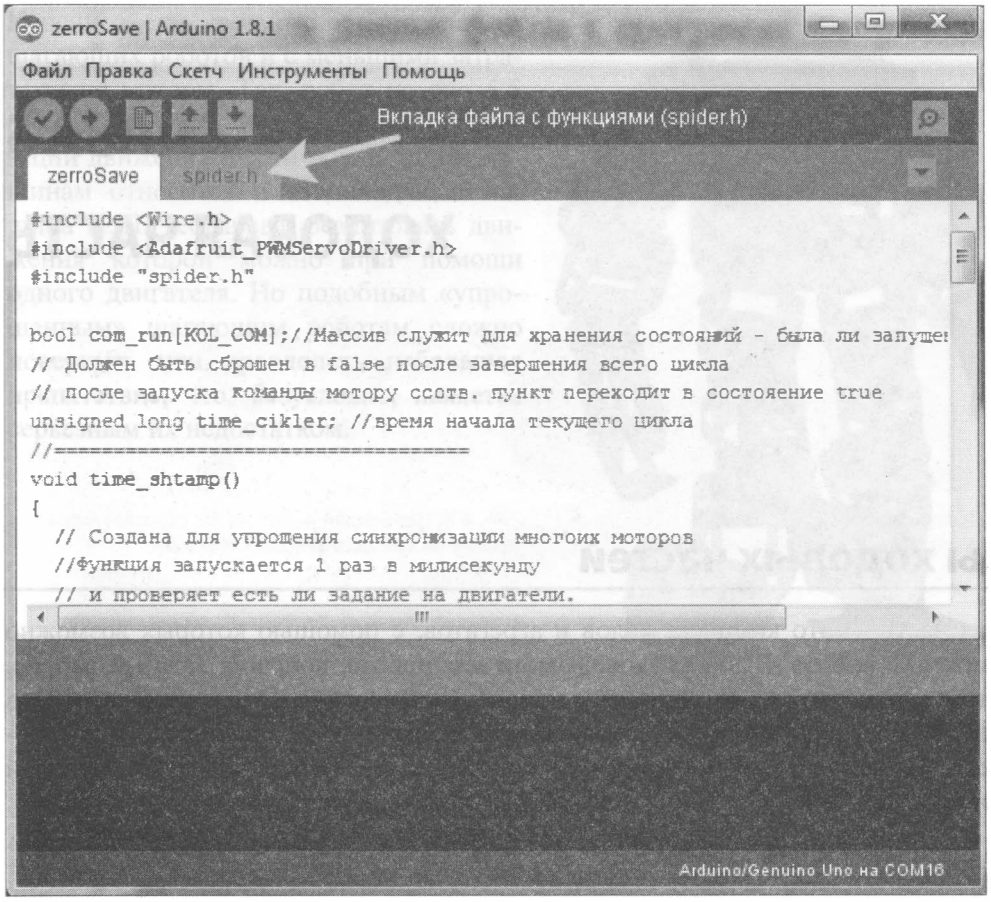


Рис. 4.18. Пример разделения программы на два файла: основной — zerroSave.ino и дополнительный — spider.h

вкладках, что удобно для редактирования. В своей практике я создаю подобные файлы для хранения библиотек функций. При этом для функций различной направленности создаю разные файлы. Таким образом, основной файл программы проекта освобождается от лишней, затрудняющей понимание, информации.

Выводы

Начальное знакомство с языком программирования Arduino C++ завершено. Более детальную информацию можно получить из тематической литературы и на сайте **arduino.ru** в разделе **Программирование**.

Далее в книге также будут рассмотрены типичные программы функционирования роботов: управления двигателями постоянного тока, измерения расстояния датчиком-дальномером, взаимодействия с электронным компасом и др.

А в следующих двух главах мы познакомимся с конструкциями роботов и соберем свою базовую модель.

ГЛАВА 5
ХОДОВАЯ ЧАСТЬ

Типы ходовых частей

Ходовая часть — это комплекс узлов и агрегатов, с помощью которых возможно передвижение робота. Выбор ходовой части для робота, конечно, важный вопрос. При этом можно исходить либо из поставленных перед роботом задач, либо из имеющихся ресурсов, а если основная задача робота — это знакомство с робототехникой, то немаловажным критерием становится доступность деталей ходовой части и простота ее изготовления. Рассмотрим некоторые виды ходовых частей.

Ноги

Ходовую часть робота можно организовать на основе ног (шагающий робот). Бывают они как двуногие (рис. 5.1), так и многоногие (рис. 5.2–5.4). Двуногие роботы применяются там, где требуется уподобить робота человеку, их тогда называют *человекоподобными*, или *андроидами*.

Двуногим роботам требуется большое количество высокоточных моторов — так, для робота, показанного на рис. 5.1, в создании ног задействовано 10 сервоприводов, а это усложняет управление ими, поскольку вращение всех моторов нужно синхронизировать. Кроме того, потребуется научить робота держать равновесие, а при падении он должен уметь вставать на ноги. Так что, поскольку мы только знакомимся с робототехникой, временно отложим создание и использование подобной ходовой части.

Роботы, ходовые части которых состоят из 4-х и более ног, более устойчивы и могут не содержать механизмов балансировки, т. к. падение их почти невозможно. Ноги подобных роботов проще в реализации и управлении, да и содержат они по три (см. рис. 5.2) или даже по два (см. рис. 5.3) сервопривода. Количество сервоприводов для разных ног может различаться — например, робот на рис. 5.4 имеет по три сервопривода на передних ногах и по два на задних.

Конечно, есть возможность создавать шагающих роботов и с меньшими затратами на моторы. Так, робот на рис. 5.5 обходится двумя моторами для организации движения шести ног. К таким машинам относится и стопоходящая машина П. Л. Чебышева, реализовать движения которой можно при помощи одного двигателя. Но подобным «упрощенным» шагающим роботам сложно повернуть или преодолеть небольшое препятствие, что, безусловно, является серьезным их недостатком.

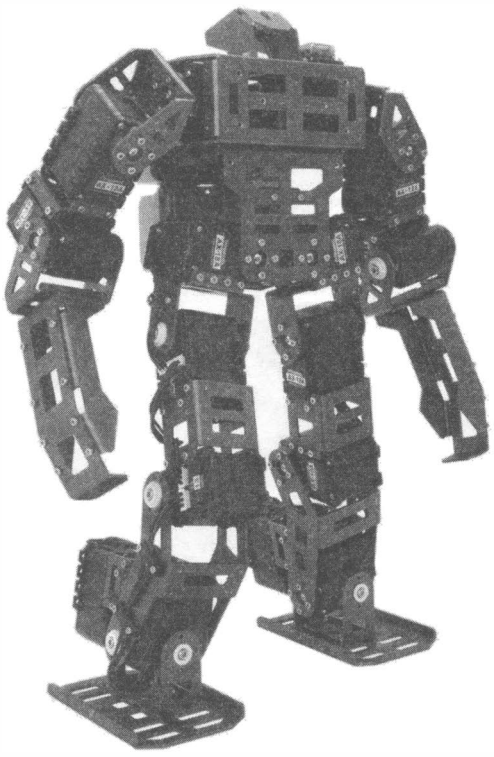


Рис. 5.1. Двуногий шагающий робот

Рис. 5.2. Четырехногий робот на 12-ти сервомоторах (по 3 на ногу)

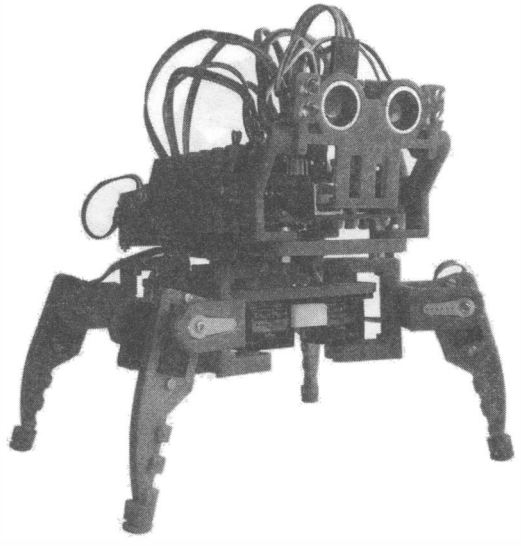


Рис. 5.3. Шагающий робот на 8-ми сервомоторах (по 2 на ногу)

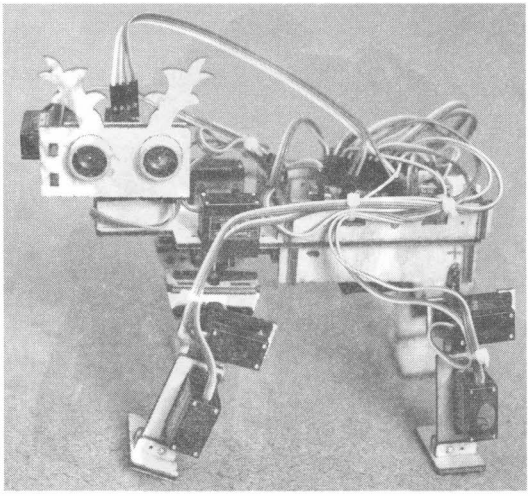


Рис. 5.4. Шагающий робот на 10-ти сервомоторах (по 3 на передних ногах, по 2 на задних)

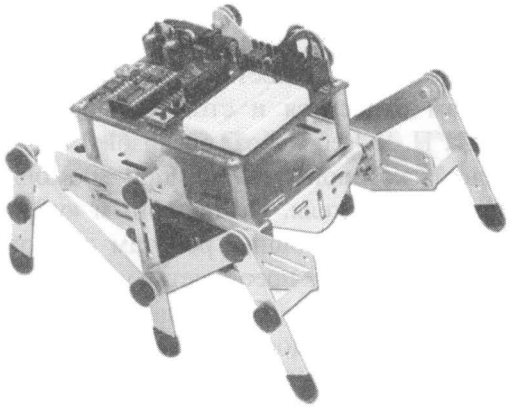


Рис. 5.5. Шагающий робот на 2-х сервомоторах (по одному на три ноги каждой стороны)

Гусеницы

Гусеничная ходовая часть (рис. 5.6) весьма широко распространена в робототехнике благодаря простой реализации и высокой проходимости роботов, построенных на ее основе. Все, что будет описываться в практических главах, относится и к ходовой части, реализованной на основе гусениц.

Если у вас есть под рукой гусеничная ходовая часть, например, от сломанной радиоуправляемой модели, смело можете использовать ее для создания робота по принципам управления, описываемым далее.

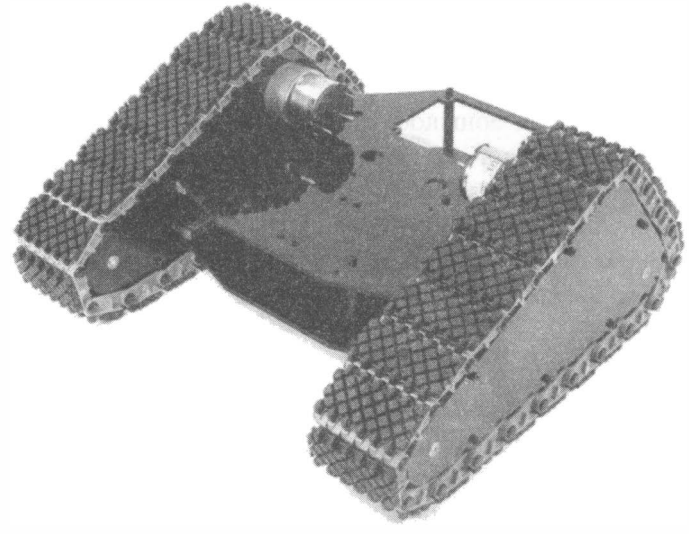


Рис. 5.6. Гусеничная ходовая

Колеса с дифференциалом

Колесная ходовая часть с дифференциалом часто встречается в радиоуправляемых моделях машин. Как правило, в них задействовано два мотора постоянного тока: на дифференциал задних колес и на рулевую тягу (рис. 5.7). Чтобы воспользоваться материалом практических глав для создания робота на таком ходу, вам придется существенно изменить функции, реализующие движения и повороты.

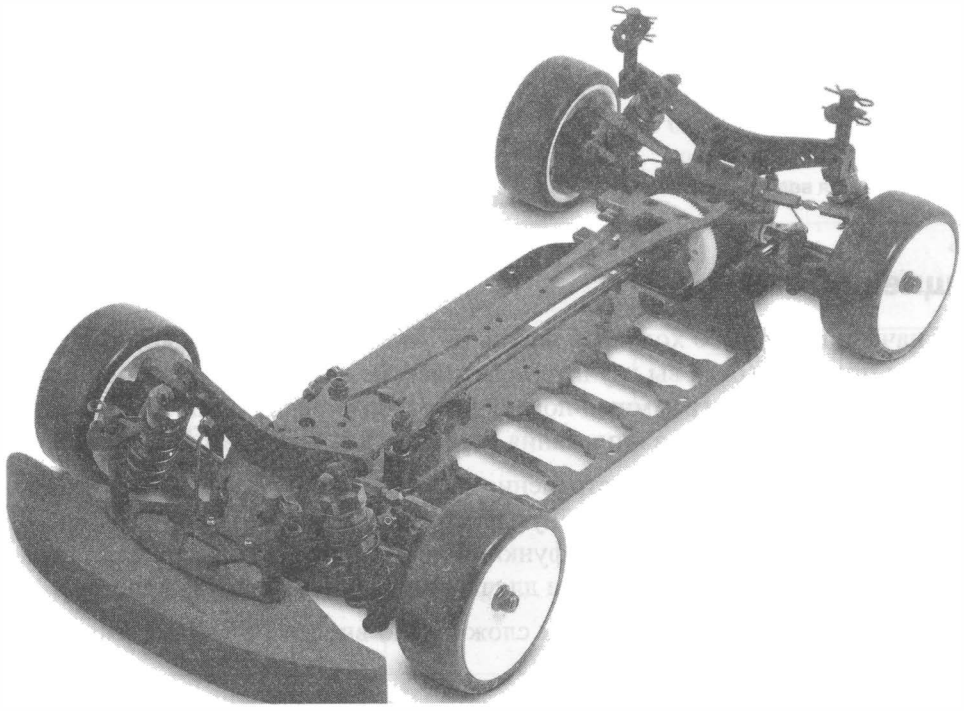


Рис. 5.7. Ходовая с дифференциалом на заднем мосту

Колеса на моторах

Колесные ходовые части с мотором на каждое колесо бывают двух видов: с двумя моторными колесами и одним опорным роликом (рис. 5.8) и с четырьмя моторными колесами (рис. 5.9).

Существенных различий между ними нет, кроме того, что четырехколесный робот более мощный и устойчивый. Поэтому именно четырехколесная ходовая будет использована в проектах, представленных в *главах 10–13*. Но если вам по душе робот, имеющий ходовую с двумя моторами, смело берите ее за основу. Особенности задействованной в проектах электроники позволяют применять все примеры программ без каких-либо изменений.

Подобные ходовые части можно приобрести в специализированных магазинах и на сайтах интернет-магазинов, занимающихся продажей конструкторов для роботов.

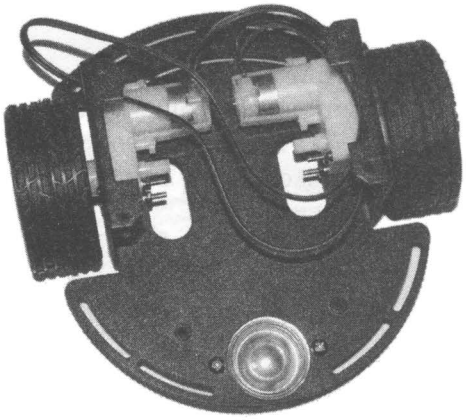


Рис. 5.8. Ходовая с 2-мя ведущими колесами

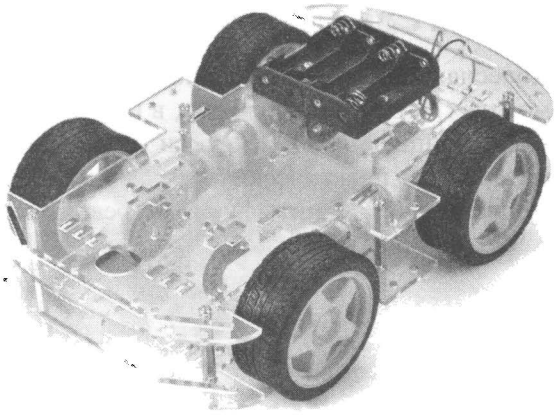


Рис. 5.9. Ходовая с 4-мя ведущими колесами

Летающие роботы

Раз уж мы начали говорить о ходовых, нельзя не упомянуть роботов, которые умеют летать или плавать. Когда вы изучите практические главы и поймете принципы управления моторами и программную логику роботов, то реализация более экзотических конструкций для полетов и плавания перестанет быть для вас проблемой.

В настоящее время широкое распространение получили радиоуправляемые квадрокоптеры (рис. 5.10), и наиболее продвинутые из них уже можно отнести к классу роботов, поскольку в них реализованы функции программного управления, а также использования компаса и GPS-навигации для планирования маршрута полетов.

Конечно, управление летающим роботом сложнее, но вполне по силам даже начинающему конструктору.

Рис. 5.10. Квадрокоптер с установленной видеокамерой

Выбор двигателей

Итак, когда структура ходовой выбрана, следует определить, какие двигатели будет использовать наш робот. Конечно, исходить при этом надо из наличия соответствующих комплектующих и/или поставленных задач, поэтому вам предстоит выбор между двигателями постоянного тока с редуктором, сервомоторами постоянного вращения и шаговыми двигателями. Их различия представлены в табл. 5.1.

Таблица 5.1. *Сравнительные параметры различных типов двигателей*

Сравниваемые параметры	**Двигатель постоянного тока с понижающим редуктором**	**Сервомотор постоянного вращения**	**Шаговый двигатель**
Количество задействованных портов микроконтроллера для управления	2–3	1	4
Скорость реакции на поступившую команду, сек	~1/1000	~1/10	~1/1000
Скорость вращения (с учетом понижающего редуктора), об/мин	20–240	1-60	1-60
Стоимость (за единицу принята стоимость двигателя постоянного тока с редуктором), сравниваются двигатели, равные по мощности	1	5	3
Минимальное количество портов управления ходовой частью для проекта колесного робота с 4-мя ведущими колесами	4 (при отказе от управления мощностью и параллельном управлении двумя правыми и двумя левыми моторами)	2 (при параллельном управлении двумя правыми и двумя левыми моторами)	8 (при параллельном управлении двумя правыми и двумя левыми моторами)

Как можно видеть, наиболее быстрая реакция у двигателя постоянного тока, у него также и самая низкая стоимость. Сервомотор превосходит его за счет задействования меньшего количества портов Arduino. Шаговый двигатель при использовании в ходовой части не дает особых преимуществ, хотя если количество сделанных роботом оборотов колес нужно считать и строго дозировать, то это неплохой выбор.

Сервомоторами хорошо регулируется скорость движения робота, если скорость колес нужно менять постоянно и держать в точно заданных пределах. Скорость оборота колеса, вращаемого сервомотором, зависит только от установленного на порту значения, и при увеличении сопротивления (когда, например, робот движется

в гору) практически не меняется, в то время как двигатель постоянного тока изменяет скорость своего вращения в зависимости от нагрузки.

Шаговым двигателем также можно держать постоянную скорость, но технически реализовать это несколько сложнее из-за большего количества задействованных портов Arduino. Так, роботу, имеющему 4 ведущих колеса, потребуется 16 портов управления, правда правые и левые колеса можно подключить к управлению параллельно, и тогда количество портов уменьшится до 8, что, однако, тоже избыточно.

Тем не менее, все три вида двигателей применяются в ходовых частях колесных роботов, и выбор зависит от требований, предъявляемых к роботу. Далее будет рассматриваться ходовая с двигателями постоянного тока, снабженными редукторами, т. к. строгих требований к скорости и мощности робота мы предъявлять не станем.

Драйверы двигателей

Платы Arduino, кроме специализированных, не поддерживают возможностей управления двигателями постоянного тока напрямую, и для этого приходится применять специальные специализированные микросхемы, называемые *драйверами двигателей*. Наиболее распространенные из них приведены в табл. 5.2.

Таблица 5.2. *Специализированные микросхемы драйверов двигателей*

Характеристики	L293D	L293B	L298N
Максимальный ток на канал, А	0,6	1	3
Количество каналов	4	4	4
Встроенная диодная защита от паразитных токов	Есть	Нет	Нет

Здесь рассмотрены три микросхемы драйверов, которые подходят для управления направлением вращения и мощностью двигателей постоянного тока. Они же могут применяться и для управления шаговыми двигателями.

Микросхему L293D можно использовать без дополнительных доработок, более того — для повышения максимального тока эти микросхемы иногда применяются парами и снабжаются радиаторами. Микросхемы L298N и L293B должны быть установлены вместе с защитными диодами, которые предохраняют их от перегрузок.

Если вы не боитесь пайки, то можно приобрести две микросхемы L293D, установить их друг на друга и спаять ножки, а к средним ножкам припаять дополнительный радиатор — например, кусок толстой медной проволоки. Теперь это изделие можно использовать в качестве драйвера для нашей модели (рис. 5.11).

Однако с целью сохранения простоты сборки в проектах будет использована схемная реализация на основе микросхемы L298N (рис. 5.12). Она довольно мощная, имеет хороший запас по максимальному току (3 А на канал), на плате уже установлены ограничительные диоды и имеется дополнительный стабилизатор напряжения на 5 В.

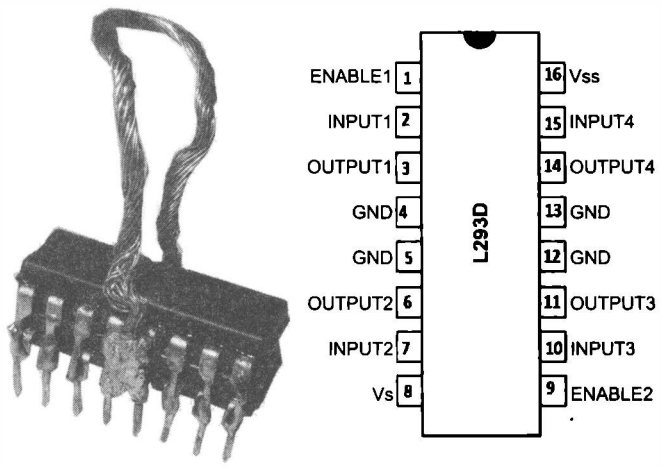


Рис. 5.11. Спаренная микросхема L293D с радиатором из проволоки (*слева*); назначение контактов микросхемы (*справа*)

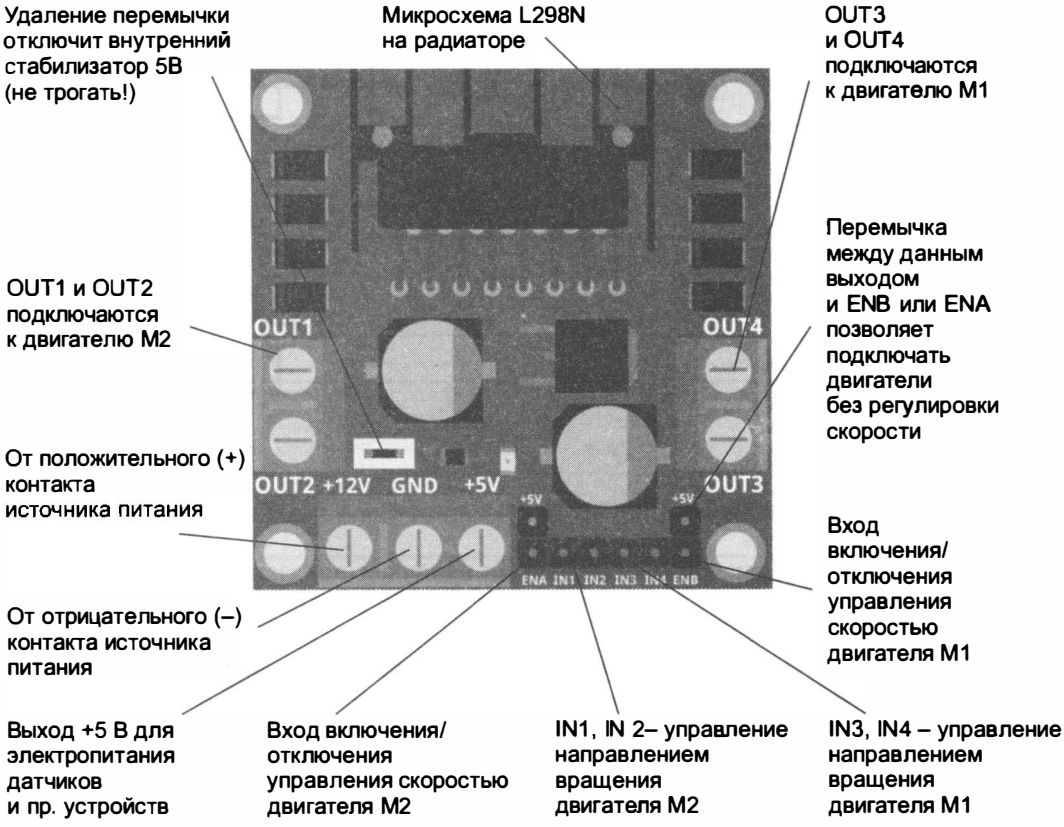


Рис. 5.12. Специализированная плата драйвера на микросхеме L298N

Порядок подключения микросхем, показанных на рис. 5.11 и 5.12, одинаков. Если не требуется обеспечивать двустороннее вращение двигателя, то можно использовать схему, приведенную на рис. 5.13. В этом случае к каждому контакту Out (1, 2, 3, 4) можно подключить по двигателю. Высокий сигнал на контакте In 4 включит мотор M1, а высокий сигнал на In 3 — мотор M2. Постоянным колебанием сигнала на контакте Enable 2 можно изменять скорость вращения и мощность, отдаваемую в двигатели, — фактически, Enable 2 может быстро включать и отключать электропитание двигателей M1 и M2, что приводит к указанному эффекту. Такое явление называется *широтно-импульсной модуляцией* (ШИМ), Arduino на некоторых портах поддерживает ШИМ, такие порты отмечены на плате волнистой линией (~). Для Arduino UNO порты с ШИМ имеют номера 3, 5, 6, 9, 10, 11. В эти порты при использовании их как ШИМ может быть записано число от 0 до 255, при этом 0 — соответствует полному отсутствию сигнала (низкий сигнал), 255 — генерации импульсов нет, на выходе постоянно установлен высокий сигнал, 123 — соответствует генерации импульсов, у которых полтакта сигнал высокий, а полтакта — низкий.

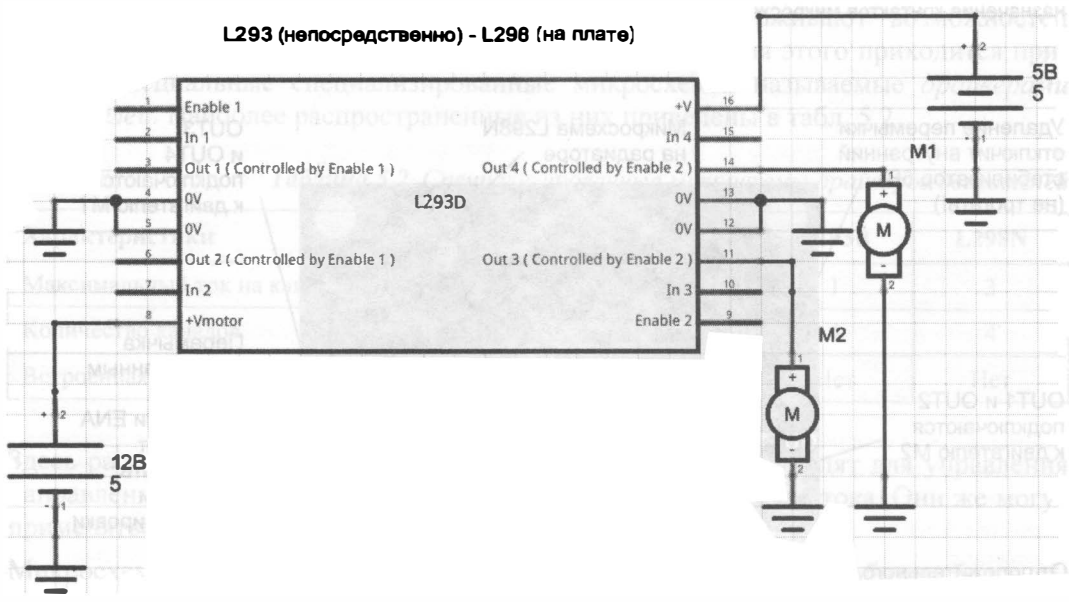


Рис. 5.13. Подключение двигателей для одностороннего вращения

В табл. 5.3 приведены значения сигналов на входах и показана соответствующая реакция двигателя.

Таблица 5.3. *Значения сигналов на входах и соответствующая реакция двигателя*

In 4	Enable 2	M1 (направление вращения условное)
0	0	Отключен
0	1	Отключен
1	1	Вал вращается по часовой стрелке

Таблица 5.3 (окончание)

In 4	Enable 2	M1 (направление вращения условное)
1	0	Отключен
1	ШИМ	Вал вращается по часовой стрелке (мощность зависит от величины положительного импульса ШИМ)

Широтно-импульсная модуляция

Рассмотрим подробнее, как работает широтно-импульсная модуляция. В режиме ШИМ в соответствующий порт командой `analogWrite (Порт, Значение)` записывается определенное число. В зависимости от этого числа на выходе порта с частотой около 490 Гц генерируется последовательность импульсов, вид которой зависит от записанного числа. Эта зависимость показана на рис. 5.14. На время положительного фронта включается двигатель — соответственно, чем шире положительный фронт, тем дольше подключен двигатель, тем больше его мощность и скорость вращения.

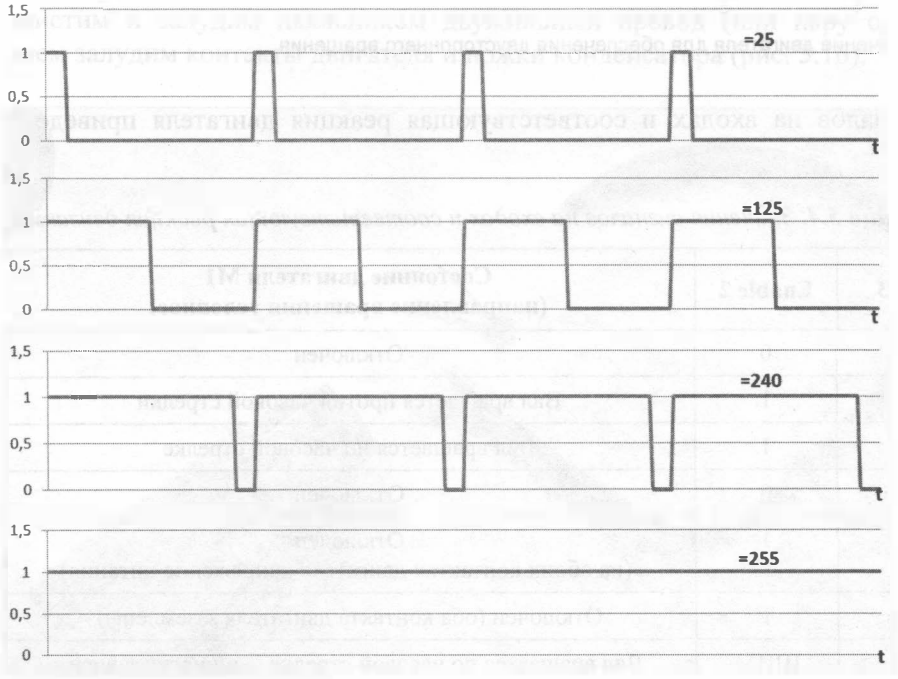


Рис. 5.14. Принцип работы ШИМ

Вращение в обе стороны

Если вал двигателя должен вращаться в обоих направлениях, нужно использовать схему, приведенную на рис. 5.15. Она позволяет подключать только два независимых двигателя, но параллельное подключение двигателей является допустимым.

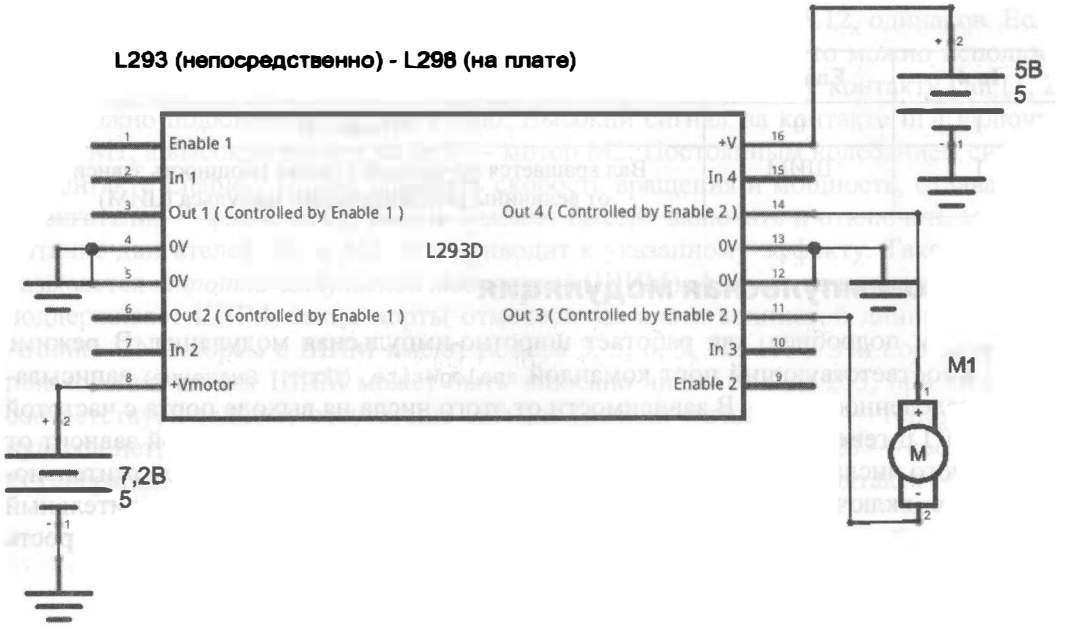


Рис. 5.15. Подключение двигателя для обеспечения двустороннего вращения

Значения сигналов на входах и соответствующая реакция двигателя приведены в табл. 5.4.

Таблица 5.4. *Значения сигналов на входах и соответствующая реакция двигателя*

In 4	In 3	Enable 2	Состояние двигателя M1 (направление вращения условное)
0	0	0	Отключен
0	1	1	Вал вращается против часовой стрелки
1	0	1	Вал вращается по часовой стрелке
1	1	0	Отключен
1	1	1	Отключен (на обоих контактах двигателя напряжение питания)
0	0	1	Отключен (оба контакта двигателя заземлены)
1	0	ШИМ	Вал вращается по часовой стрелке (мощность зависит от величины положительного импульса ШИМ)
0	1	ШИМ	Вал вращается против часовой стрелки (мощность зависит от величины положительного импульса ШИМ)

Подключение второго двигателя осуществляется аналогично — с другой стороны драйвера на порты Out 1 и Out 2.

При использовании показанной на рис. 5.12 платы с драйвером L298N питание логической части (стабилизированные 5 В) на драйвер подавать не нужно, поскольку на плате присутствует собственный стабилизатор напряжения. Более того, сама плата может использоваться как стабилизированный источник напряжения на 5 В для внешних устройств.

Сборка макета

Для закрепления материала полезно собрать несколько макетов, которые продемонстрируют работу двигателей постоянного тока вместе с драйвером L298N.

Управляем двигателем без Arduino

Потребуются: драйвер L298N, двигатели постоянного тока, аккумуляторы с боксом, провода, керамические конденсаторы 0,1 мкФ (маркируются числом 104), паяльник. При пайке легко испортить поверхность стола, поэтому используйте фанеру или постелите на место работы с паяльником пару листов бумаги.

Очистим и залудим паяльником двужильный провод (или пару одножильных), затем залудим контакты двигателя и ножки конденсатора (рис. 5.16).

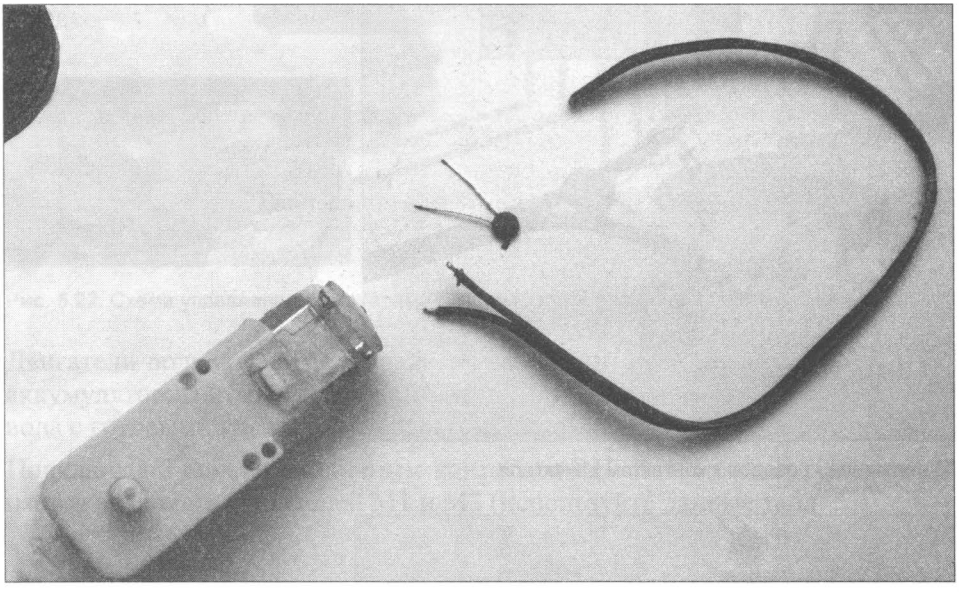


Рис. 5.16. Двигатель постоянного тока, керамический конденсатор и провода

Обмотаем ножки конденсатора вокруг залуженных кончиков проводов, как показано на рис. 5.17.

Теперь ножки конденсатора хорошо держатся на проводе, и их легко можно припаять (рис. 5.18).

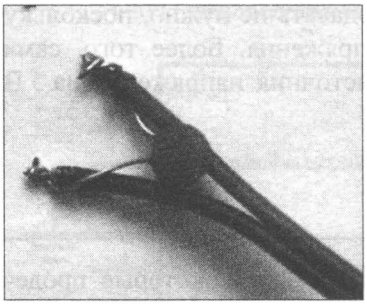


Рис. 5.17. Ножки конденсатора обмотаны вокруг оголенных концов провода

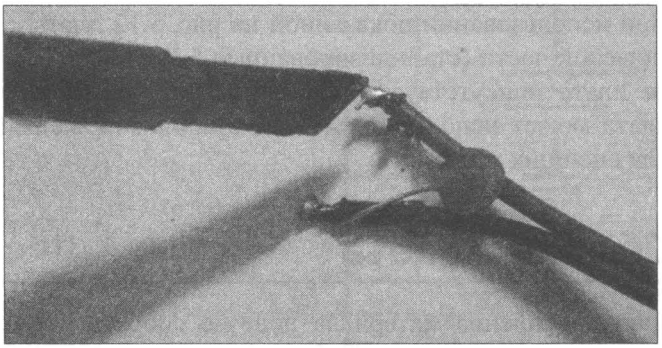


Рис. 5.18. Пайка конденсатора

Далее припаяем кончики проводов к залуженным контактам двигателя (рис. 5.19). Если лужение контактов двигателя проходит плохо, аккуратно зачистите контакты перочинным ножом или обработайте паяльной кислотой (после паяльной кислоты нужно промыть контакты спиртом или водой).

Готовый к работе двигатель показан на рис. 5.20.

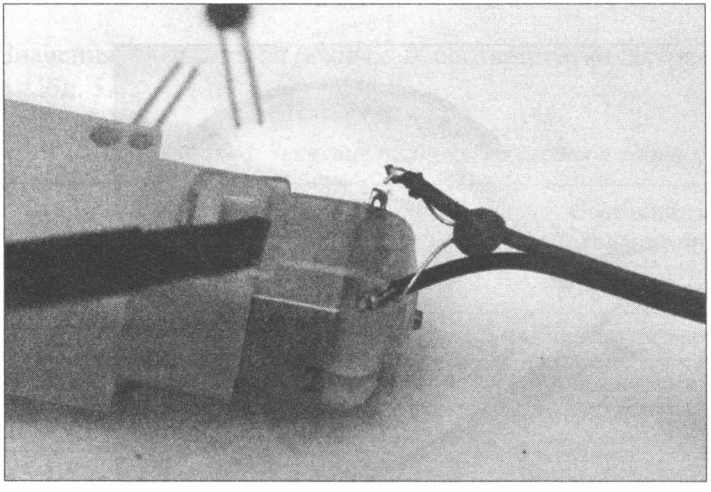


Рис. 5.19. Припаивание проводов к контактам двигателя

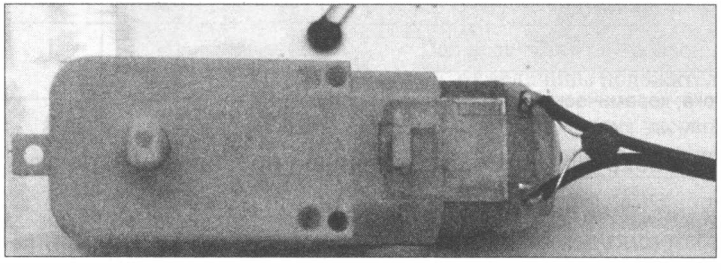


Рис. 5.20. Двигатель с припаянными проводами и конденсатором

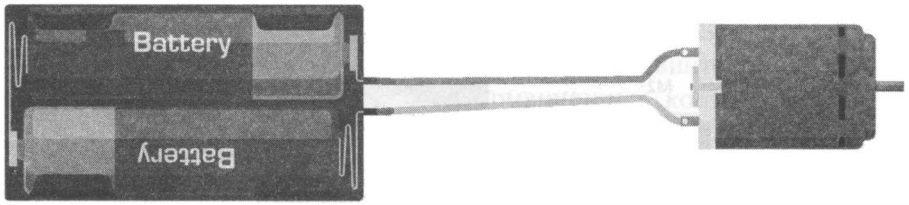


Рис. 5.21. Схема тестирования двигателя

Качество пайки можно проверить, присоединив двигатель к аккумуляторам (рис. 5.21), — если двигатель не работает, значит, пайка выполнена не качественно (предварительно проверьте наличие напряжения на контактах аккумуляторного бокса).

Теперь, когда двигатели готовы, приступим к сборке схемы, изображенной на рис. 5.22. Перемычки на контактах ENA и ENB драйвера не убираем!

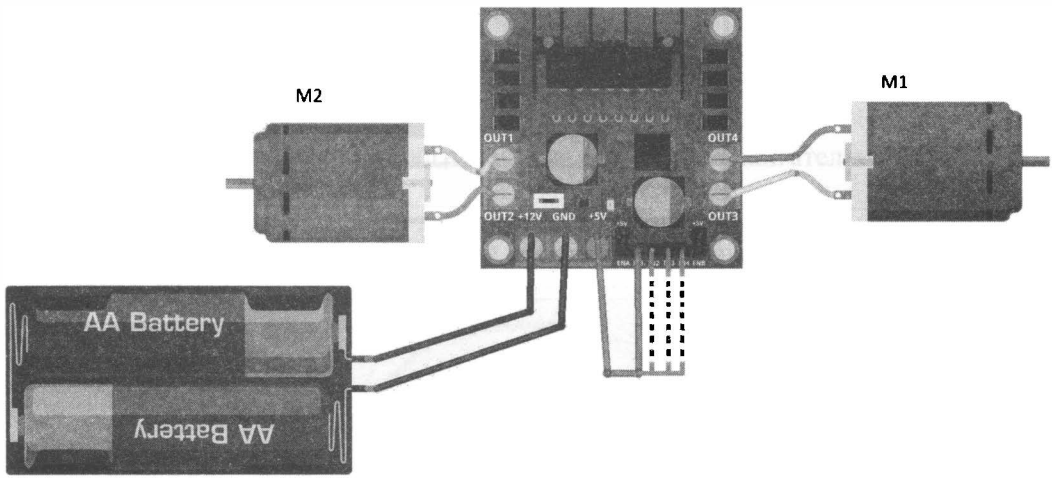


Рис. 5.22. Схема управления двигателями без контроллера Arduino

Двигатели подключаются к драйверу через винтовые зажимы, также подключается аккумуляторный бокс, а для контактов IN1–IN4 драйвера лучше использовать провода с готовыми клеммами Dupont (см. *главу 2*).

Подключая 5 вольт к различным контактам IN1–IN4, можно проследить, как изменяется вращение двигателей M1 и M2 (используйте данные табл. 5.4).

> ***Подсказка***
>
> Контакты IN1 и IN2 отвечают за работу двигателя M2, а IN3 и IN4 — двигателя M1.

Подключаем контроллер Arduino

Добавим к собранной схеме контроллер Arduino и научим его управлять двигателями (рис. 5.23). Дополнительно потребуются несколько проводов с клеммами Dupont.

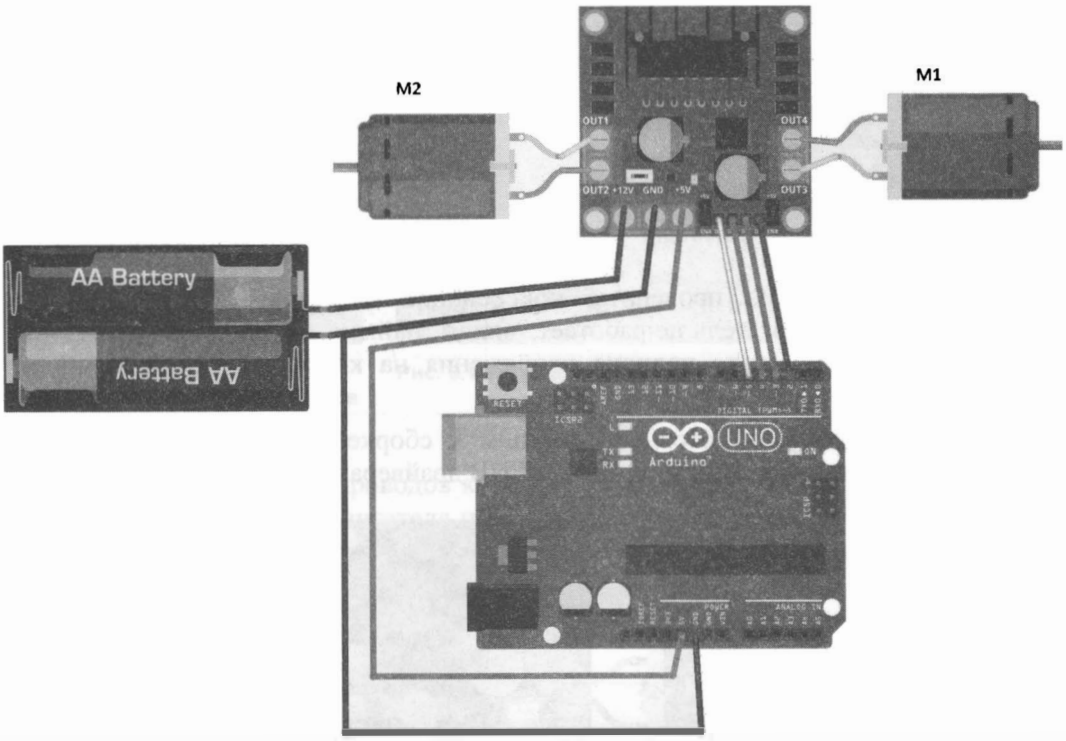


Рис. 5.23. Схема управления двигателями посредством Arduino (без регуляции ШИМ)

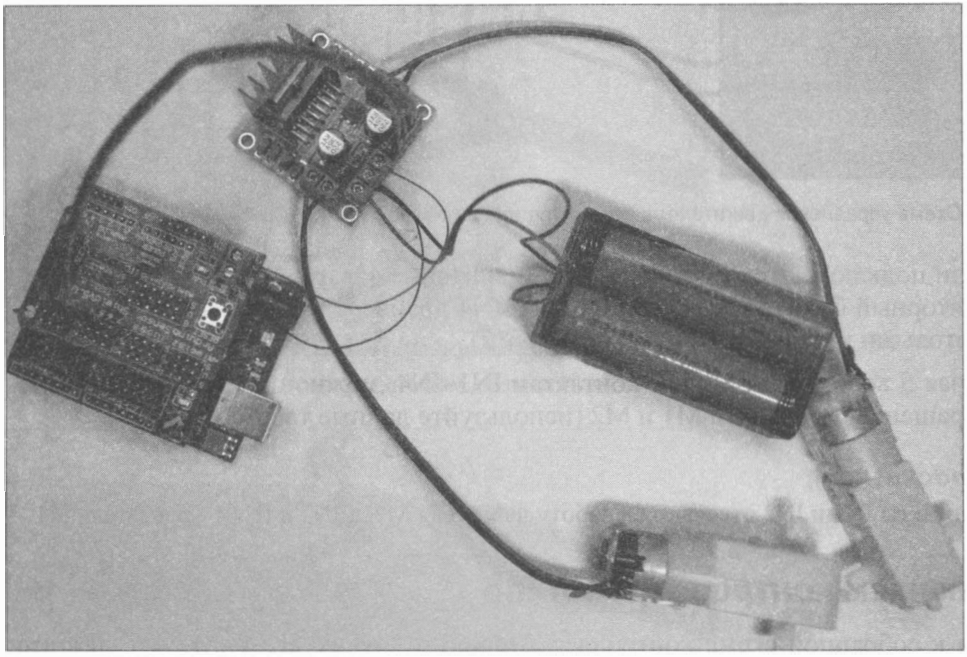


Рис. 5.24. Схема в сборе: Arduino UNO с установленным на нем Arduino Sensor shield V5.0, драйвер двигателей L298N, литиевые аккумуляторы в боксе, двигатели постоянного тока с редукторами

Удобно задействовать дополнительную плату Arduino Sensor shield V5.0 (рис. 5.24). Для этого потребуются два гибких провода под винтовой зажим и четыре типа «мама-мама» для объединения информационных контактов драйвера и Arduino Uno.

Далее приведены примеры управления вращением двигателей при помощи программы, работающей на контроллере Arduino UNO.

Тестовая программа управления двигателями

Логика работы этой тестовой программы (листинг 5.1) следующая.

В бесконечном цикле:

1. Отключаем оба двигателя.
2. Вращаем двигатель M2 1 сек в одну сторону: `digitalWrite(In1, LOW)`, `digitalWrite(In2, HIGH)` и 1 сек в другую: `digitalWrite(In1, HIGH)`, `digitalWrite(In2, LOW)`.
3. Отключаем двигатель M2.

Самостоятельно измените эту программу для управления двигателем M1.

Листинг 5.1. Тестовая программа управления двигателями

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  In1 = 5;
  In2 = 4;
  In3 = 3;
  In4 = 2;
  //Переведем данные пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
}
//Тело программы
void loop() {
  //Отключим оба двигателя.
  digitalWrite(In1, LOW); //двигатель M2.
  digitalWrite(In2, LOW);
  digitalWrite(In3, LOW); //двигатель M1.
  digitalWrite(In4, LOW);
  delay(1000); // Ждем 1 сек.
```

```
  //Включим двигатель M2.
  digitalWrite(In1, LOW);  //двигатель M2.
  digitalWrite(In2, HIGH);
  delay(1000); // Ждем 1 сек.
  // Вращаем двигатель M2 в другую сторону.
  digitalWrite(In1, HIGH);
  digitalWrite(In2, LOW);
  delay(1000); // Ждем 1 сек.
   // Проделайте те же операции для двигателя M1 самостоятельно.
  // ....
}
```

Добавляем регулирование на основе ШИМ

Добавим к существующей схеме возможность управления мощностью двигателей с помощью ШИМ (рис. 5.25), что позволит аппаратно управлять скоростью вращения вала двигателя и, соответственно, скоростью движения робота. Для этого потребуются два порта с аппаратным ШИМ, на плате они обозначены символом волнистой линии (~) — пусть это будут порты 6 и 9. Уберем перемычки (они еще пригодятся) с контактов ENA и ENB драйвера и подсоединим эти контакты к портам 6 и 9 платы Arduino.

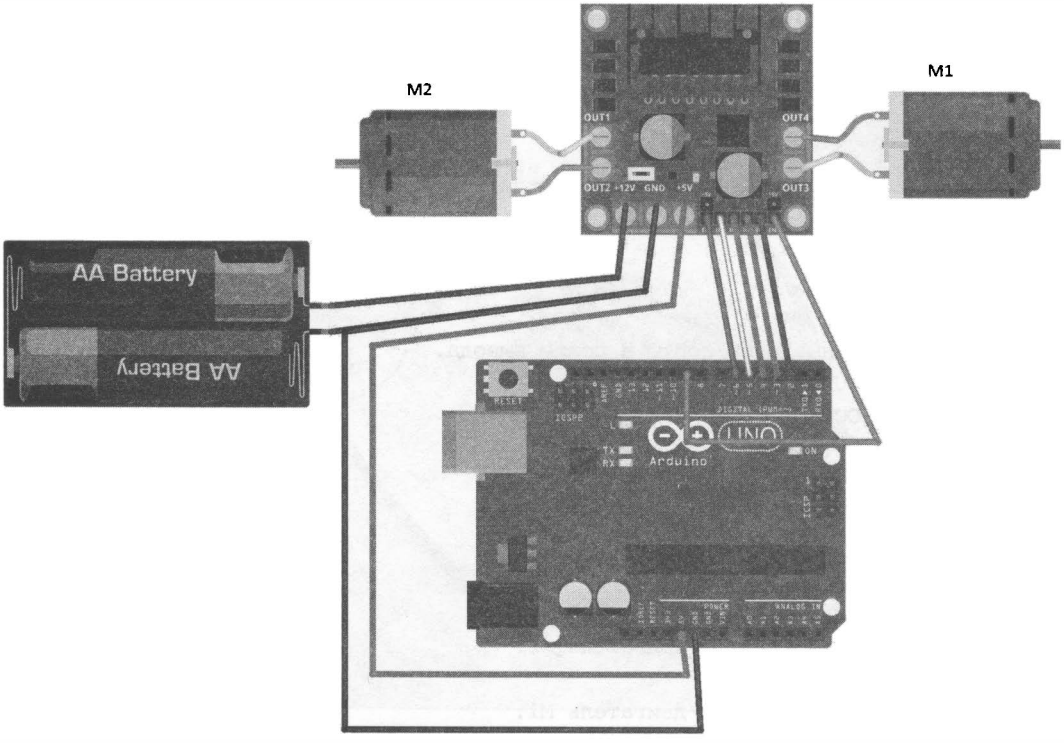


Рис. 5.25. Схема управления двигателями с регуляцией на основе ШИМ

Дождитесь, пока двигатель разгонится

Напомним, что значения ШИМ в Arduino могут изменяться от 0 до 255. На практике вал двигателя не сразу начнет вращаться — сначала двигатель станет гудеть и лишь по достижении определенного значения ШИМ начнет медленно увеличивать обороты, что связано с недостатком мощности для компенсации сил трения.

Тестовая программа управления двигателями с регуляцией на основе ШИМ

Логика работы этой тестовой программы (листинг 5.2) следующая.

В бесконечном цикле:

1. Отключаем оба двигателя подачей на оба управляющих контакта низкого сигнала.
2. Отключаем ШИМ (для чего присваиваем ШИМ значение 0).
3. После секундной паузы даем команду вращения двигателю M2 путем подачи положительного сигнала на контакт IN2.
4. Далее в цикле через 300 миллисекунд увеличиваем ширину положительного фронта ШИМ на контакте ENA от 0 до 255.
5. Затем полностью отключаем ШИМ на ENA записью 0: `analogWrite(ENA,0)`.
6. Переключаем полярность питания на двигателе M2 и аналогично начинаем в цикле увеличивать ширину ШИМ.

Итогом работы программы будет поочередное ускоряющееся вращение двигателя M2 то в одну, то в другую сторону. Самостоятельно измените программу для управления вторым двигателем M1.

Листинг 5.2. Тестовая программа управления двигателями с регуляцией на основе ШИМ

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4, ENA, ENB;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  In1 = 5; //Двигатель M2
  In2 = 4; //Двигатель M2
  In3 = 3; //Двигатель M1
  In4 = 2; //Двигатель M1
  ENA=6; //Двигатель M2
  ENB=9; //Двигатель M1
  //Переведем данные пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
```

```
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);

}
//Тело программы
void loop() {
  //Отключим оба двигателя.
  digitalWrite(In1, LOW); //двигатель M2.
  digitalWrite(In2, LOW);
  digitalWrite(In3, LOW); //двигатель M1.
  digitalWrite(In4, LOW);
  analogWrite(ENA,0); //ШИМ двигателя M2 полностью выключен.
  analogWrite(ENB,0); //ШИМ двигателя M1 полностью выключен.

  delay(1000); // Ждем 1 сек.
  //Включим двигатель M2.
  digitalWrite(In1, LOW);  //двигатель M2.
  digitalWrite(In2, HIGH);
  for(int i=0;i<255;i++)
  {
    analogWrite(ENA,i);
    delay(300); // Ждем 0.3 сек.
  }
  analogWrite(ENA,0); // ШИМ двигателя M2 полностью выключен.

  // Вращаем двигатель M2 в другую сторону.
  digitalWrite(In1, HIGH);
  digitalWrite(In2, LOW);
  for(int i=0;i<255;i++)
  {
    analogWrite(ENA,i);
    delay(300); // Ждем 0.3 сек.
  }
  analogWrite(ENA,0); // ШИМ двигателя M2 полностью выключен.
  delay(1000); // Ждем 1 сек.
   // Проделайте те же операции для двигателя M1 самостоятельно.
  // ....
}
```

Регулирование скорости вращения без использования аппаратного ШИМ

Регулировать скорость вращения колес можно и без использования аппаратного ШИМ. Для этого вернем схему в состояние, показанное на рис. 5.23–5.24 (не забудьте установить перемычки на контакты ENA и ENB драйвера), и загрузим программу, приведенную в листинге 5.3.

Отличие этой программы от предыдущей в том, что значение ШИМ на контактах ENA и ENB всегда максимально (на них постоянное напряжение 5 вольт), а скорость вращения двигателей регулируется программно путем краткосрочного включения или отключения вращения двигателей в заданном направлении.

В бесконечном цикле для левого двигателя это выглядит так:

1. Отключаем двигатель M2 подачей низкого сигнала на контакты IN1 и IN2, ждем 1 секунду.
2. В цикле `for` по переменной `i` от `0` до `long_c` (в программе это 100) начинаем отключать и включать вращение двигателя в одном направлении, при этом время, когда двигатель выключен, задается командой `delay(long_c - i)`, а когда включен — `delay(i)`. Так как значение переменной `i` в цикле каждый повтор увеличивается, то время, в течение которого двигатель выключен, постепенно уменьшается, а время работы двигателя увеличивается. Это приводит к плавному увеличению скорости вращения вала.
3. Затем та же процедура повторяется для вращения в обратную сторону.

Теперь измените программу для управления двигателем M1 самостоятельно.

Листинг 5.3. Тестовая программа управления скоростью вращения двигателей без использования регулирования на основе аппаратного ШИМ

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  In1 = 5; //Двигатель M2
  In2 = 4; //Двигатель M2
  In3 = 3; //Двигатель M1
  In4 = 2; //Двигатель M1

  //Переведем эти пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
}
//Тело программы
void loop() {
  int i;
  int long_c;
  //Остановить оба двигателя.
  digitalWrite(In1, LOW); //двигатель M2.
  digitalWrite(In2, LOW);
  digitalWrite(In3, LOW); //двигатель M1.
```

```
  digitalWrite(In4, LOW);
  delay(1000); // Ждем 1 сек.
  long_c = 100;
  for (i = 0; i <= long_c; i++)
  {
    //Остановить двигатель M2.
    digitalWrite(In1, LOW); //двигатель M2.
    digitalWrite(In2, LOW);
    delay(long_c - i); // Ждем (long_c-i)*0.001 сек.
    //Включим двигатель M2.
    digitalWrite(In1, LOW);  //двигатель M2.
    digitalWrite(In2, HIGH);
    delay(i); // Ждем (i*0.001) сек.
  }
  delay(3000);

  // Вращаем двигатель M2 в другую сторону.
  for (i = 0; i <= long_c; i++)
  {
    //Остановить двигатель M2.
    digitalWrite(In1, LOW); //двигатель M2.
    digitalWrite(In2, LOW);
    delay(long_c - i); // Ждем (long_c-i)*0.001 сек.

    //Включим двигатель M2.
    digitalWrite(In1, HIGH);  //двигатель M2.
    digitalWrite(In2, LOW);
    delay(i); // Ждем (i*0.001) сек.//
  }
  delay(3000);
  // Проделайте те же операции для второго двигателя M1 самостоятельно.
  // ....
}
```

Выводы

Итак, тип ходовой выбран — это колесная ходовая часть с четырьмя (или двумя) ведущими колесами. Выбран и используемый тип двигателей — это двигатель постоянного тока с понижающим редуктором. Рассмотрен способ управления двигателями посредством специального драйвера. Определен порядок подключения и управления двигателями.

Для закрепления теоретических знаний собраны и опробованы на практике различные схемы управления двигателями постоянного тока. Следующий шаг — сборка базовой модели робота.

ГЛАВА 6
СБОРКА БАЗОВОЙ МОДЕЛИ

Сборка робота — процесс не сложный, но требует внимания, так как неправильное подсоединение проводов может привести к порче электронных компонентов. А неправильное крепление двигателей — например, сильная затяжка винтов, может привести к их заклиниванию.

Минимальный комплект

Итак, нами выбрана база с четырьмя моторами постоянного тока с редукторами и колесами под эти редукторы, корпусом из пластика, необходимыми крепежными элементами и соответствующей управляющей логикой (рис. 6.1–6.4).

> ***Набор комплектующих «Мобильные роботы на базе Arduino»***
>
> В этом примере использованы комплектующие из набора «Мобильные роботы на базе Arduino» серии «Дерзай!» от издательства «БХВ-Петербург». Информация о том, как можно приобрести этот набор, размещена на сайте **www.bhv.ru**.

Наш робот состоит из двух уровней:

- нижний: двигатели, колеса, драйвер двигателей;
- верхний: источники электропитания, выключатель, плата Arduino, голова с датчиком на сервомоторе.

Небольшие изменения в структуре корпуса робота и даже количество двигателей (два или четыре) не имеют принципиального значения и не требуют внесения изменений в схему.

Если корпус изготовлен методом резки лазером из листов акрилового стекла, покрытого специальной защитной пленкой, то на пленке, как правило, остаются черные разводы от продуктов горения. Пленка также может мешать соединению деталей «в паз». Поэтому перед сборкой остатки пленки следует удалить, поддев острым лезвием (рис. 6.1).

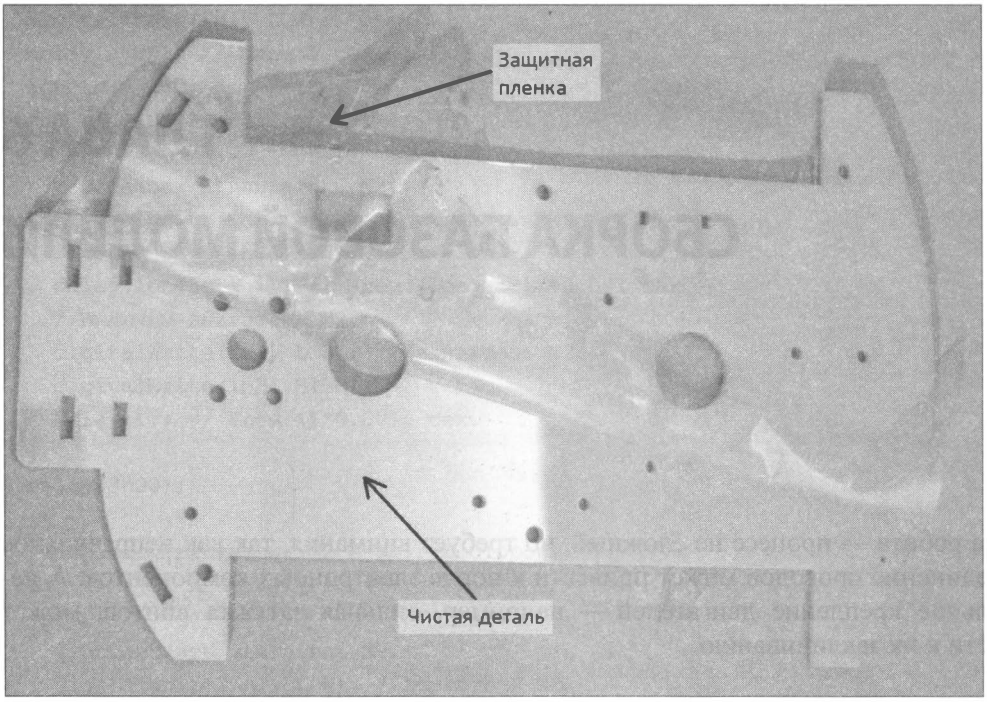


Рис. 6.1. Деталь робота из акрилового стекла с отделяемой пленкой

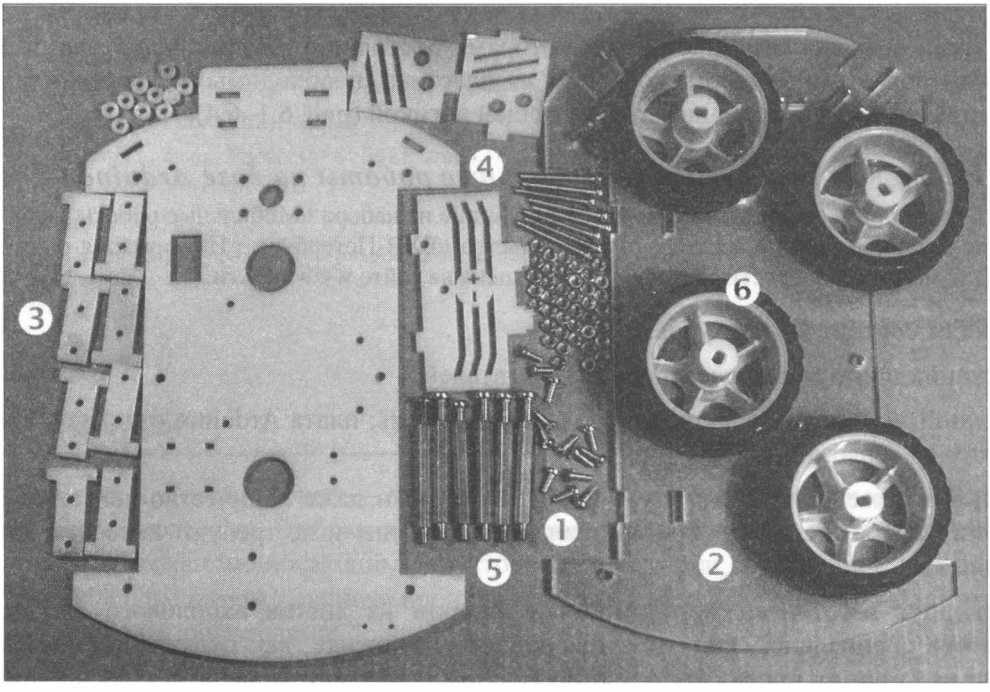


Рис. 6.2. Набор конструкционных деталей: винты и гайки (1); верх корпуса и низ (2); Т-образные элементы для установки двигателей в корпус (3); элементы для установки вращающейся «головы» с датчиком расстояния (4); стойки (5); колеса (6)

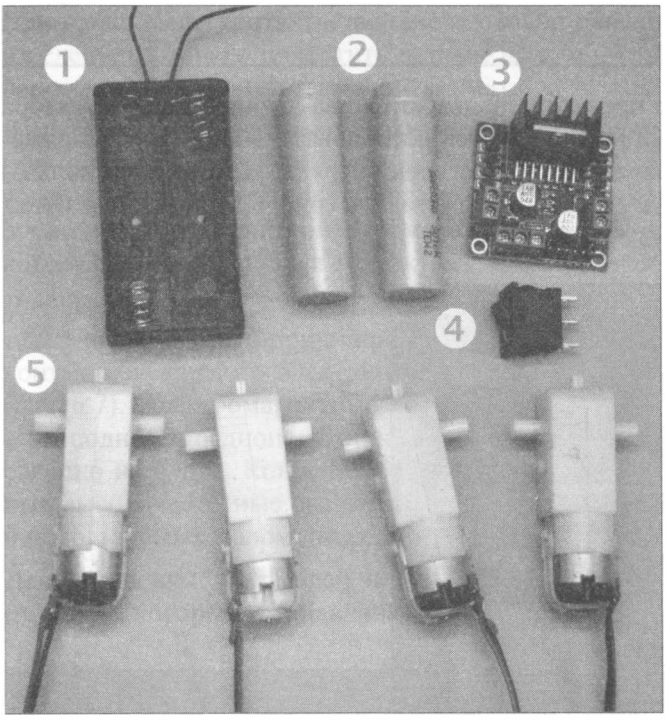


Рис. 6.3. Питание и двигатели: бокс для аккумуляторов формата 18650 (1); литиевые аккумуляторы формата 18650 (2); драйвер двигателей (3); выключатель подачи электропитания (4); моторы с редукторами (5)

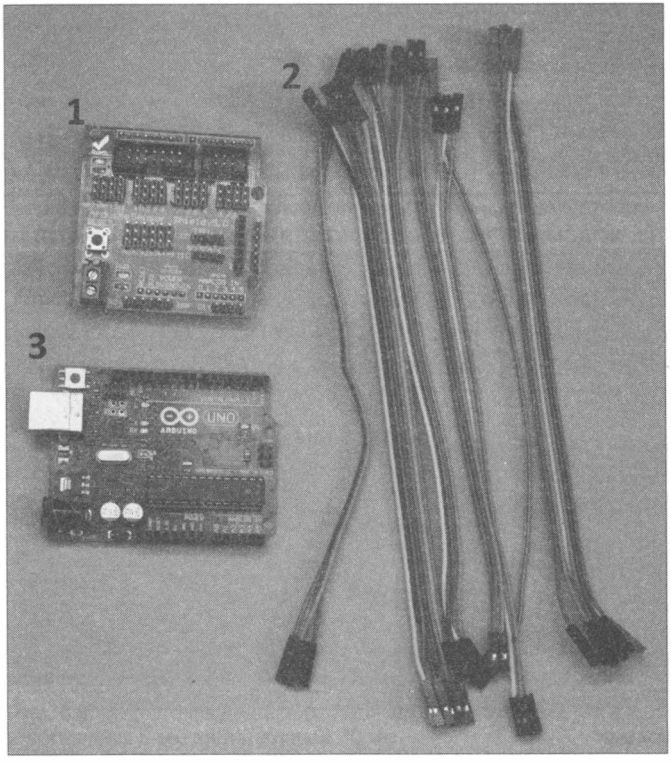


Рис. 6.4. Управляющая логика и провода: плата Arduino Sensor Shield v5.0 (1); провода с клеммами dupont (2); плата Arduino Uno (3)

Элементы питания

В качестве элементов питания в проекте используются два литиевых аккумулятора формата 18650 (см. рис. 6.3). Они подключаются последовательно и дают напряжение питания 7,2–8,4 вольта в зависимости от степени заряда. Но можно использовать и шесть батареек или аккумуляторов формата AA (рис. 6.5), при этом будет получено напряжение 7,2–9 вольт.

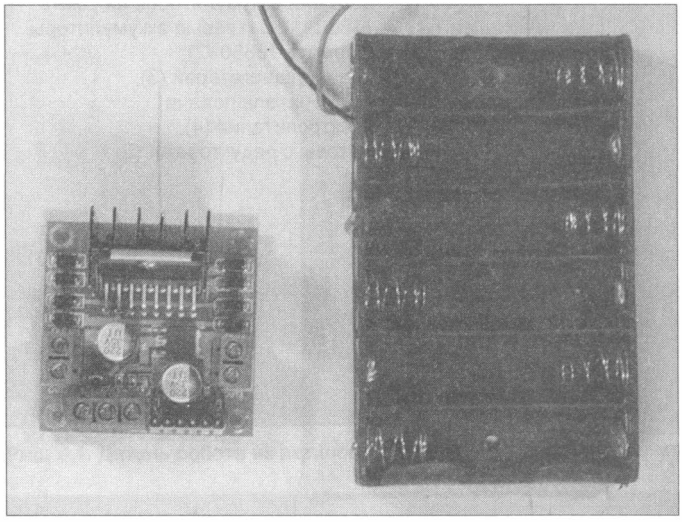


Рис. 6.5. Плата драйвера двигателей (*слева*)
и бокс для шести батарей/аккумуляторов формата AA (*справа*)

Двигатели

Перед установкой двигателей на ходовую часть следует припаять к контактным площадкам двигателей провода (рис. 6.6). Лучше использовать для этого гибкие многожильные провода разного цвета. Провода должны быть достаточной длины, чтобы не только достать до платы драйвера двигателей, но и быть аккуратно уложенными. Оставьте для них длину 12–15 см, при монтаже к плате драйвера лишнее можно будет отрезать.

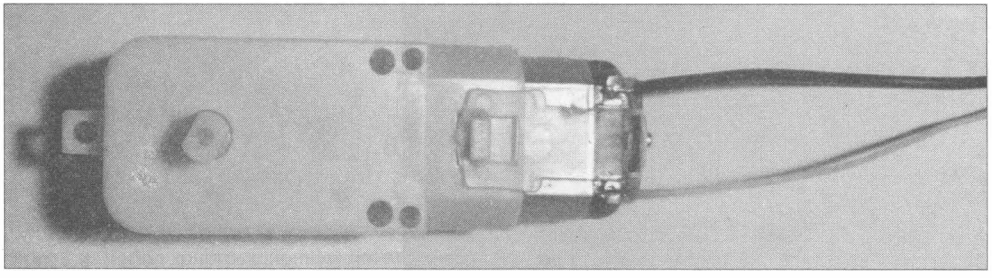


Рис. 6.6. Двигатель с припаянными проводами

При пайке, если контакты двигателя сильно окислились и не поддаются лужению, их следует немного почистить острым ножом — это ускорит процесс пайки. Но не перестарайтесь, контакты можно сломать!

Я настоятельно рекомендую при креплении проводов к двигателю использовать именно пайку, но если она для вас недоступна, можно снять изоляцию с конца провода, конец без изоляции тщательно скрутить и продеть в ушко контакта двигателя, после чего продетый конец обмотать вдоль той неизолированной части провода, которая осталась до ушка контакта.

Для исключения электромагнитных наводок, которые возникают при работе двигателей и могут стать причиной сбоев в работе электроники робота, следует между контактами каждого электрического двигателя впаять керамический конденсатор (рис. 6.7). Электромагнитные наводки могут влиять на радиоприем — например, если поднести радиоприемник к работающему двигателю, то ничего, кроме помех, слышно не будет. Конденсатор же, благодаря своим свойствам, поглощает подобные высокочастотные скачки электрического напряжения. Для напряжения 5–12 В и используемых маломощных моторов достаточно конденсатора емкостью 100 нФ.

Для установки двигателей в корпус робота применяются Т-образные элементы (рис. 6.8), которые устанавливаются снизу корпуса в пазы (рис. 6.9). Каждый двига-

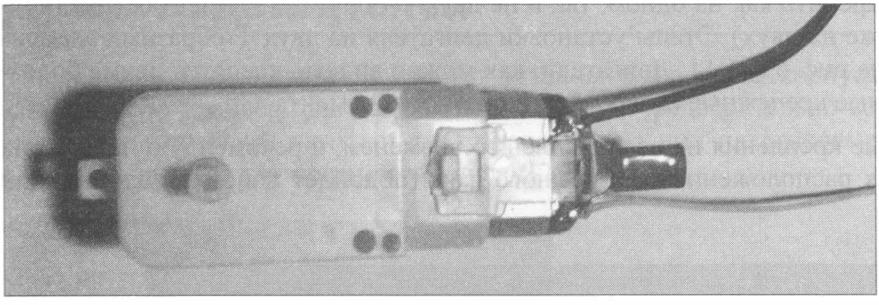


Рис. 6.7. Двигатель с припаянными проводами и конденсатором

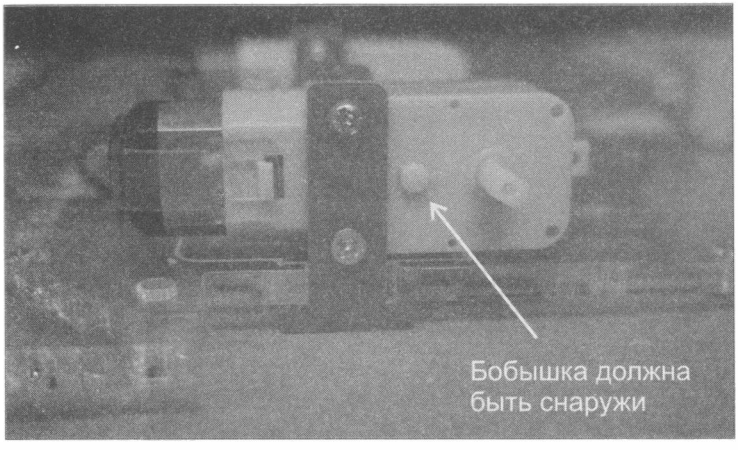


Рис. 6.8. Подготовка двигателя к монтажу: на двигатель установлен внешний Т-образный элемент и вставлены 3-мм винты длиной 30 мм

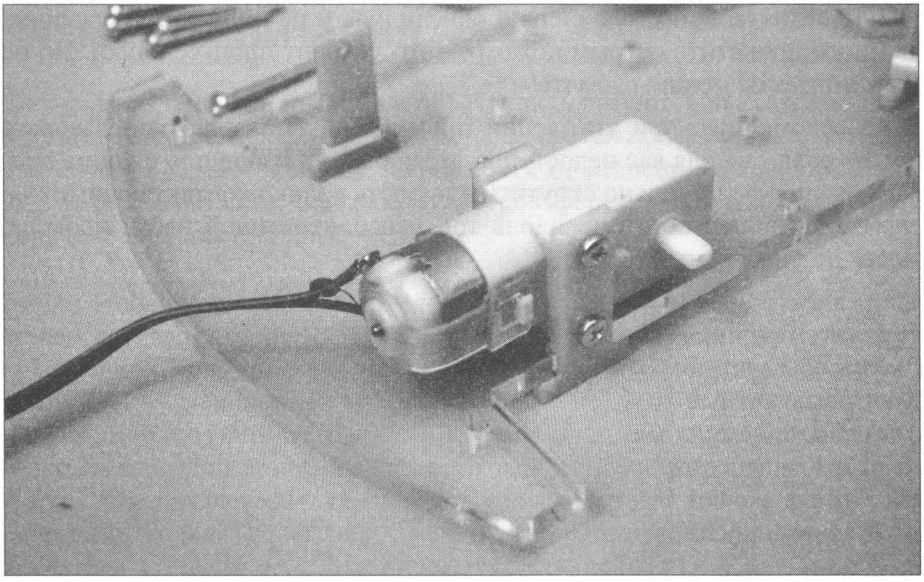


Рис. 6.9. Установка двигателя на Т-образные крепления в корпус робота

тель можно закрепить как на одном, так и на двух Т-образных элементах (предпочтительнее все же на двух). Этапы установки двигателя на двух Т-образных элементах показаны на рис. 6.8–6.11. Двигатели, как можно видеть, крепятся двумя болтами в специальные крепежные отверстия Т-образного элемента.

Чтобы винтовые крепления не ослаблялись со временем, я рекомендую нанести на винты в местах расположения гаек немного лака (подойдет цапонлак или лак для ногтей).

Рис. 6.10. Фиксируем винты

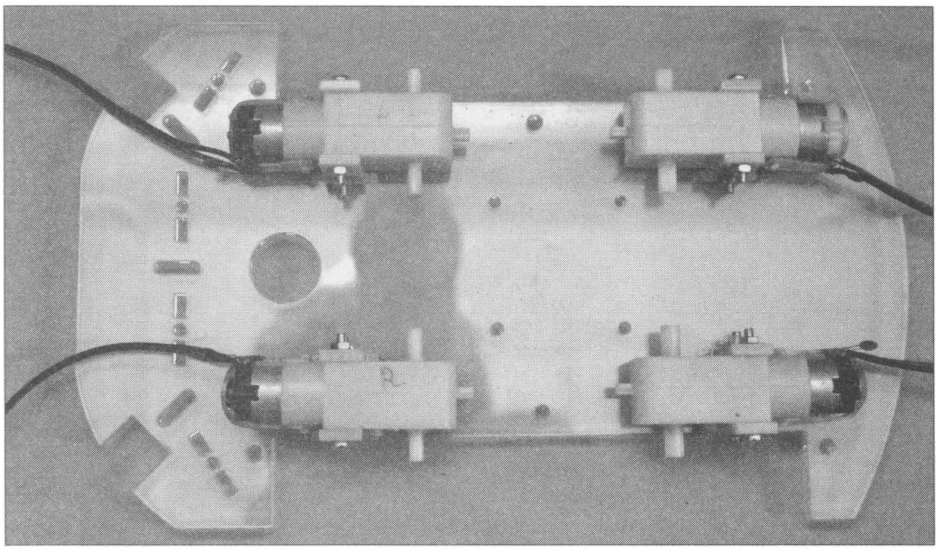


Рис. 6.11. Все двигатели смонтированы

Драйвер двигателей

После установки двигателей следует установить на плате корпуса робота плату, содержащую драйвер двигателей (типа L298N). Предварительно вставьте в отверстия платы драйвера винты и снизу наденьте на них пластиковые шайбы, иначе можно повредить плату драйвера (рис. 6.12).

Рис. 6.12. Плата драйвера со вставленными винтами длиной 14 мм и пластиковыми шайбами

В плате корпуса робота просверлены специальные отверстия для крепления платы драйвера (рис. 6.13) — устанавливать плату драйвера следует именно на них.

Плата драйвера крепится четырьмя винтами в эти отверстия на корпусе (рис. 6.14). После установки платы переверните корпус и затяните гайки (рис. 6.15).

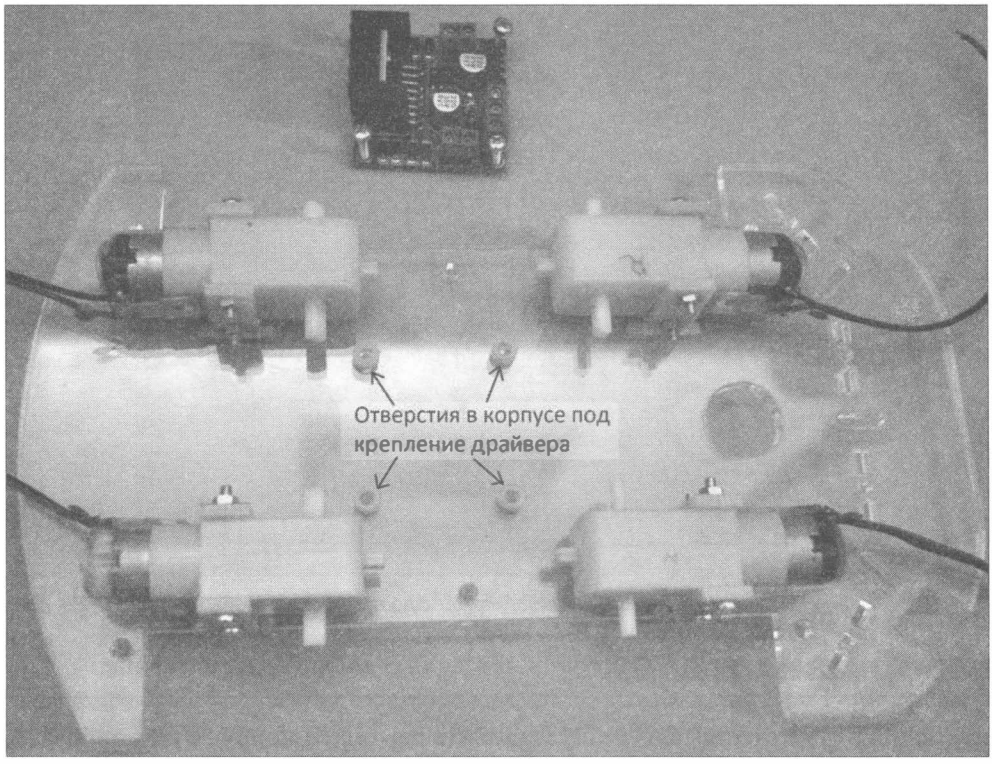


Рис. 6.13. Расположение отверстий для крепления драйвера на корпусе робота

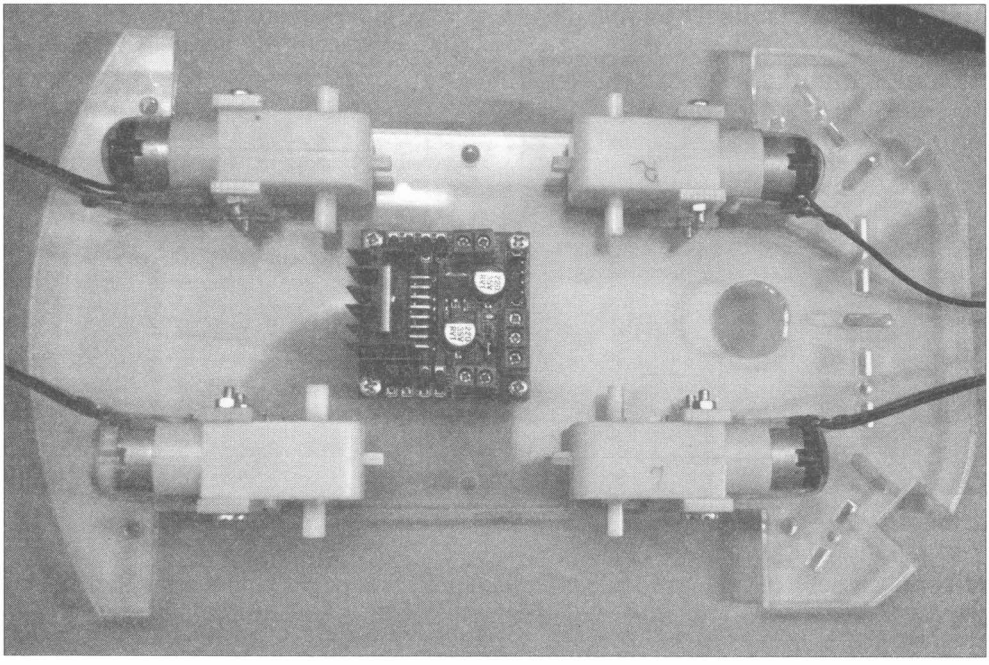


Рис. 6.14. Монтаж платы драйвера: вид сверху

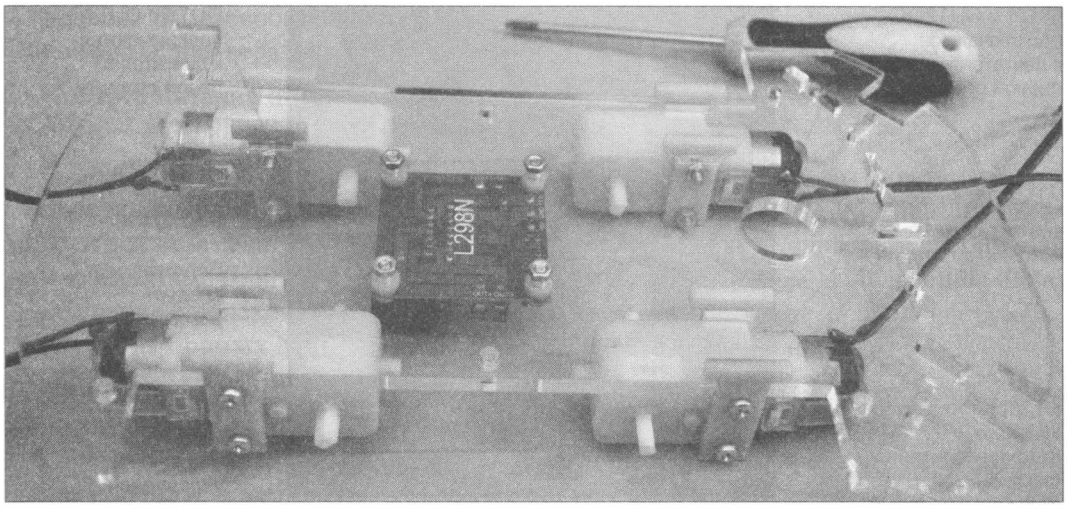


Рис. 6.15. Монтаж платы драйвера: вид снизу

Соединение платы драйвера и двигателей

Пояснения к установке и настройке платы драйвера L298N представлены на рис. 6.16.

> ***Соблюдайте порядок подключения!***
>
> Важно соблюдать указанный на рис. 6.16 порядок подключения! Все программы управления движением робота, приведенные в этой книге, ориентированы именно на такую схему. Если вы при монтаже поменяете местами двигатели и контакты управления, вам придется самостоятельно вносить изменения в управляющие программы по всей книге, что может привести к путанице.

Контакты OUT1 и OUT2 подключаем к двигателям правой стороны робота, OUT3 и OUT4 — к двигателям его левой стороны. Клемма +12V подключается к основному электропитанию робота, клемма GND («земля») — к отрицательному выводу источника питания (в дальнейшем для всех используемых приборов контакты GND должны быть объединены). Клемма +5V будет использована для электропитания платы Arduino UNO, датчиков и других маломощных, но требующих стабилизированного электропитания устройств, от нее потребуется вывести провод. Перемычки ENA и ENB включают или отключают двигатели, а также при помощи широтно-импульсной модуляции через них можно управлять скоростью вращения колес, но во втором издании книги мы их использовать не будем и оставим с установленными перемычками.

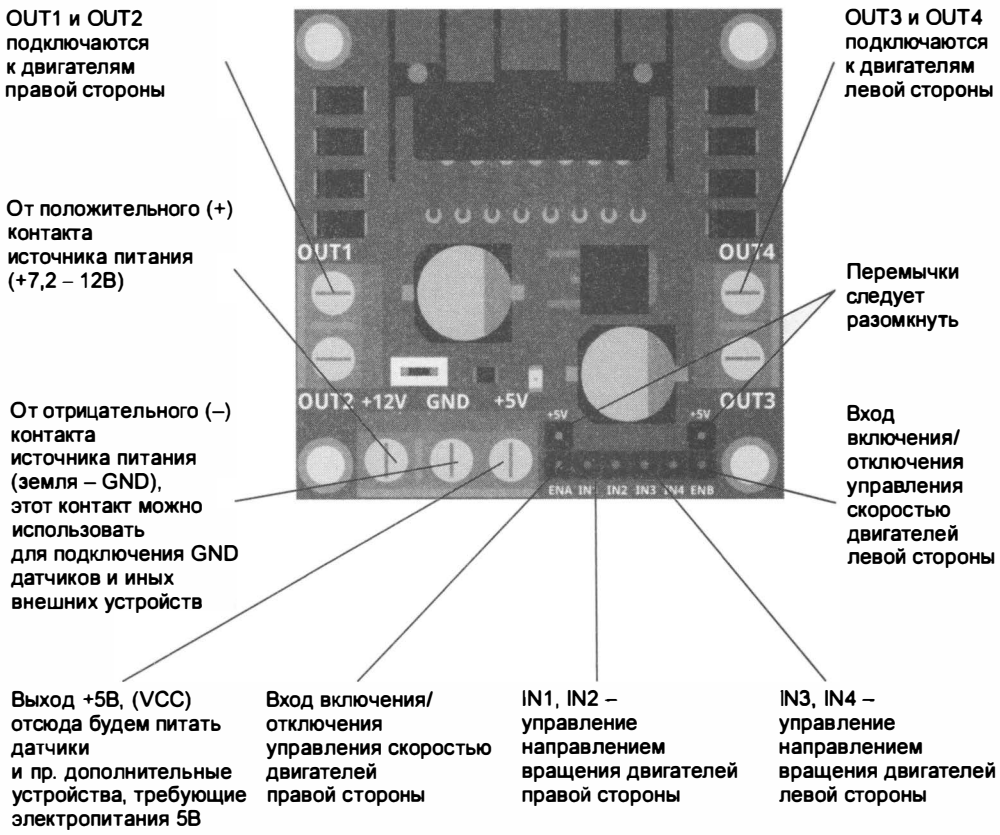


Рис. 6.16. Вид платы драйвера L298N с пояснениями

Проверка правильности подключения платы драйвера и двигателей

При установке двигателей, как показано на рис. 6.11, передний и задний двигатели каждой стороны перевернуты относительно друг друга. Это потребует перекрестного соединения проводов от них при подключении к драйверу: верхний провод переднего двигателя соединяется с нижним проводом от заднего и наоборот. При этом есть вероятность ошибки, и прежде чем приступить к сборке верхнего уровня робота, требуется провести тщательную проверку и исключить любые ошибки.

Такую проверку для уверенности лучше расписать по шагам:

1. Прежде всего убедитесь, что установлены перемычки ENA и ENB, разрешающие включение двигателей.
2. Соберите схему, показанную на рис. 5.22. Но подключите к каждому выходу драйвера по паре двигателей, как только что описано. При этом контакты IN1–IN4

должны быть свободны — никаких напряжений на них подавать не надо, т. к. контакты драйвера уже шунтированы через резисторы на «землю» (GND).

3. Подайте питание от блока питания на плату драйвера: «плюс» — на +12V, «минус» — на GND. Должен загореться светодиод.
4. Отдельным проводом (удобно использовать провод от тестера) кратковременным прикосновением подайте +5V от платы драйвера на IN1 — **правые** двигатели должны начать вращаться. Если что-то пошло не так, проверьте соединения и контакты. Двигатели также должны вращаться в одном направлении. Если вращение происходит в разных направлениях — проверьте правильность объединения **правых** двигателей в пару и объедините правильно.

   Кроме того, обратите внимание, что при подаче «плюса» на IN1 колеса должны вращаться **назад**, а при подаче «плюса» на IN2 — **вперед**. Если происходит обратное — поменяйте местами провода, подводимые к драйверу исследуемой пары двигателей.
5. Так же проверьте и настройте **левые** двигатели. При подаче «+5V» на IN4 **левые** колеса должны вращаться **вперед**, а при подаче «плюса» на IN3 — **назад**.

Контакты IN1, IN2, IN3, IN4 управляют подачей электропитания на соответствующие по номерам выводы OUT1, OUT2, OUT3, OUT4. Например, подача логического нуля на IN1 подключит OUT1 на «землю» (GND), а подача логической единицы на IN2 подключит OUT2 к положительному контакту электропитания (обозначен на рис. 6.16 как +12V) — это вызовет вращение подключенного двигателя. Если же на IN1 подать логическую единицу, а на IN2 — ноль, то вал двигателя начнет вращаться в противоположную сторону (подробнее об этом см. в *главе 5*).

Провода от двигателей соединяются с платой драйвера через винтовой зажим (рис. 6.17).

Учтите также, что каждый выход L298N будет управлять сразу парой двигателей, подключенных параллельно, и, перепутав полярность, можно получить колеса, вращающиеся в разные стороны! Для соблюдения полярности рекомендую к контактным площадкам, расположенным ближе к корпусу, припаять по черному проводу, а к верхним — по белому. Контактные площадки передних и задних двигателей перевернуты относительно друг друга (это видно на рис. 6.15), поэтому следует при подключении к драйверу черный и белый провода от разных двигателей одной стороны объединить (рис. 6.18).

Порядок подключения проводов от первой пары двигателей к контактам OUT1–OUT2 и OUT3–OUT4, конечно, имеет значение. В нашем случае:

- верхний контакт переднего правого двигателя и нижний контакт заднего правого двигателя следует подключить к OUT1;
- нижний контакт переднего правого двигателя и верхний контакт заднего правого — к OUT2;
- верхний контакт переднего левого двигателя и нижний контакт заднего левого двигателя следует подключить к OUT4;

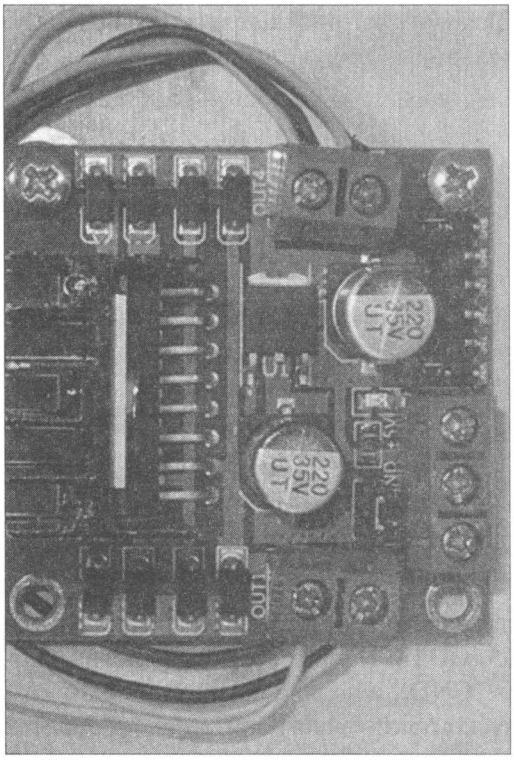


Рис. 6.17. Провода от двигателей присоединены к плате драйвера

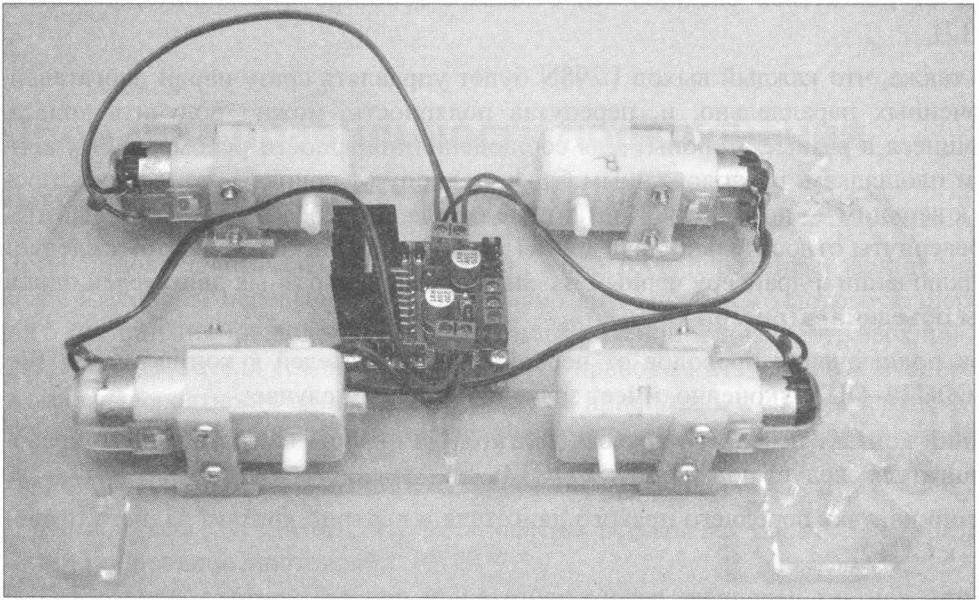


Рис. 6.18. Двигатели робота подключены к драйверу

- нижний контакт переднего левого двигателя и верхний контакт заднего левого — к OUT3.

> ***Важно!***
>
> Обратите внимание — контактные площадки всех двигателей должны быть направлены к центру ходовой части робота, а бобышки (см. рис. 6.8) смотреть наружу.

Чтобы провода не болтались, удобно прикрепить их к корпусу клеем (клеевым пистолетом) либо стяжками.

После того, как полярность двигателей проверена, и в случае ошибки исправлена, можно приступать к сборке верхнего уровня.

Перед этим установим в контакты EN1–EN4 драйвера провода со съемными клеммами («мама-мама»), а в винтовые зажимы +12, GND и 5V зажмем провода длиной примерно 15 см (рис. 6.19).

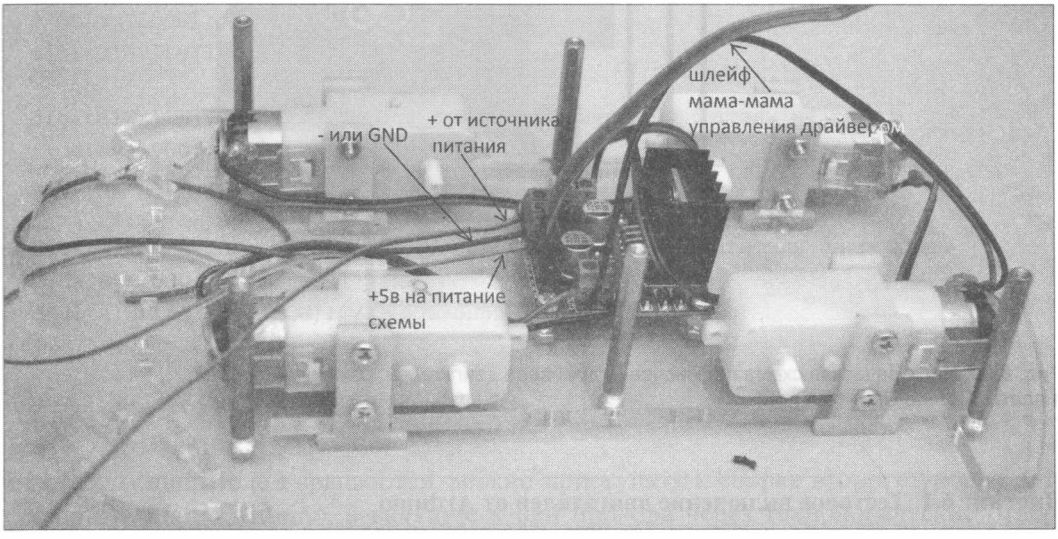


Рис. 6.19. Нижний уровень корпуса робота в сборе с установленными проводами и стойками

Далее рекомендую протестировать работоспособность сборки, присоединив к драйверу плату Arduino по схеме, приведенной на рис. 6.20.

Для тестирования сначала присоедините провода к плате драйвера двигателей. Используйте для этого провода разного цвета. После соединения запишите для себя на бумаге, какого цвета провод к какому контакту платы драйвера подключен, — это поможет избежать путаницы. Провода можно паять или использовать клеммы. Если вы еще не очень хорошо паяете, то рекомендую использовать провода с клеммами, которые быстрее в монтаже. Присоединив провода к плате драйвера, подключите, как показано на рис. 6.20, плату Arduino и загрузите в нее программу из листинга 6.1.

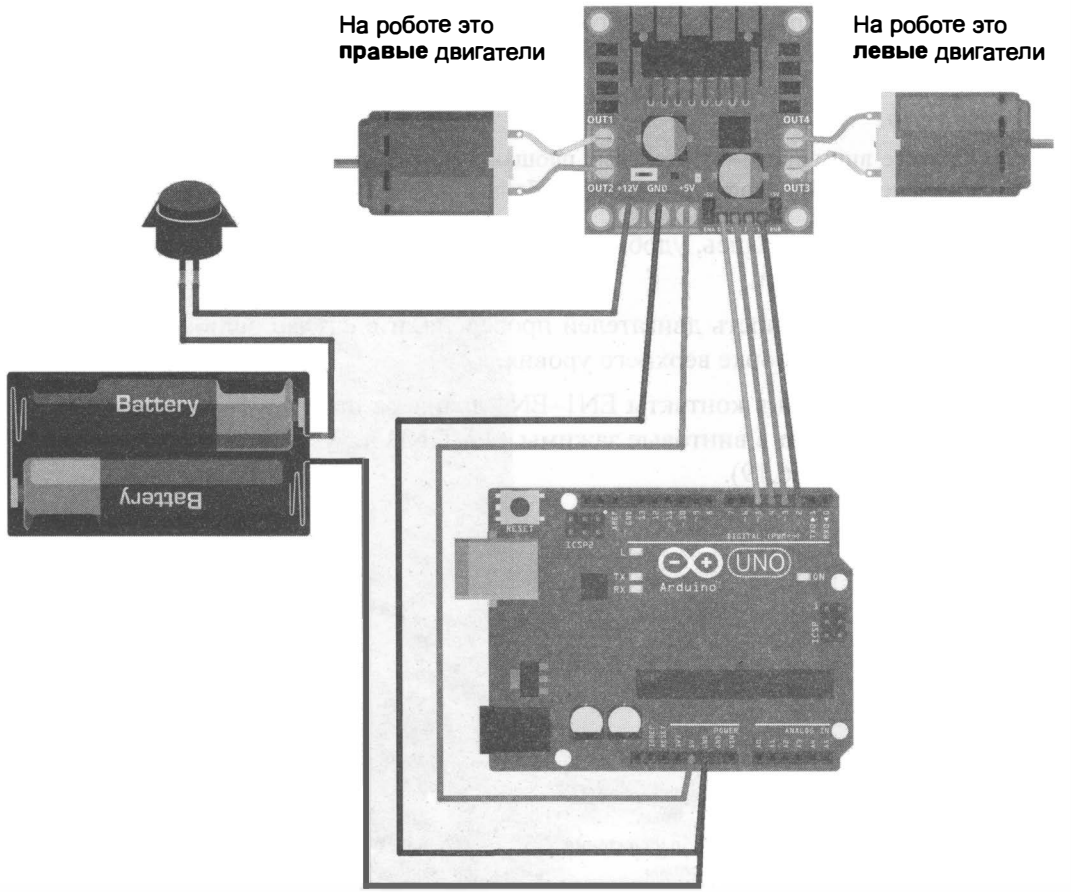


Рис. 6.20. Электрическая схема подключения драйвера двигателей, самих двигателей и платы Arduino Uno

Листинг 6.1. Тестовое включение двигателей от Arduino

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  In1 = 2;
  In2 = 3;
  In3 = 4;
  In4 = 5;
  //Переведем эти пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
```

```
  pinMode(In4, OUTPUT);
}
//Тело программы
void loop() {
  //Остановить все двигатели.
  digitalWrite(In1, LOW); //Правые двигатели на роботе.
  digitalWrite(In2, LOW);
  digitalWrite(In3, LOW); //Левые двигатели на роботе.
  digitalWrite(In4, LOW);
  delay(1000); // Ждем 1 сек.

//Включить-выключить поочередно левые и правые двигатели
  digitalWrite(In1, HIGH); //Включили один правый двигатель на роботе.
    delay(1000); // Ждем 1 сек.
  digitalWrite(In1, LOW); //Выключили

  digitalWrite(In2, HIGH); //Включили другой правый двигатель на роботе.
    delay(1000); // Ждем 1 сек.
  digitalWrite(In2, LOW); //Выключили

  digitalWrite(In3, HIGH); //Включили один левый двигатель на роботе.
    delay(1000); // Ждем 1 сек.
  digitalWrite(In3, LOW); //Выключили

  digitalWrite(In4, HIGH); //Включили другой левый двигатель на роботе.
    delay(1000); // Ждем 1 сек.
  digitalWrite(In4, LOW); //Выключили
}


}
```

Если валы двигателей вращаются, можно приступать к сборке второго уровня. Отсоедините плату Arduino, а ведущие к ней провода через технологические отверстия пластины корпуса выведите наверх, т. к. они должны будут позже подсоединены к плате Arduino, которая стационарно монтируется на втором уровне робота.

Верх корпуса

Крепление верхней части корпуса осуществляется с помощью бронзовых стоек, закрепляемых на винты (см. рис. 6.19). Установим стойки, временно отложим собранную нижнюю часть робота в сторону и подготовим необходимые элементы для сборки верхнего уровня (рис. 6.21).

Прикрепим плату Arduino винтами к корпусу, как показано на рис. 6.22, не забыв проложить под плату изолирующие шайбы. Закрепим бокс для аккумуляторов, воспользовавшись для этого винтами с потайной головкой (иначе потом возникнут

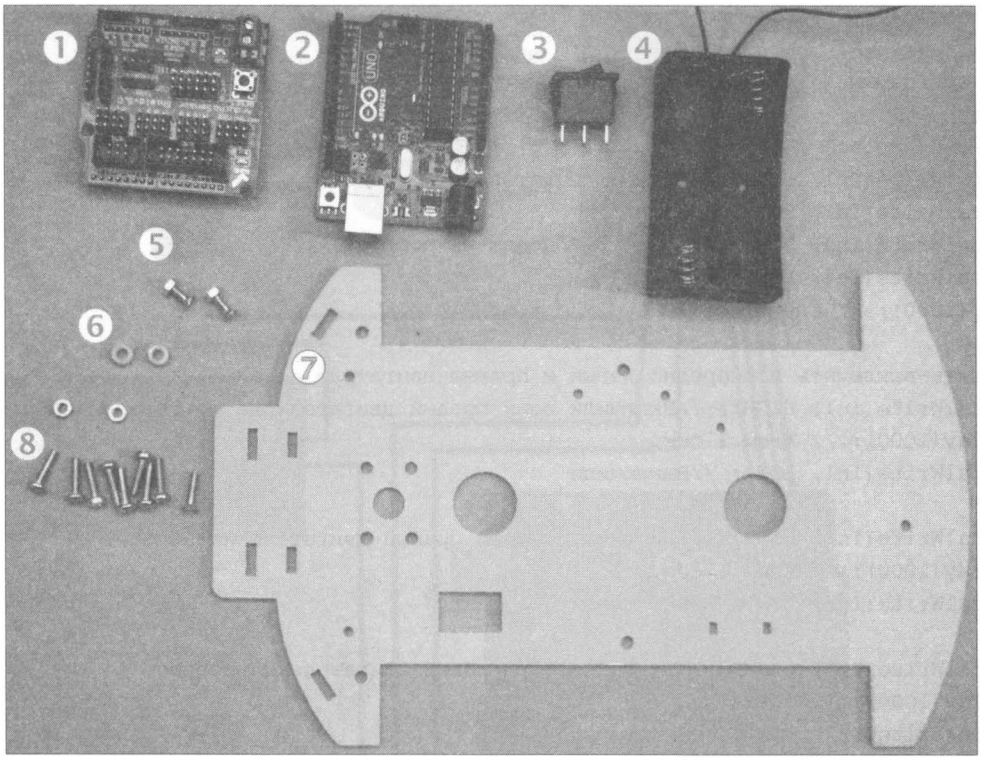


Рис. 6.21. Минимальный набор для верхнего уровня робота: плата Arduino Sensor Shield v5.0 (1); плата Arduino Uno (2); выключатель (3); бокс для аккумуляторов (4); 3-мм винты с потайными головками (5); изолирующие шайбы (6); верх корпуса (7); 3-мм винты и гайки для крепления платы Arduino и корпуса к стойкам (8)

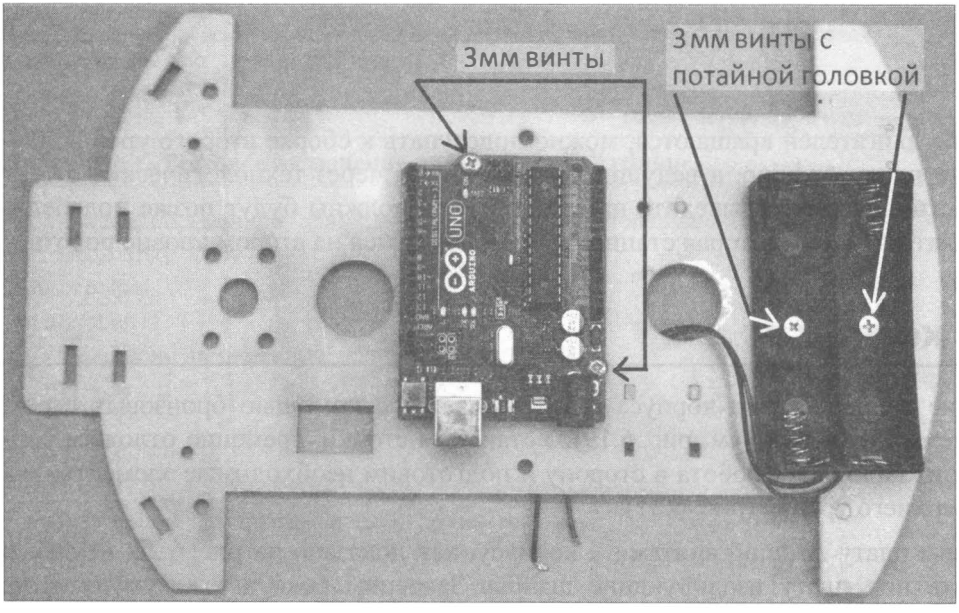


Рис. 6.22. Этап установки платы Arduino и бокса для аккумуляторов

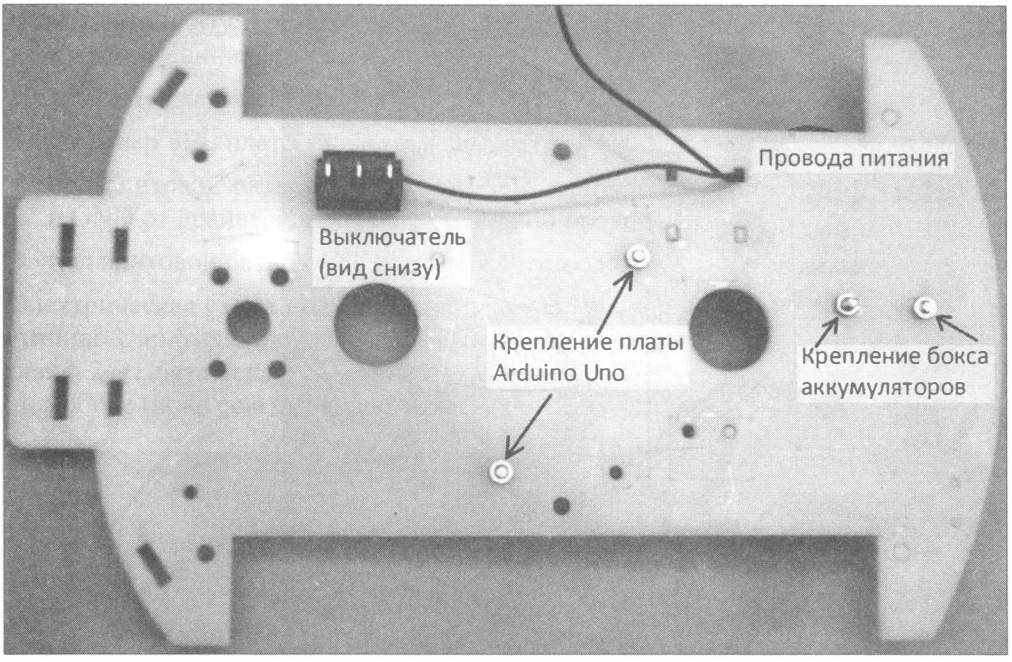


Рис. 6.23. Верх корпуса с установленным выключателем, боксом и Arduino Uno (вид снизу)

трудности с установкой в него аккумуляторов). Вместо винтов для крепления бокса можно использовать двусторонний скотч.

На следующем этапе нужно установить выключатель в прямоугольный паз (рис. 6.23). Выключатель имеет подпружиненный механизм-защелку и не будет выпадать. Если ваш выключатель трехконтактный, то средний контакт — общий, он замыкается с левым или правым контактом в зависимости от положения выключателя. Для включения/выключения робота потребуются два контакта: средний и один из крайних. Настоятельно рекомендую **припаять** провода питания к выключателю и только при невозможности пайки использовать скрутку. На рис. 6.24 правый контакт присоединен к красному проводу (+) от бокса аккумуляторов, а средний — к проводу, который мы ранее присоединили к клемме 12V драйвера. Шлейф от контактов EN1–EN4 драйвера и провода, присоединенные к GND и 5V, следует продеть в технологические отверстия верха корпуса для использования на втором уровне.

Установим верхнюю часть корпуса на стойки нижней части и закрепим их винтами (рис. 6.25). Далее осторожно, чтобы не повредить ножки, установим плату Arduino Sensor Shield v5.0 на Arduino Uno. Подсоединим выведенные наверх провода к Sensor Shield.

Стабилизатор +5 В на плате Arduino Uno недостаточно мощный, чтобы питать от него все датчики и серводвигатель «головы» робота. Поэтому поступим следующим образом. Нестабилизированное напряжение 7–12 В от аккумуляторов подадим через выключатель на вход драйвера двигателей (см. рис. 6.20). В драйвере двигателей есть существенно более мощный стабилизатор на +5 В. От него и будем

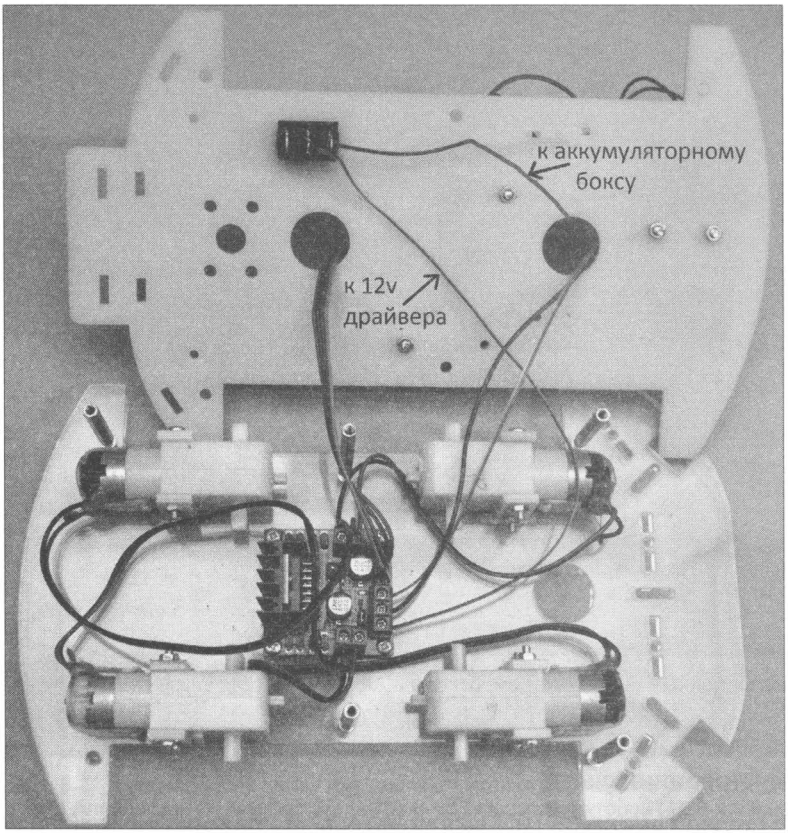


Рис. 6.24. Подготовка верхней части корпуса робота к установке (прикреплены и выведены провода)

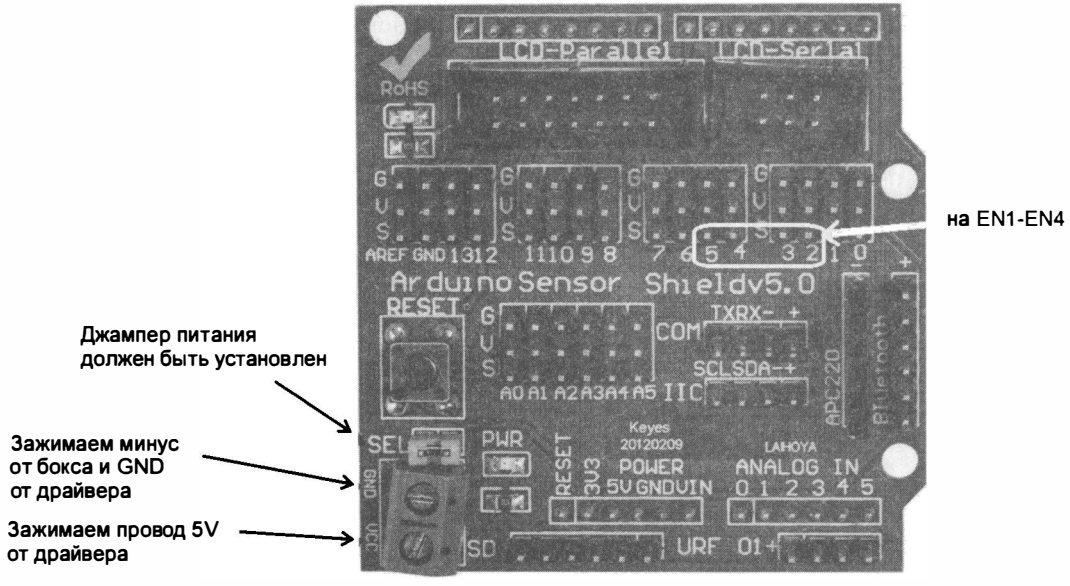


Рис. 6.25. Плата Arduino Sensor Shield v5.0 с пояснениями по закреплению проводов

запитывать плату Arduino Uno, плату Sensor Shield, а через нее сервомотор «головы» и все внешние датчики робота.

На плате Sensor Shield (рис. 6.25):

- джампер SEL платы оставляем замкнутым;
- под винтовой разъем GND зажимаем отрицательный провод от аккумуляторов и GND от драйвера двигателей (средний провод);
- под винтовой разъем VCC зажимаем провод от 5V драйвера двигателей.

Электрическая схема подключения драйвера двигателей, самих двигателей и платы Arduino Uno была приведена на рис. 6.20. Общий вид всех соединений приведен на рис. 6.24. Плата Sensor Shield не показана, чтобы не усложнять схему. Корпус робота в сборе (пока еще без колес) показан на рис. 6.26.

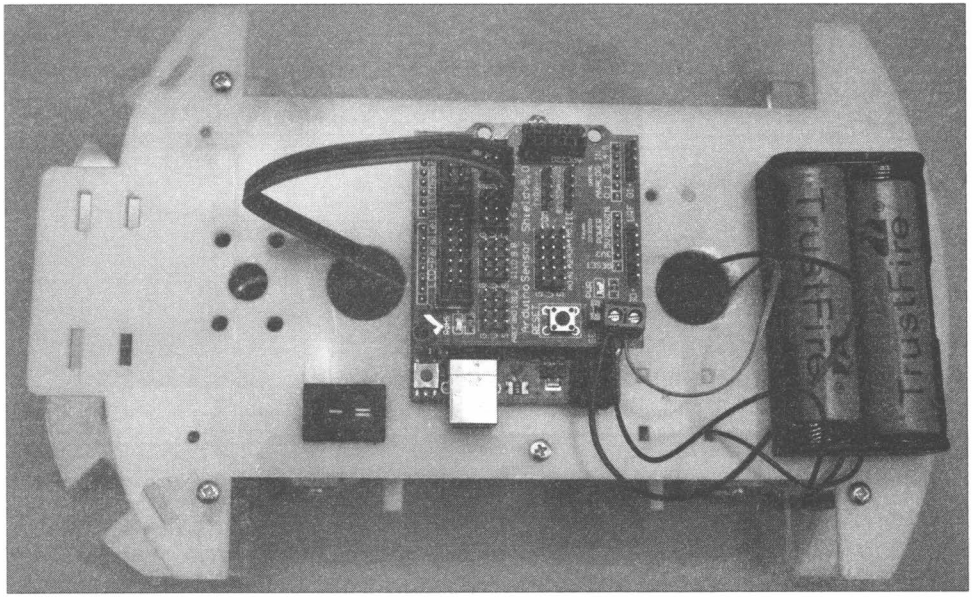


Рис. 6.26. Корпус робота в сборе (осталось установить колеса)

Установка устройств обратной связи

Рассмотрим, как можно дать роботу возможность подавать сигналы в ответ на определенные события — например, начало поворота, остановку и т. д. Для этого можно применить светодиод или зуммер, присоединенные к контактам (D2–D13 или A0–A3) платы Arduino робота.

Светодиод

Светодиод (рис. 6.27) светится, если через него протекает ток от анода к катоду. Значит, для его использования на анод светодиода следует подать положительное

напряжение, а катод заземлить (на минус). При этом светодиод будет нормально светиться лишь тогда, когда к нему приложено напряжение, соответствующее номинальному (от 1,4 до 3 вольт — зависит от типа), если же напряжение будет больше, он очень быстро выйдет из строя. Для того чтобы светодиод светил долго, применяется токоограничивающий резистор.

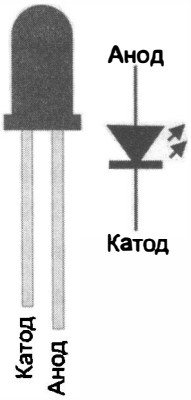


Рис. 6.27. Светодиод

Простейшая схема соединения светодиода показана на рис. 6.28. Длинная ножка светодиода — анод, короткая — катод. Длинная ножка через резистор сопротивлением 200 Ом будет подключена к контакту D6 платы Arduino, а короткая ножка посажена на «землю» (GND). Обратите внимание, что если поменять ножки светодиода местами, он не будет излучать свет.

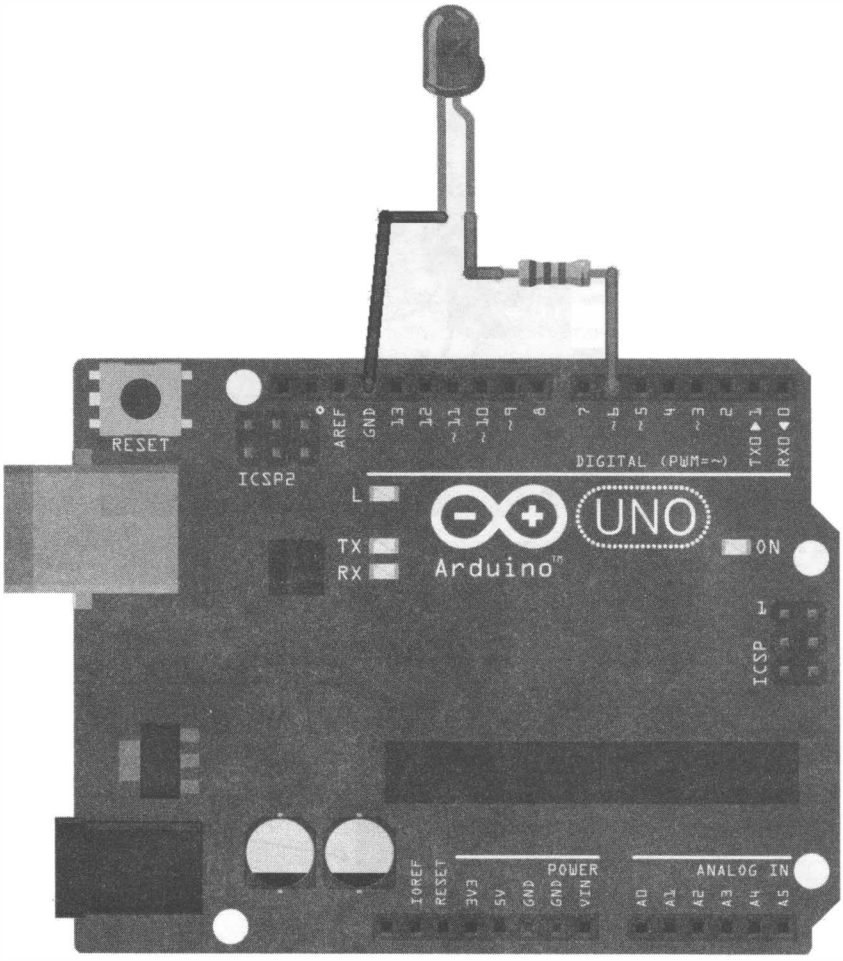
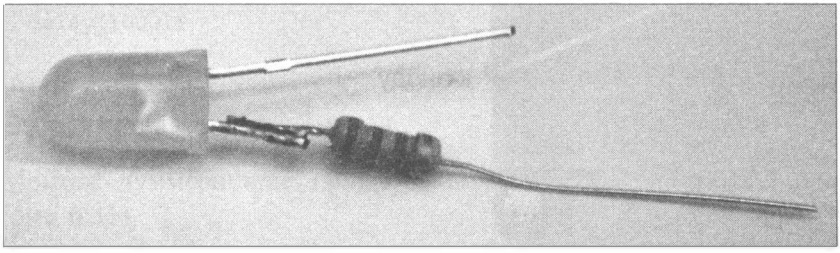
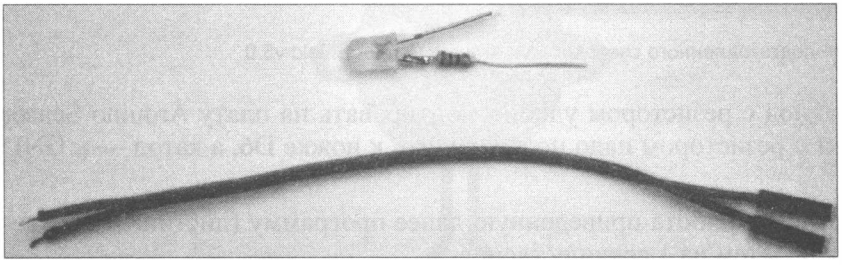


Рис. 6.28. Схема подключения светодиода

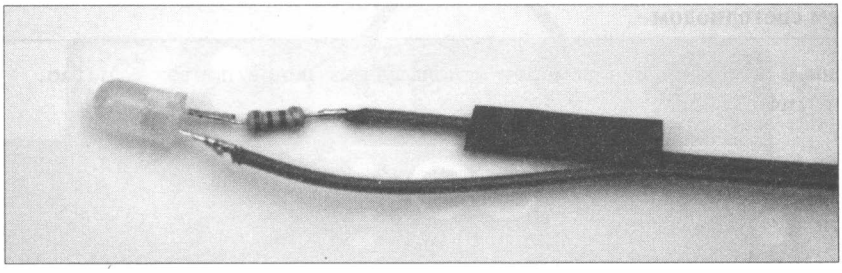
Для подключения светодиода к проводам я использовал пайку: обрезал длинную ножку и подпаял к ней резистор сопротивлением 200 Ом (рис. 6.29, *а*), затем подготовил провода с клеммами «мама» с одной стороны, а с другой стороны клеммы отрезал, зачистил и залудил кончики (рис. 6.29, *б*), далее подпаял провода к светодиоду (рис. 6.29, *в*) и изолировал контакт с резистором, надев на него термотрубку — она сначала широкая, но после нагрева сильно стягивается, что очень удобно (рис. 6.29, *г*).



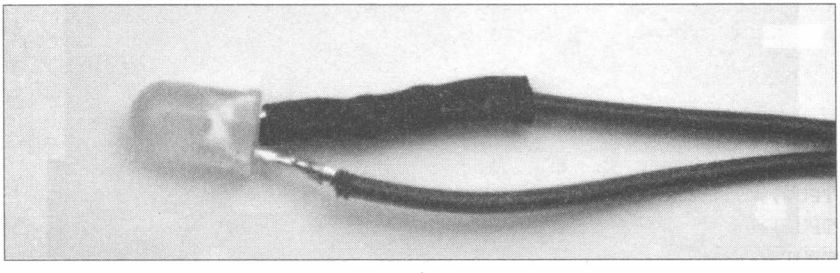
а



б



в



г

Рис. 6.29. Пайка светодиода к проводам

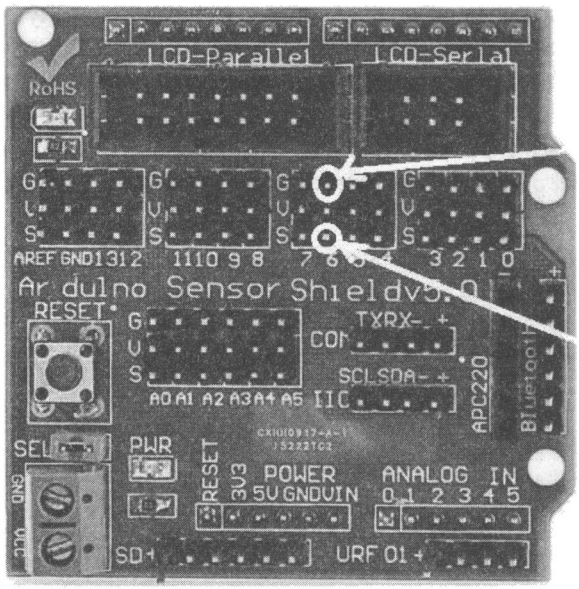


Рис. 6.30. Подключение подготовленного светодиода к Arduino Sensor Shield v5.0

Полученный светодиод с резистором удобно монтировать на плату Arduino Sensor Shield v5.0: контакт с резистором надо подсоединить к ножке D6, а катод — к GND (рис. 6.30).

Для проверки загрузим в робота приведенную далее программу (листинг 6.2). Светодиод 10 раз мигает, затем на 1 секунду гаснет.

Листинг 6.2. Мигаем светодиодом

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4, InDiod;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  InDiod = 6;
  pinMode(InDiod, OUTPUT);
  digitalWrite(InDiod, HIGH);
  In1 = 2;
  In2 = 3;
  In3 = 4;
  In4 = 5;

  //Переведем эти пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
}
```

```
//Тело программы
void loop() {
 for (int i = 0; i < 10; i++)
  {
    digitalWrite(InDiod, HIGH);
    delay(200);
    digitalWrite(InDiod, LOW);
    delay(200);
  }
  delay(1000);
}
```

Зуммер

Монтаж зуммера еще проще — ему не нужен токоограничивающий резистор (рис. 6.31).

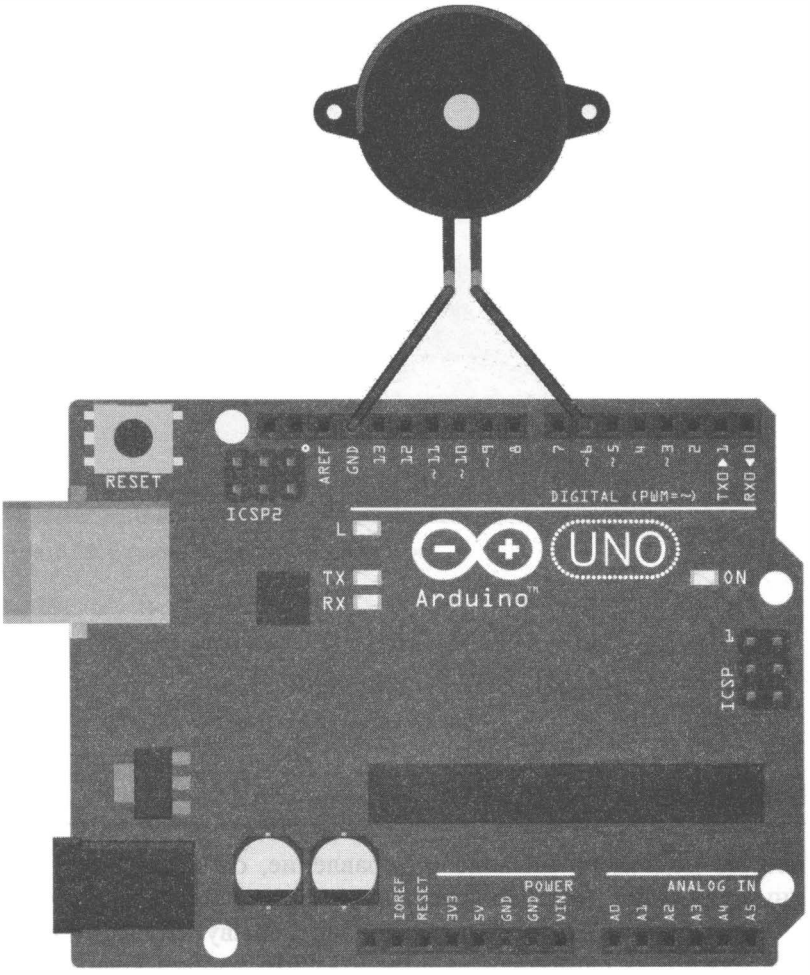


Рис. 6.31. Подключение зуммера к Arduino

Для тестирования работы зуммера можно использовать программу из листинга 6.3. Контакт зуммера (в данном случае это ножка 6) переводим в режим вывода, а затем командой `tone(номер пина, частота)` заставляем его издавать писк. Команда `noTone(номер пина)` отменяет звук. Отмечу, что можно использовать только один зуммер для Arduino одновременно. В программе зуммер, прерываясь, гудит на частоте 1000 Гц в течение 2 секунд, затем на 1 секунду замолкает.

Листинг 6.3. Тестирование зуммера

```
//Создадим переменные для хранения номеров используемых пинов/портов Arduino.
int In1, In2, In3, In4, InZ;
//Настройка
void setup() {
  // Присвоим переменным номера пинов Arduino.
  InZ = 6;
  pinMode(InZ, OUTPUT);
  In1 = 2;
  In2 = 3;
  In3 = 4;
  In4 = 5;

  // Переведем эти пины/порты в режим вывода.
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(In3, OUTPUT);
  pinMode(In4, OUTPUT);
}
//Тело программы
void loop() {
  for (int i = 0; i < 10; i++)
  {
    tone(InZ, 1000);
    delay(100);
    noTone(InZ);
    delay(100);
  }
  delay(1000);
}
```

Укладка проводов

Когда к роботу полностью подведено питание и управление, он скрывается под ворохом незакрепленных проводов. Болтающиеся провода, как правило, за что-нибудь цепляются, что приводит к их отсоединению или обрыву. Во избежание таких проблем провода нужно объединять и укладывать. Это удобно делать при помощи маленьких стяжек или изоленты. Можно также использовать термоклей —

для этого на корпус переносится капля термоклея, а затем в нее утапливаются провода. Если потом провода потребуется освободить, то термоклей нужно нагреть до 200 градусов, например, феном.

Выводы

Робот собран. Теперь следует сделать его контрольный осмотр, проверить, нет ли оторванных проводов, надеть колеса (рис. 6.32) и приступать к созданию первой программы, которая научит робота пользоваться моторами, совершать прямолинейное движение и повороты.

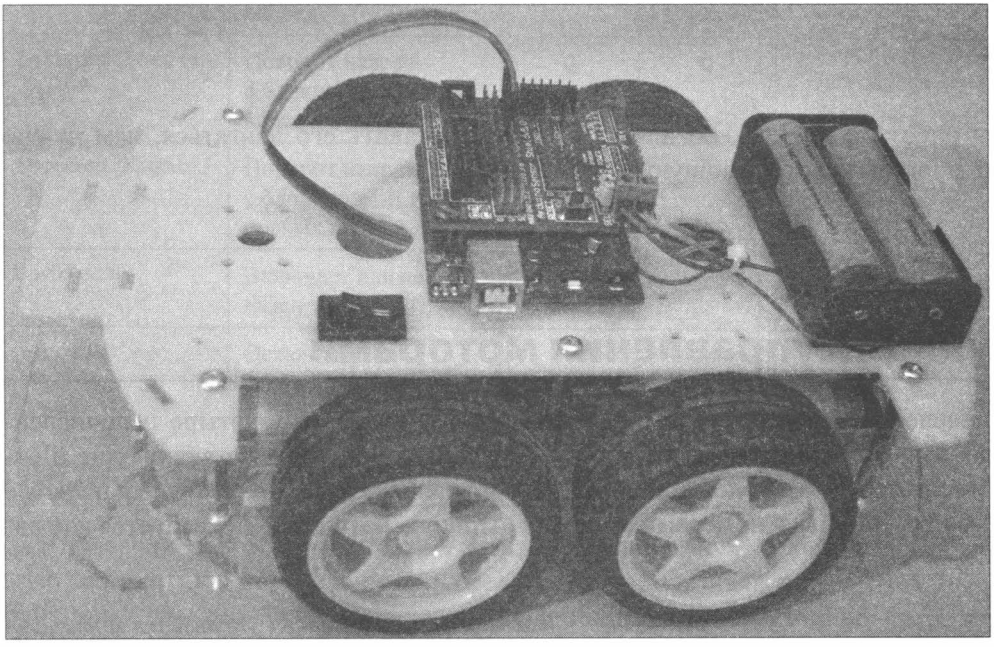


Рис. 6.32. Базовая модель робота в сборе

ГЛАВА 7
СХЕМА УПРАВЛЕНИЯ ДВИЖЕНИЕМ

Сборка робота закончена, но для того чтобы заставить его двигаться, нам нужно создать программу, отвечающую за его движения.

Переменные и функции управления моторами

Для управления моторами надо включить в нашу программу четыре целочисленные глобальные переменные (табл. 7.1) — они будут отвечать за включение и отключение моторов, а также за направление вращения колес. Значения, записанные в них, будут являться номерами выводов Arduino, которыми потребуется управлять.

Таблица 7.1. *Переменные управления моторами*

Переменная	Назначение: номер вывода Arduino — контакт драйвера
`motor_L1`	2 — IN4
`motor_L2`	3 — IN3
`motor_R1`	4 — IN1
`motor_R2`	5 — IN2

Создадим также базовую функцию `setup_motor_system` — для присвоения глобальным переменным значений с номерами портов Arduino и перевода используемых портов в режим вывода данных.

Функции движений

Для управления роботом нам также потребуются функции, управляющие его движением. Дадим им названия и закрепим за ними определенные действия (табл. 7.2).

Таблица 7.2. Функции движений

Функция	Назначение	Выставляемые значения на портах			
		motor_L1	motor_L2	motor_R1	motor_R2
`forward()`	Включает движение вперед	1	0	1	0
`forward_left()`	Поворот налево с блокировкой левых колес	0	0	1	0
`forward_right()`	Поворот направо с блокировкой правых колес	1	0	0	0
`backward()`	Включает движение назад	0	1	0	1
`_stop()`	Остановка	0	0	0	0

Первая поездка

Для проверки работоспособности собранного робота нужно записать в него простую программу с определенной последовательностью движений. Созданные нами функции только включают моторы с заданными параметрами, но не позволяют управлять длительностью процесса. Поэтому нам следует позаботиться о том, чтобы после включения какой-либо функции, — например, `forward()`, прошло некоторое время до включения следующей. За это время робот, отрабатывая поданную на него команду, каким-то образом сместится. Естественно, что от времени, в течение которого выполняется движение, зависит величина самого движения: если это поворот, то увеличение времени увеличит угол поворота, а если это движение вперед, то увеличится пройденное расстояние.

Алгоритм

Рис. 7.1 демонстрирует небольшой линейный (в нем нет условных операторов) алгоритм тестовых движений робота. Согласно этому алгоритму, робот начинает двигаться, совершает повороты и проезды, а после выполнения всех команд начинает все сначала.

Если ваш робот совершает повороты и едет не так, как задумано в программе, не спешите его разбирать. Отключите в программе все команды, кроме движения вперед, и проверьте, какие колеса вращаются неправильно. Для этих колес поменяйте местами контакты на плате Arduino Sensor Shield, а если это проблематично, то измените значения переменных `motor_L1`, `motor_L2`, `motor_R1`, `motor_R2`.

Например, если колеса правой стороны вращаются в нужную сторону, а левой — в обратную, то следует поменять местами контакты на выводах 2 и 3, если же наоборот, то на выводах 5 и 4.

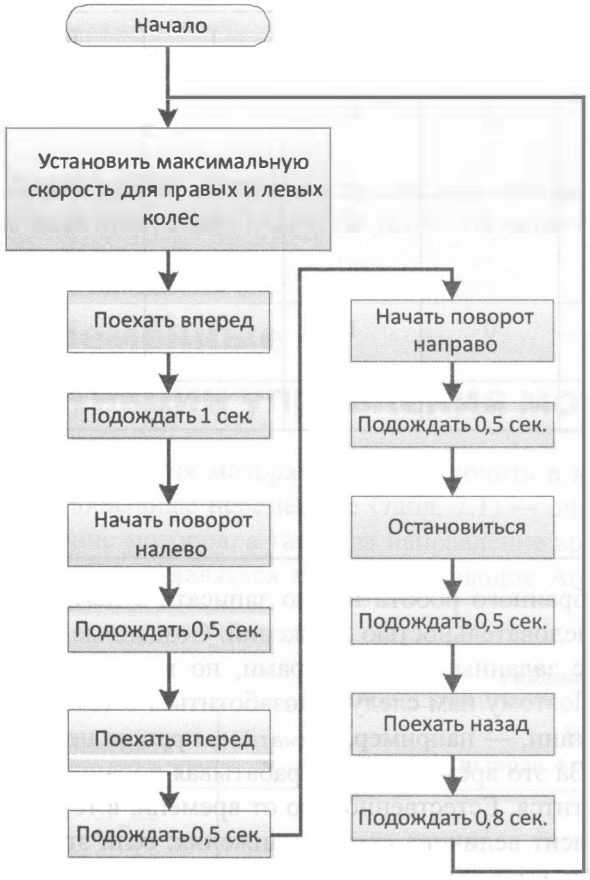


Рис. 7.1. Линейный алгоритм тестовых движений робота

Программа

Напишем в соответствии с приведенным на рис. 7.1 алгоритмом небольшую тестовую программу (листинг 7.1), которая осуществляет в цикле несколько видов движений. Ее цель — проверить, что все сделано правильно, и колеса вращаются в нужном направлении.

Листинг 7.1. Тестовая программа движений робота

```
// Объявляем переменные для хранения состояния двух моторов.
int motor_L1, motor_L2;
int motor_R1, motor_R2;
//=========================================
// Функция инициализации управления моторами.
void setup_motor_system(int L1, int  L2, int  R1, int R2)
{
  // Заносятся в переменные номера контактов (пинов) Arduino.
  // Для левых
  motor_L1=L1; motor_L2=L2;
  // и правых моторов робота.
  motor_R1=R1; motor_R2=R2;
  // Переводятся указанные порты в состояние вывода данных.
  pinMode(motor_L1, OUTPUT);
  pinMode(motor_L2, OUTPUT);
  pinMode(motor_R1, OUTPUT);
  pinMode(motor_R2, OUTPUT);
}
//=========================================
// движение вперед.
void forward()
{
    // Если двигатель будет работать не в ту сторону,
    // поменять на нем контакты местами.
    digitalWrite(motor_L1, HIGH);
    digitalWrite(motor_L2,LOW);
    digitalWrite(motor_R1, HIGH);
    digitalWrite(motor_R2,LOW);
}
//=========================================
// Поворот налево с блокировкой левых колес.
void forward_left()
{
  // блокировка вращения левых колес.
    digitalWrite(motor_L1, LOW);
    digitalWrite(motor_L2, LOW);
   // правые колеса вращаются.
    digitalWrite(motor_R1, HIGH);
    digitalWrite(motor_R2, LOW);
}
//=========================================
// Поворот направо с блокировкой правых колес.
void forward_right()
```

```
{
  // левые колеса вращаются.
    digitalWrite(motor_L1, HIGH);
    digitalWrite(motor_L2, LOW);
    // блокировка вращения правых колес.
    digitalWrite(motor_R1, LOW);
    digitalWrite(motor_R2, LOW);
}
// Включаем движение назад.
void backward()
{
    // Смена направления вращения двигателей.
    digitalWrite(motor_L2, HIGH);
    digitalWrite(motor_L1,LOW);
    digitalWrite(motor_R2, HIGH);
    digitalWrite(motor_R1,LOW);
}
//================================================
void _stop()
{
    // Блокировка всех колес.
    digitalWrite(motor_L2, LOW);
    digitalWrite(motor_L1,LOW);
    digitalWrite(motor_R2, LOW);
    digitalWrite(motor_R1,LOW);
}
//================================================
// Функция инициализации, выполняется один раз.
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2,3,4,5);
  // Двигатели остановлены.
   _stop();
}
//================================================
// Основная программа.
void loop()
{
  // Пример движения робота задан жестко в программе
  // и повторяется в цикле.
   forward(); // едет вперед 1 секунду.
   delay(1000);
   forward_left(); // поворачивает налево 0,5 секунд.
   delay(500);
```

```
    forward(); // едет вперед 0,5 секунд.
    delay(500);
    forward_right(); // поворачивает направо 0,5 секунд.
    delay(500);
    _stop();
    delay(500);
    backward(); // движется назад 0,8 секунд.
    delay(800);
}
```

Разделяем программу на два файла

Чтобы наша программа хорошо читалась, разделим ее на два файла. Все функции управления моторами (листинг 7.2, *а*) поместим в файл motor.h (он должен быть создан в папке с текущей программой) и подключим его к основной программе инструкцией `#include "motor.h"`. Файл этот можно создать в редакторе Блокнот.

Листинг 7.2, *а*. Все функции управления моторами. Вспомогательный файл motor.h

```
// Объявляем переменные для хранения состояния двух моторов.
int motor_L1, motor_L2;
int motor_R1, motor_R2;
//========================================
// Функция инициализации управления моторами.
void setup_motor_system(int L1, int  L2, int  R1, int R2)
{
  // Заносятся в переменные номера контактов (пинов) Arduino.
  motor_L1=L1; motor_L2=L2;
  // Для левых и правых моторов робота.
  motor_R1=R1; motor_R2=R2;
  // Переводятся указанные порты в состояние вывода данных.
  pinMode(motor_L1, OUTPUT);
  pinMode(motor_L2, OUTPUT);
  pinMode(motor_R1, OUTPUT);
  pinMode(motor_R2, OUTPUT);
}
//========================================
// движение вперед.
void forward()
{
    // Если двигатель будет работать не в ту сторону,
    // поменять на нем контакты местами.
    digitalWrite(motor_L1, HIGH);
    digitalWrite(motor_L2,LOW);
```

```
    digitalWrite(motor_R1, HIGH);
    digitalWrite(motor_R2,LOW);
}
//=====================================
// Поворот налево с блокировкой левых колес.
void forward_left()
{
  // блокировка вращения левых колес.
    digitalWrite(motor_L1, LOW);
    digitalWrite(motor_L2, LOW);
   // правые колеса вращаются.
    digitalWrite(motor_R1, HIGH);
    digitalWrite(motor_R2, LOW);
}
//=====================================
// Поворот направо с блокировкой правых колес.
void forward_right()
{
  // левые колеса вращаются.
    digitalWrite(motor_L1, HIGH);
    digitalWrite(motor_L2, LOW);
    // блокировка вращения правых колес.
    digitalWrite(motor_R1, LOW);
    digitalWrite(motor_R2, LOW);
}
// Включаем движение назад.
void backward()
{
    // Смена направления вращения двигателей.
    digitalWrite(motor_L2, HIGH);
    digitalWrite(motor_L1,LOW);
    digitalWrite(motor_R2, HIGH);
    digitalWrite(motor_R1,LOW);
}
//=====================================
void _stop()
{
    // Блокировка всех колес.
    digitalWrite(motor_L2, LOW);
    digitalWrite(motor_L1,LOW);
    digitalWrite(motor_R2, LOW);
    digitalWrite(motor_R1,LOW);
}
```

В этом случае основная программа будет очень короткой (листинг 7.2, *б*).

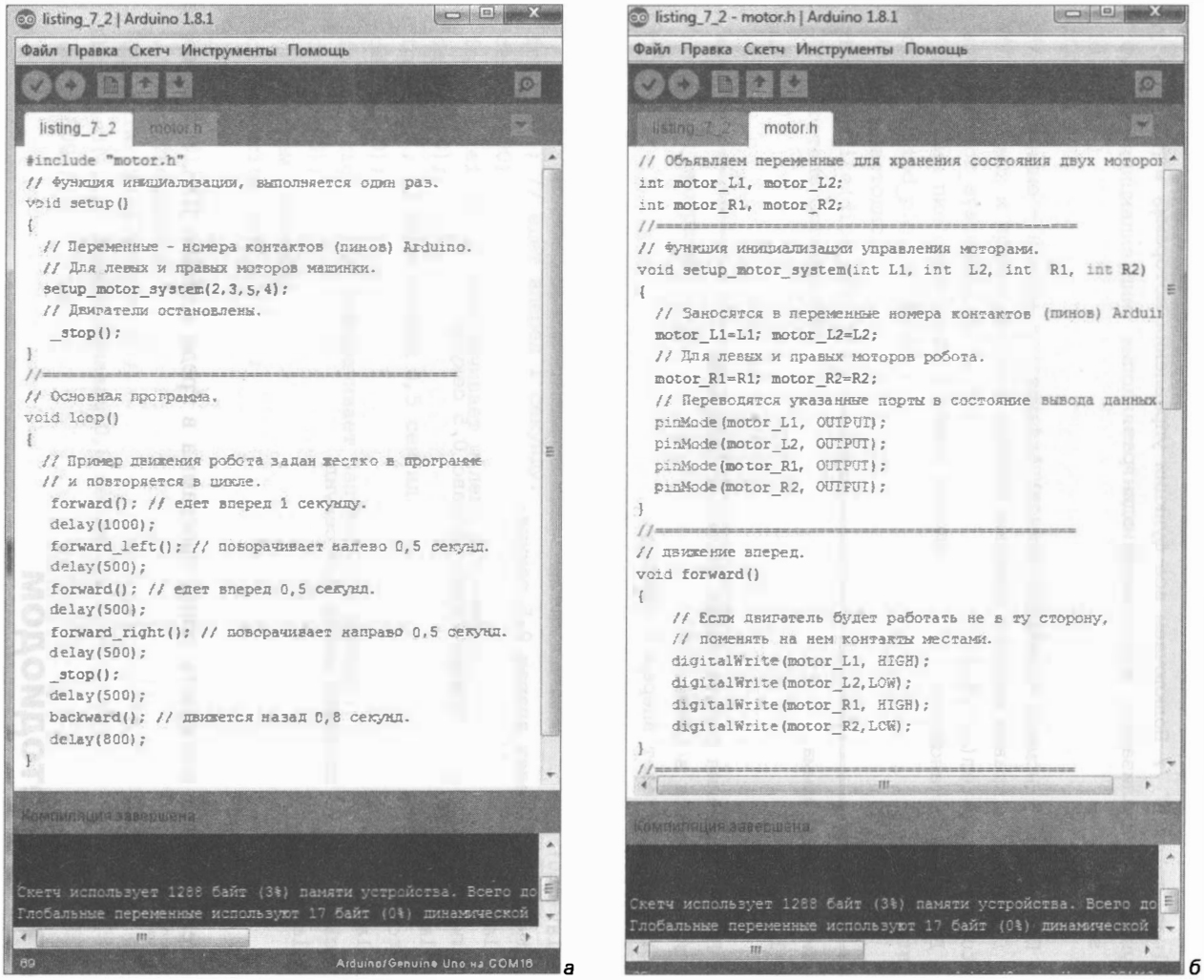


Рис. 7.2. Демонстрация деления программы на два файла: *а* — listing_7_2 и *б* — motor.h

Листинг 7.2, *б*. Тестовая программа движений робота. Основной файл программы listing_7_2.ino

```
#include "motor.h" // Подключаем все функции управления моторами
// Функция инициализации, выполняется один раз.
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2,3,5,4);
  // Двигатели остановлены.
   _stop();
}
//=========================================
// Основная программа.
void loop()
{
  // Пример движения робота задан жестко в программе
  // и повторяется в цикле.
   forward(); // едет вперед 1 секунду.
   delay(1000);
   forward_left(); // поворачивает налево 0,5 секунд.
   delay(500);
   forward(); // едет вперед 0,5 секунд.
   delay(500);
   forward_right(); // поворачивает направо 0,5 секунд.
   delay(500);
   _stop();
   delay(500);
   backward(); // движется назад 0,8 секунд.
   delay(800);
}
```

Как теперь будет выглядеть наша программа в среде Arduino IDE, показано на рис. 7.2.

Сигнал светодиодом

Модифицируем основную программу таким образом, чтобы при остановке загорался светодиод, подключенный к контроллеру Arduino в предыдущей главе (напомню, что установлен он был на управление от его 6-го контакта). Для этого нам всего лишь нужно перевести контакт 6 в режим вывода и подавать на него 1, чтобы светодиод загорелся, и 0 — чтобы погас (листинг 7.3).

Листинг 7.3. Робот движется и сигнализирует светодиодом

```
#include "motor.h"
int diod_pin = 6;
// Функция инициализации, выполняется один раз.
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 5, 4);
  //переводим пин светодиода в режим вывода.
  pinMode(diod_pin, OUTPUT);
  //гасим светодиод.
  digitalWrite(diod_pin, 0);
  // Двигатели остановлены.
  _stop();
}
//=====================================
// Основная программа.
void loop()
{
  // Пример движения робота задан жестко в программе
  // и повторяется в цикле.
  forward(); // едет вперед 1 секунду.
  delay(1000);
  forward_left(); // поворачивает налево 0,5 секунд.
  delay(500);
  forward(); // едет вперед 0,5 секунд.
  delay(500);
  forward_right(); // поворачивает направо 0,5 секунд.
  delay(500);
  //Включаем светодиод.
  digitalWrite(diod_pin, 1);
  _stop();
  delay(500);
  //Гасим светодиод.
  digitalWrite(diod_pin, 0);
  backward(); // движется назад 0,8 секунд.
  delay(800);
}
```

Выводы

Робот научился исполнять команды, позволяющие ему двигаться и поворачивать, а также сигнализировать при определенных ситуациях. Таким образом, нами создана универсальная программа, на основании которой довольно просто сделать про-

грамму движений любой сложности, чередуя этапы включения определенных двигательных функций и времени, в течение которого они работают.

Однако недостатком нашего робота является то, что он никак не реагирует на изменение внешней обстановки — не может обходить препятствия и не умеет что-либо анализировать. Причина этого в том, что анализировать ему пока нечего, поскольку у него нет информации о внешнем мире. А чтобы у него появился материал для анализа, требуется оснастить его хотя бы одним датчиком. И станет этим датчиком датчик черного цвета (черной линии), работу с которым мы рассмотрим в *главе 9*. Но прежде нам следует научиться управлять роботом дистанционно, опираясь на материал *главы 8*.

ГЛАВА 8

ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ РОБОТОМ

В этой главе мы научимся управлять роботом дистанционно, решив следующие задачи:

- определение способов дистанционного управления роботами на плате Arduino;
- обоснование выбора системы дистанционного управления;
- подбор элементной базы и разработка схемы соединений;
- подключение робота к системе дистанционного управления и тестирование его работы.

Способы дистанционного управления

Удаленное управление подвижным роботом возможно по разным каналам. В *главе 1* уже были упомянуты способы управления через инфракрасный канал, радиоканал с использованием стандарта Bluetooth и радиоканал на основе стандарта Wi-Fi. Возможно также управление роботом посредством специализированной системы управления, подобной используемой в качественных моделях летающих аппаратов (рис. 8.1).

Нам необходимо выбрать решение не дорогое, но в то же время довольно эффективное и не требующее сложного этапа проектирования и настройки. Дело в том, что при использовании дешевых систем радиоуправления можно столкнуться с проблемой, когда недорогие радиоуправляемые модели путаются в командах из-за того, что воспринимают сигналы управления, направленные как к ним, так и к другим подобным устройствам.

Выделим два приемлемых решения: управление по инфракрасному каналу (ИК/IR) и управление по Bluetooth. Управление по IR-каналу самое простое, и реализовать его можно, имея только IR-приемник (рис. 8.2), а в качестве пульта управления воспользоваться любым IP-пультом — например, от телевизора. Но IR-прием работает только в пределах прямой видимости, что делает управление роботом не очень

Рис. 8.1. Профессиональный пульт дистанционного управления спортивным роботом

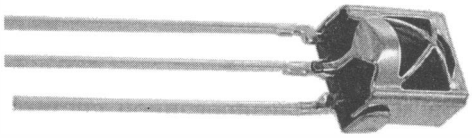


Рис. 8.2. Инфракрасный приемник

эффективным. Управление роботом по Bluetooth лишено этого недостатка, но требует создания дополнительного устройства.

Управление роботом по каналу инфракрасной связи

Рассмотрим для начала пример управления роботом при помощи IR-пульта. Готовый набор IR-компонентов для использования с Arduino-роботами представлен на рис. 8.3. Подойдет нам также и любой инфракрасный пульт от телевизора или другой бытовой техники.

Для приема инфракрасного сигнала требуется IR-приемник. Он может быть смонтирован на плате (рис. 8.4, *а*) — в этом случае у нас будет возможность визуально следить за приемом сигнала по миганию индикатора. Можно также использовать IR-приемник без платы (рис. 8.4, *б*). Следует обратить внимание, что контакты платы IR-приемника и контакты IR-приемника без платы не совпадают.

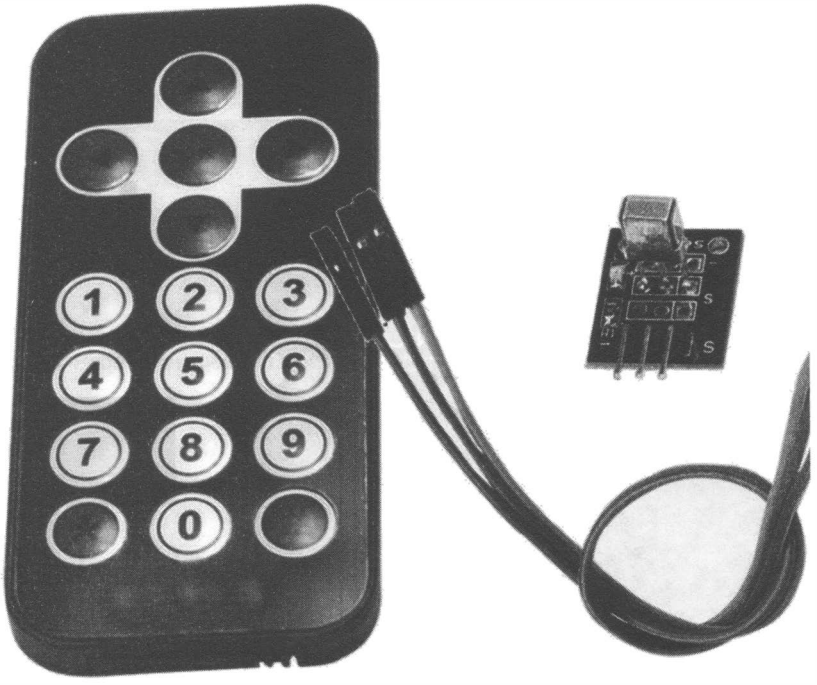


Рис. 8.3. IR-пульт (*слева*); плата IR-приемника (*справа вверху*); соединительные провода (*справа внизу*)

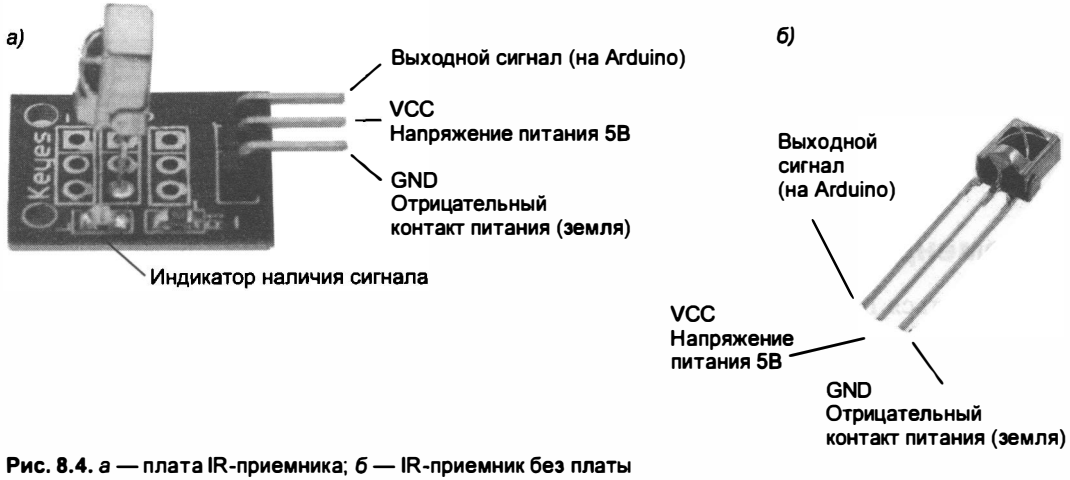


Рис. 8.4. *а* — плата IR-приемника; *б* — IR-приемник без платы

Схема подключения

Пример подключения IR-приемника, уже установленного на плате, приведен на рис. 8.5. Питание подано от платы Arduino, но можно подать его и от другого источника стабилизированного электропитания 5 В. На плате приемника (см. рис. 8.4, *а*) левый контакт обозначен символом «–» — это «земля» (GND), правый контакт обозначен символом «S» — это сигнальный контакт (Data), средний контакт на плате не обозначен — это напряжение питания (Vcc = 5 В).

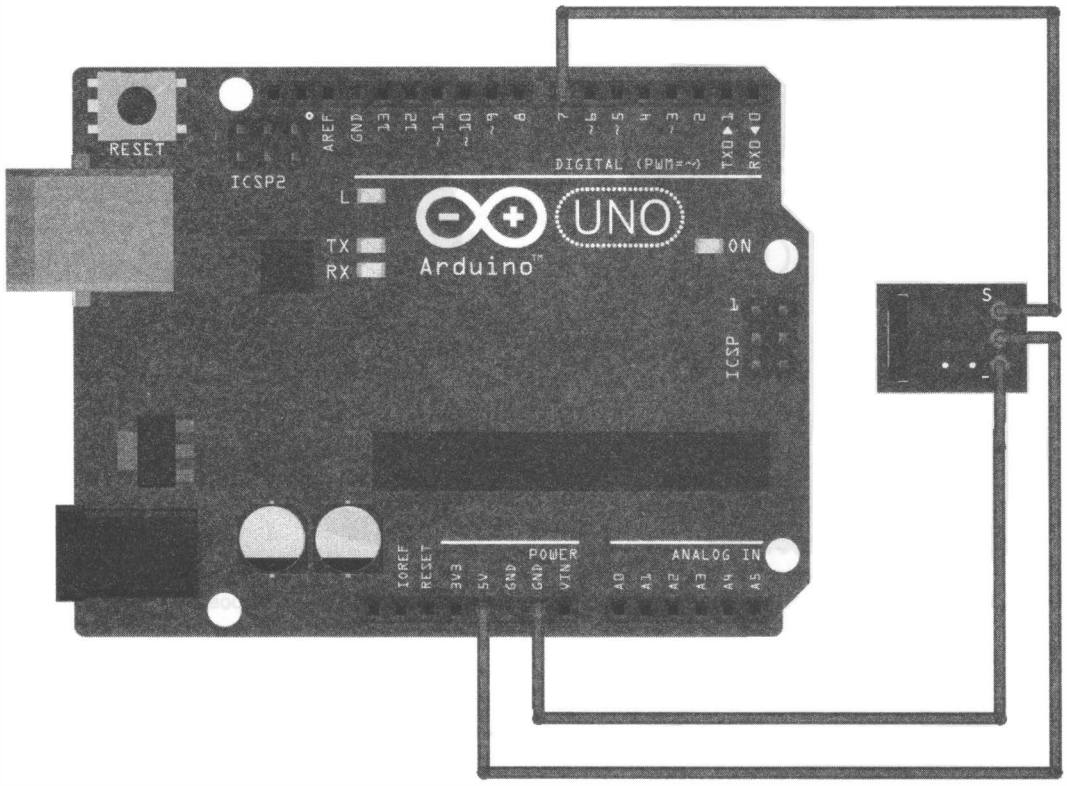


Рис. 8.5. Подключение IR-приемника к Arduino UNO

Рекомендации по установке

Располагаться IR-приемник должен таким образом, чтобы быть в прямой видимости от пульта управления. На роботе это место не должно ничем перекрываться. Общий вид робота в сборе, оснащенного IR-приемником, приведен на рис. 8.6.

В случае использования IR-приемника без платы рекомендуется для уменьшения помех и ложного срабатывания установить дополнительный фильтр питания (рис. 8.7).

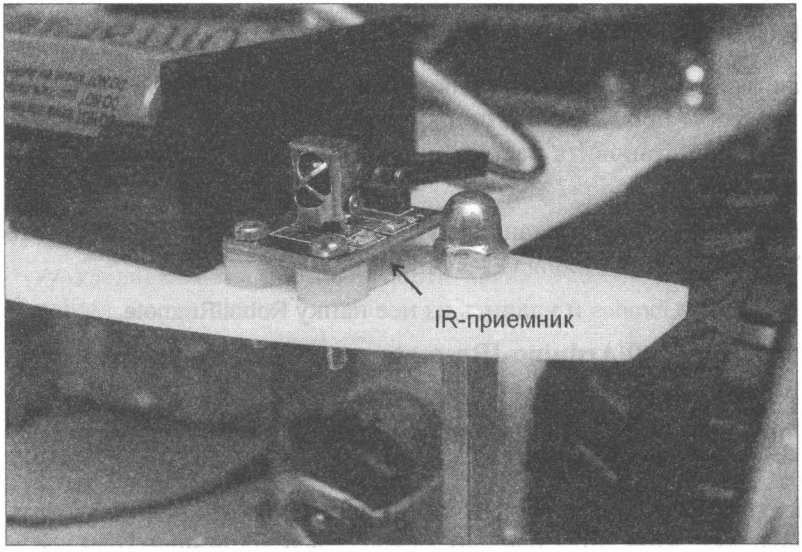


Рис. 8.6. Робот, оснащенный IR-приемником

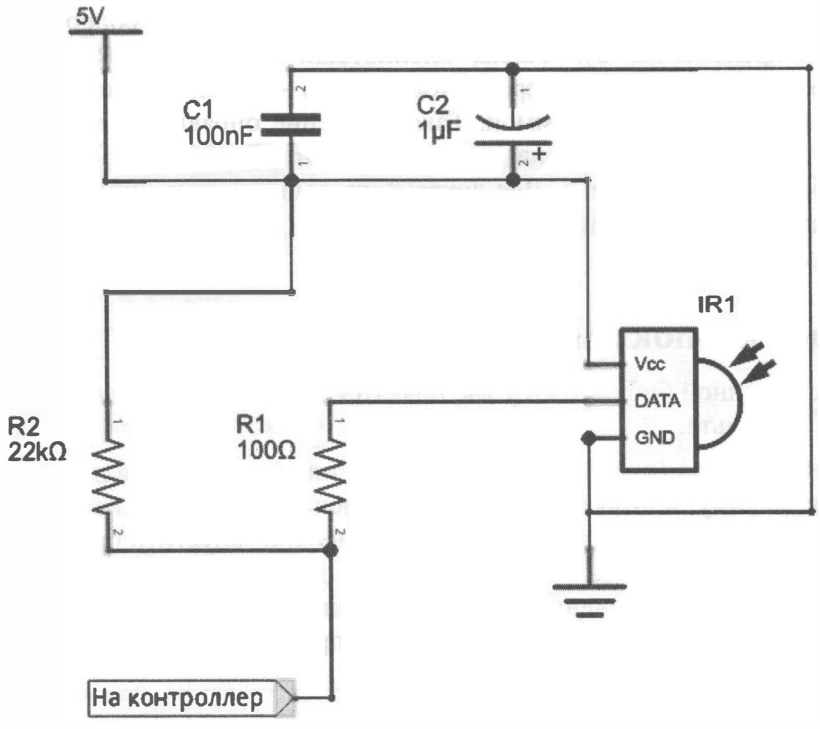


Рис. 8.7. Фильтр помех для IR-приемника

Установка расширенной библиотеки

С Arduino IDE уже поставляется библиотека для работы с IR-приемником, но она не содержит примеров. Чтобы посмотреть и попробовать типичные примеры работы с IR-приемником, нужно установить расширенную библиотеку. Для этого следует выполнить ряд действий:

1. Зайти на компьютере в папку, куда установлена программа Arduino (обычно это C:\Program Files (x86)\Arduino или C:\Program Files\Arduino).
2. В этой папке зайти в папку Libraries и удалить из нее папку RobotIRremote.
3. С сайта **https://github.com/z3t0/Arduino-IRremote** скачать ZIP-архив, содержащий обновленную библиотеку IRRemote.h.
4. Через меню Arduino IDE **Скетч | Подключить** подключить полученную библиотеку.
5. Если примеры работы с IR-приемником вам не интересны, можно использовать и встроенную библиотеку. При этом следует удалить файлы IRemoteTools.cpp и IRemoteTools.h из папки Libraries\RobotIRremote\src, иначе компиляция программы заканчивается ошибкой.

Третий вариант работы с библиотекой IR-приемника — не устанавливать ее глобально, а воспользоваться примерами программ из электронного архива, сопровождающего книгу (см. *приложение 2*). Файлы библиотеки: boarddefs.h, irRecv.cpp, IRremote.cpp, IRremote.h, IRremoteInt.h, irSend.cpp, ir_RC5_RC6.cpp — должны находиться прямо в каталоге с работающей программой. Такое решение оправдано, если вы собираетесь изменять свою программу на чужом компьютере, для которого эта библиотека может быть не установлена. Для локального подключения библиотеки следует использовать инструкцию `#include "IRremote.h"`, а для глобального подключения — `#include <IRremote.h>`.

Получение кодов кнопок для используемого пульта

После установки расширенной библиотеки у вас появятся примеры для управления роботом с помощью IR-пульта.

Выберите пример **IRrecvDemo** (рис. 8.8) — этот пример также приведен в листинге 8.1. Проверьте, к какому контакту у вас подключена сигнальная нога IR-приемника (в примере на рис. 8.5 это контакт 7). Теперь нужно запустить программу, открыть порт для просмотра кодов кнопок пульта и дополнить полученными кодами табл. 8.1, содержащую перечень функций движения робота и их назначение. Для пульта, изображенного на рис. 8.3, коды кнопок приведены в табл. 8.2.

Листинг 8.1. Проверка кодов кнопок пульта

```
// Подключение библиотеки IRremote.h.
//#include <IRremote.h> // Подключаем глобально библиотеку из Arduino IDE.
#include "IRremote.h" // Подключаем локально библиотеку из текущего каталога
```

```
// Порт Arduino для приемника.
int RECV_PIN = 7;
// Создаем объект IR-приемник.
IRrecv irrecv(RECV_PIN);
// Создаем структуру результата приема данных IR-канала.
decode_results results;
void setup()
{
  // Устанавливаем скорость порта связи с ПК.
  Serial.begin(9600);
  // Запуск IR-приемника.
  irrecv.enableIRIn();
}
void loop() {
  // Если пришли данные.
  if (irrecv.decode(&results)) {
    // Послать полученные данные на ПК в 16-м представлении.
    Serial.println(results.value, HEX);
    // Готов к приему.
    irrecv.resume();
  }
  delay(100);
}
```

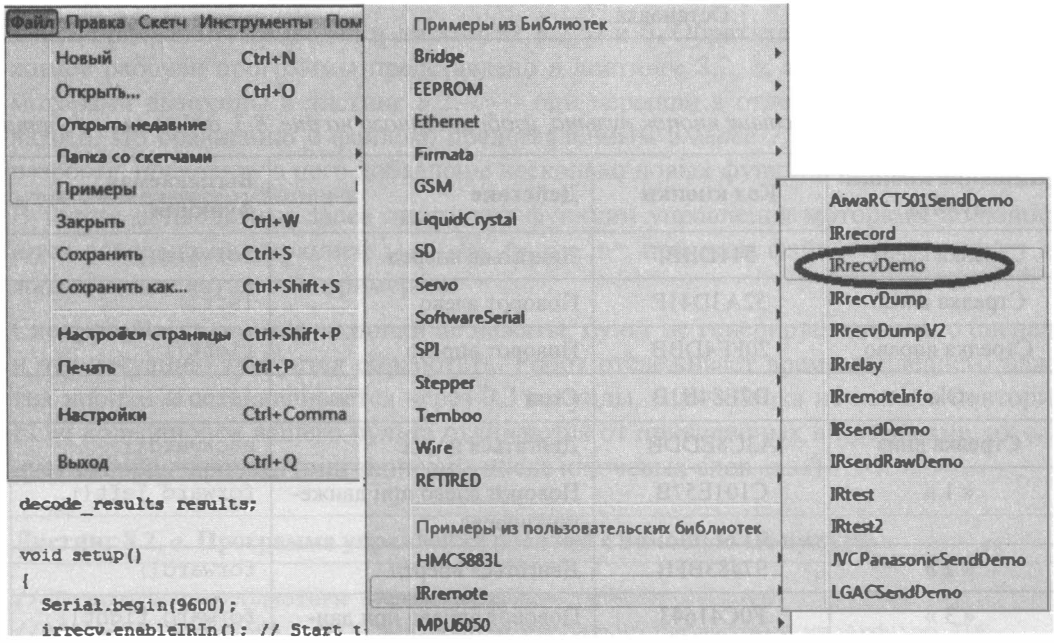


Рис. 8.8. Выбор примера программы для тестирования IR-приемника

Таблица 8.1. Функции движения робота и их назначение

№ п.п.	Функция	Назначение
1.	`forward()`	Движение вперед
2.	`forward_left()`	Поворот налево с блокировкой левых колес. Поворот применяется при движении вперед, при переключении от движения вперед к этому виду поворота не требуется остановка
3.	`forward_right()`	Поворот направо с блокировкой правых колес. Поворот применяется при движении робота вперед, при переключении от движения вперед к этому виду поворота не требуется остановка
4.	`left()`	Поворот налево на месте. Правые колеса вращаются вперед, левые назад
5.	`right()`	Поворот направо на месте. Правые колеса вращаются назад, левые вперед
6.	`backward()`	Движение назад
7.	`backward_left()`	Поворот налево с блокировкой левых колес. Поворот применяется при движении задним ходом, при переключении от движения назад к этому виду поворота не требуется остановка
8.	`backward_right()`	Поворот направо с блокировкой правых колес. Поворот применяется при движении задним ходом, при переключении от движения назад к этому виду поворота не требуется остановка
9.	`_stop()`	Остановка

Таблица 8.2. Соответствие кнопок пульта, изображенного на рис. 8.3, действиям робота

Название кнопки пульта	Код кнопки	Действие	Вызываемая функция
Стрелка вверх	511DBB	Двигаться вперед	`forward()`
Стрелка влево	52A3D41F	Поворот влево	`left()`
Стрелка вправо	20FE4DBB	Поворот вправо	`right()`
«Ok»	D7E84B1B	Стоп	`_stop()`
Стрелка вниз	A3C8EDDB	Двигаться назад	`backward()`
« 1 »	C101E57B	Поворот влево при движении вперед	`forward_left()`
« 2 »	97483BFB	Двигаться вперед	`forward()`
« 3 »	F0C41643	Поворот вправо при движении вперед	`forward_right()`
« 4 »	9716BE3F	Поворот влево	`left()`

Таблица 8.2 (окончание)

Название кнопки пульта	Код кнопки	Действие	Вызываемая функция
« 5 »	3D9AE3F7	Стоп	`_stop()`
« 6 »	6182021B	Поворот вправо	`right()`
« 7 »	8C22657B	Поворот влево при движении назад	`backward_left()`
« 8 »	488F3CBB	Двигаться назад	`backward()`
« 9 »	449E79F	Поворот вправо при движении вперед	`backward_right()`
« 0 »	1BC0157B	Стоп	`_stop()`
« * »	32C6FDF7	Стоп	`_stop()`
« # »	3EC3FC1B	Стоп	`_stop()`

Программа

Программа для робота предусматривает, что данные принимаются с IR-датчика, коды, генерируемые пультом при нажатии той или иной кнопки, взяты из табл. 8.2, принятые коды анализируются на совпадение, и при совпадении запускается та или иная функция (см. табл. 8.1).

Текст программы приведен в листингах 8.2, *а* и *б*. Обратите внимание, что содержимое рабочей программы представлено в листинге 8.2, *а*, а функции управления моторами вынесены в листинг 8.2, *б* — они перешли в отдельный файл с именем motor.h. По сравнению с файлом, представленным в *главе 7*, файл motor.h немного разросся, поскольку в него добавлены несколько новых функций (см. табл. 8.1).

В тексте приводимых далее листингов функции управления моторами подключаются локально инструкцией `#include "motor.h"`, при этом файл motor.h должен лежать в одном каталоге с примером.

Следует учесть, что если кнопки не нажаты, пульт не генерирует никакого сигнала, и эту ситуацию требуется обработать. Робот отслеживает время последнего нажатия кнопки и останавливается через 0,3 секунды, если кнопка не нажата повторно. Если коды кнопок вашего пульта отличаются от приведенных в программе, их следует скорректировать (они записаны после ключевых слов `case`).

Листинг 8.2, *а*. Программа управления роботом с помощью IR-пульта

```
// Подключение библиотеки IRremote.h.
//#include <IRremote.h> // Подключаем глобально библиотеку из Arduino IDE.
#include "IRremote.h" // Подключаем локально библиотеку из текущего каталога
#include "motor.h" // Функции управления моторами.
```

```
// Порт для IR-приемника.
int RECV_PIN = 7;
// Создание IR-приемника.
IRrecv irrecv(RECV_PIN);
// Переменная для хранения кодов кнопок, получаемых от IR-приемника.
decode_results results;
// Хранит время последнего нажатия кнопки.
unsigned long _time;
//////////////
// Опишем коды кнопок макроподстановками:
#define FORWARD 0x511DBB
#define LEFT  0x52A3D41F
#define RIGHT 0x20FE4DBB
#define STOP  0xD7E84B1B
#define BACKWARD  0xA3C8EDDB
#define FORWARDLEFT 0xC101E57B
#define FORWARD2  0x97483BFB
#define FORWARDRIGHT  0xF0C41643
#define LEFT2 0x9716BE3F
#define STOP2 0x3D9AE3F7
#define RIGHT2  0x6182021B
#define BACKWARDLEFT 0x8C22657B
#define BACKWARD2 0x488F3CBB
#define BACKWARDRIGHT 0x449E79F
#define STOP3 0x1BC0157B
#define STOP4 0x32C6FDF7
#define STOP5 0x3EC3FC1B
// Инициализация.
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); // Двигатели остановлены.
  // Запуск IR-приемника.
  irrecv.enableIRIn();
  _time = millis();
 }
// Основная программа.
void loop()
{
  if (irrecv.decode(&results))
  {
    _time = millis();
    switch (results.value) {
      // Вперед
      case FORWARD:
```

```
    forward();
    break;
  // Назад
  case BACKWARD:
    backward();
    break;
  // Влево
  case LEFT:
    left();
    break;
  // Вправо
  case RIGHT:
    right();
    break;
    // Вперед
  case FORWARD2:
    forward();
    break;
  // Назад
  case BACKWARD2:
    backward();
    break;
  // Влево
  case LEFT2:
    left();
    break;
  // Вправо
  case RIGHT2:
    right();
    break;
  // Прямо и влево
  case FORWARDLEFT:
    forward_left();
    break;
  // Прямо и вправо
  case FORWARDRIGHT:
    forward_right();
    break;
  // Назад и влево
  case BACKWARDLEFT:
    backward_left();
    break;
  // Назад и вправо
  case BACKWARDRIGHT:
    backward_right();
    break;
```

```
      // Стоп
      case STOP:
        _stop();
       break;
      case STOP2:
        _stop();
       break;
      case STOP3:
        _stop();
       break;
      case STOP4:
        _stop();
       break;
      case STOP5:
        _stop();
       break;
    }
    irrecv.resume();
  }
  // Если никакая клавиша не нажата более 0,3 сек., то остановка.
  if((millis()-_time)>300) {_stop();}
}//==========================================================================
```

Листинг 8.2, *б*. Содержимое модифицированного файла motor.h

```
// Объявляем переменные для хранения состояния двух моторов.
int motor_L1, motor_L2;
int motor_R1, motor_R2;
//=========================================
// Функция инициализации управления моторами.
void setup_motor_system(int L1, int  L2, int  R1, int R2)
{
  // Заносятся в переменные номера контактов (пинов) Arduino.
  motor_L1 = L1; motor_L2 = L2;
  // Для левых и правых моторов робота.
  motor_R1 = R1; motor_R2 = R2;
  // Переводятся указанные порты в состояние вывода данных.
  pinMode(motor_L1, OUTPUT);
  pinMode(motor_L2, OUTPUT);
  pinMode(motor_R1, OUTPUT);
  pinMode(motor_R2, OUTPUT);
}
//=========================================
// движение вперед.
void forward()
{
  // Если двигатель будет работать не в ту сторону,
  // поменять на нем контакты местами.
```

```
  digitalWrite(motor_L1, HIGH);
  digitalWrite(motor_L2, LOW);
  digitalWrite(motor_R1, HIGH);
  digitalWrite(motor_R2, LOW);
}
//=============================================
// Поворот налево с блокировкой левых колес.
void forward_left()
{
  // блокировка вращения левых колес.
  digitalWrite(motor_L1, LOW);
  digitalWrite(motor_L2, LOW);
  // правые колеса вращаются.
  digitalWrite(motor_R1, HIGH);
  digitalWrite(motor_R2, LOW);
}
//=============================================
// Поворот направо с блокировкой правых колес.
void forward_right()
{
  // левые колеса вращаются.
  digitalWrite(motor_L1, HIGH);
  digitalWrite(motor_L2, LOW);
  // блокировка вращения правых колес.
  digitalWrite(motor_R1, LOW);
  digitalWrite(motor_R2, LOW);
}
// Поворот налево на месте.
void left()
{
  // левые колеса вращаются назад.
  digitalWrite(motor_L1, LOW);
  digitalWrite(motor_L2, HIGH);
  // правые колеса вращаются вперед.
  digitalWrite(motor_R1, HIGH);
  digitalWrite(motor_R2, LOW);
}
//=============================================
// Поворот направо на месте.
void right()
{
  // левые колеса вращаются вперед.
  digitalWrite(motor_L1, HIGH);
  digitalWrite(motor_L2, LOW);
  // правые колеса вращаются назад.
  digitalWrite(motor_R1, LOW);
  digitalWrite(motor_R2, HIGH);
}
```

```
// Включаем движение назад.
void backward()
{
  // Смена направления вращения двигателей.
  digitalWrite(motor_L2, HIGH);
  digitalWrite(motor_L1, LOW);
  digitalWrite(motor_R2, HIGH);
  digitalWrite(motor_R1, LOW);
}
//=======================================
// Включаем движение назад.
void backward_left()
{
  // Смена направления вращения двигателей.
  digitalWrite(motor_L2, LOW);
  digitalWrite(motor_L1, HIGH);
  digitalWrite(motor_R2, HIGH);
  digitalWrite(motor_R1, LOW);
}
//=======================================
// Включаем движение назад.
void backward_right()
{
  // Смена направления вращения двигателей.
  digitalWrite(motor_L2, HIGH);
  digitalWrite(motor_L1, LOW);
  digitalWrite(motor_R2, LOW);
  digitalWrite(motor_R1, HIGH);
}
//=======================================
void _stop()
{
  // Блокировка всех колес.
  digitalWrite(motor_L2, LOW);
  digitalWrite(motor_L1, LOW);
  digitalWrite(motor_R2, LOW);
  digitalWrite(motor_R1, LOW);
}
//=======================================
```

Управление роботом по каналу Bluetooth

Для связи с роботом по Bluetooth потребуется установить на него контроллер Bluetooth — тогда управлять роботом можно будет со смартфона на операционной системе Android. В магазине приложений для Android-устройств Google Play Маркет есть достаточное количество бесплатных приложений, реализующих функции управления колесными роботами (рис. 8.9).

Рис. 8.9. Программы управления роботами на Google Play Маркет (для смартфонов на Android)

Подбор элементной базы

В качестве Bluetooth-контроллера робота можно применить недорогие модули HC-05, HC-06 (рис. 8.10) или SPP-C (рис. 8.11). Питание 5 В подается на контакты VCC и GND. Для работы на передачу и прием используются контакты RXD и TXD, на плате Arduino робота потребуется задействовать для работы с этим Bluetooth-контроллером два свободных порта. Если хотите установить контроллеру имя, отличное от имени по умолчанию, контроллер следует перепрошить. При этом для модуля HC-05 (имя по умолчанию `HC-05`) может потребоваться еще один порт. После смены имени порт можно будет освободить.

Вариант Bluetooth-модуля SPP-C (см. рис. 8.11) удобен тем, что не требует применения пайки для изменения имени и других настроек. После подключения по умолчанию ему присваивается имя `BT04-A`. Подключается он аналогично HC-05, но есть небольшое отличие в синтаксисе написания программ: при использовании AT-команд между командой и параметром не ставится знак =. Например, команда присвоения имени модулю для HC-05 выглядит так:

```
AT+NAME=ROBOPES
```

а для SPP-C и HC-06 так:

```
AT+NAMEROBOPES
```

По этой команде модулю будет присвоено имя `ROBOPES`.

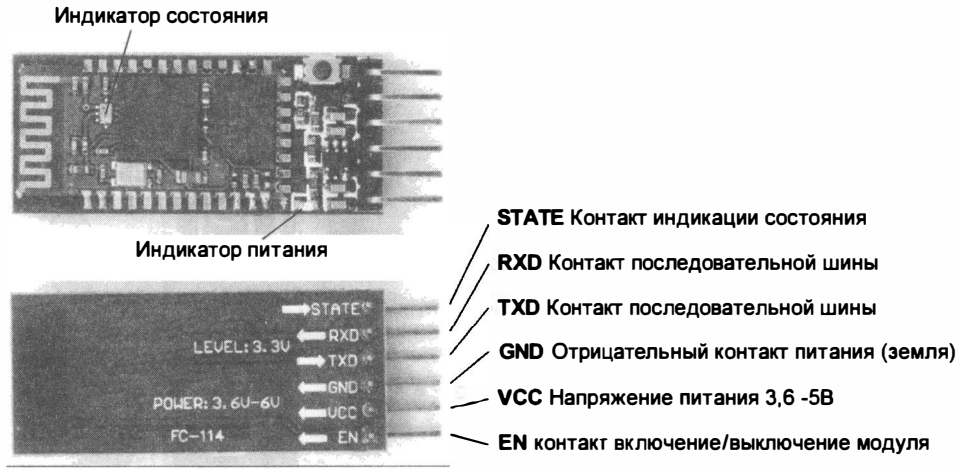


Рис. 8.10. Модуль HC-05/HC-06 приемопередачи по Bluetooth

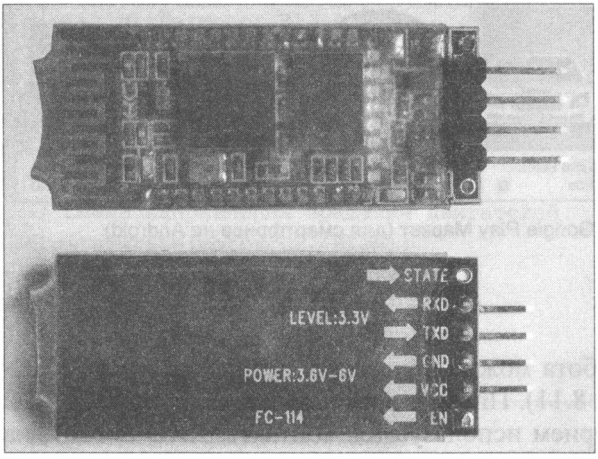


Рис. 8.11. Модуль SPP-C приемопередачи по Bluetooth. Назначение контактов то же, что и на рис. 8.10

За этим небольшим исключением все написанное далее одинаково для рассмотренных модулей. Еще стоит отметить, что HC-06 визуально не отличается от HC-05, но в HC-06 отсутствует ряд режимов работы, которые в этой книге не рассматриваются.

Подключение к Arduino

Обычно Bluetooth-контроллер подключается на стандартные входы последовательного порта D0 и D1 — на плате Arduino они обозначены как RX и TX. Но эти же входы используются при программировании робота, поэтому каждый раз при проведении процедуры программирования вам придется отсоединять Bluetooth-контроллер, иначе он будет мешать обмену информацией между компьютером и Arduino.

Программирование робота по Bluetooth

Можно программировать робота и по Bluetooth, подключив Bluetooth-контроллер к контактам D0 и D1 контроллера Arduino, но эту тему мы оставим для самостоятельного изучения.

Поэтому правильнее использовать специальную библиотеку эмуляции последовательного порта и подключить контроллер на любые свободные входы Arduino (D2–D13 или A0–A3). Выберем D10 и D11. Подключение должно быть перекрестным: RXD контроллера подключается к TX Arduino, а TXD контроллера — к RX Arduino. Контакты RXD и RX читают данные, а TXD и TX их передают, — соответственно, в паре соединенных контактов один должен быть передающим, а другой — читающим. Это нужно будет учесть при создании последовательного порта в программе робота. Порядок соединения приведен на рис. 8.12. Подключение

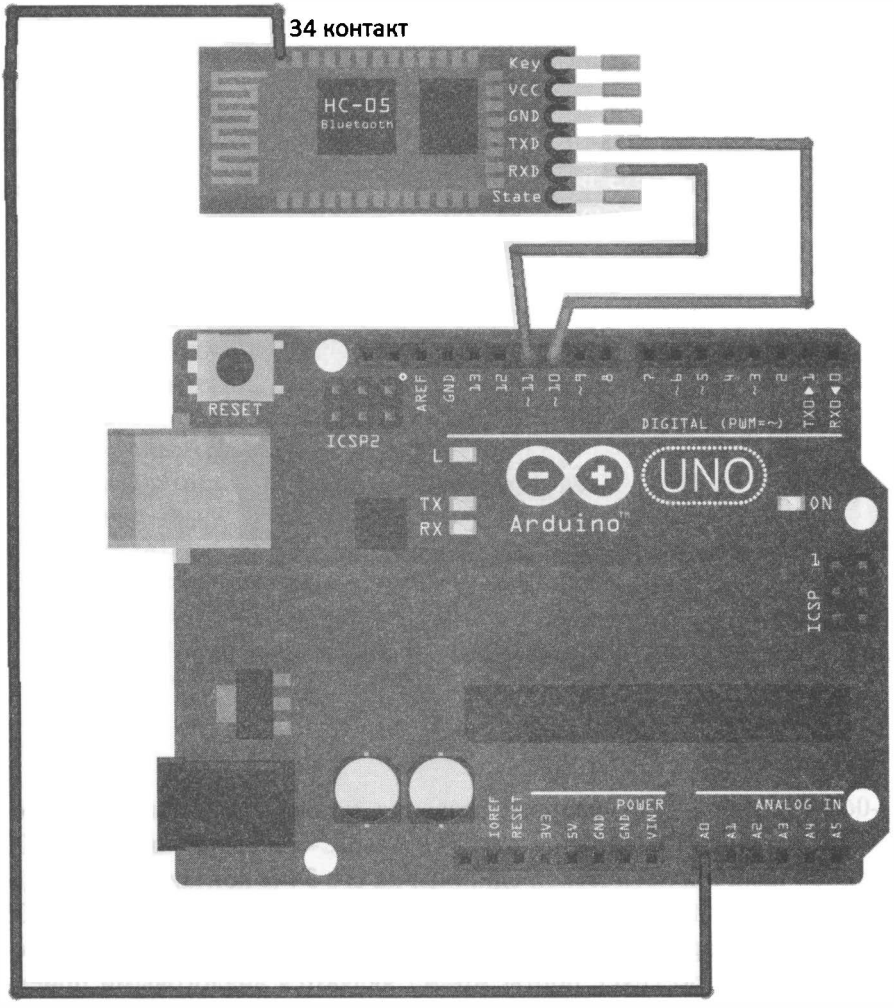


Рис. 8.12. Подключение контроллера Bluetooth HC-05 к Arduino UNO с контактом для модификации имени

34 ноги контроллера Bluetooth к порту A0 актуально, только если требуется сменить имя модуля HC-5, для HC-06 или SPP-C этого не требуется.

Так как приемопередатчик Bluetooth потребляет относительно много энергии, электропитание для него следует получать от отдельного источника — в нашем случае от стабилизатора платы драйвера двигателей через Arduino Sensor Shield v5.0 (рис. 8.13). Теперь, если включить робота и выполнить на смартфоне поиск Bluetooth-устройств, в нем появится новое Bluetooth-устройство — оно будет называться так, как всю партию прошили на фабрике, — например, **HC-05**.

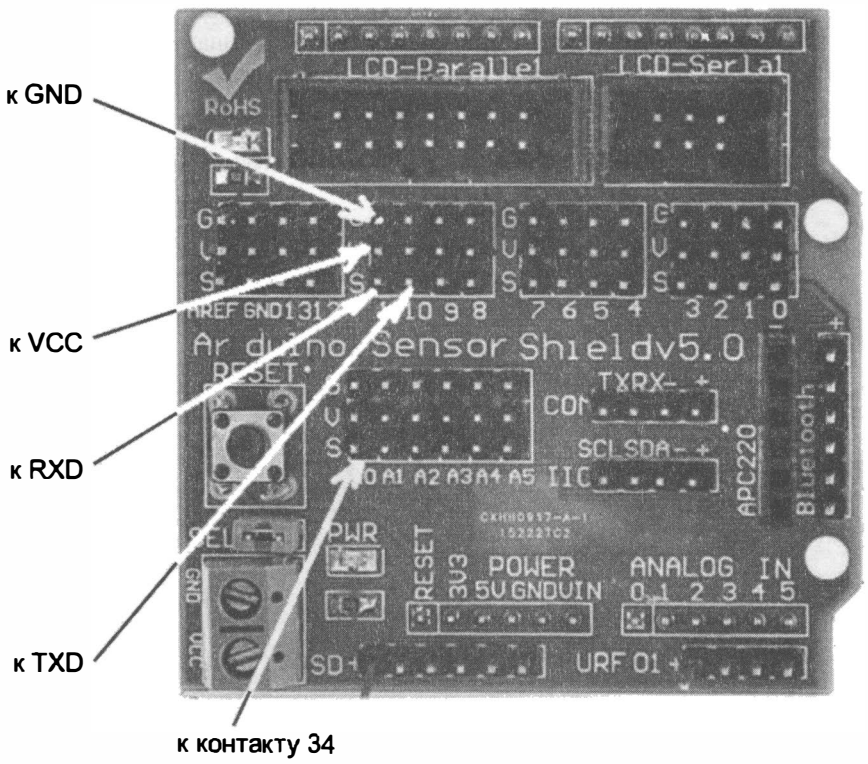


Рис. 8.13. Подключение контроллера Bluetooth HC-05 через Arduino Sensor Shield V5.0

Смена имени робота

HC-05 или HC-06 — не очень оригинальные названия, но они прошиты внутри модулей, и их можно изменить. В отличие от SPP-C и HC-6, перепрошивка HC-05 возможна только при наличии логической единицы на его 34-м контакте. Для подачи такой логической единицы потребуется присоединить к контакту 34 HC-05 (см. рис. 8.12) свободный порт контролера Arduino — пусть это будет порт A0. Согласно инструкции на Arduino UNO, порты A0–A7 являются аналоговыми приемниками, но A0–A5 могут также работать и в двоичном режиме как для чтения, так и для вывода информации. Контакт 34 расположен довольно близко к 33-му контак-

ту — паяйте осторожно! После изменения имени робота контакт 34 можно будет освободить.

Программа переименования робота для описанных настроек (используемых портов) приведена в листинге 8.3. Новое имя присваивается роботу в строке `BTSerial.println("AT+NAME=MaxiBot")`, здесь это имя `MaxiBot`. Придумайте и задайте имя своему роботу латинскими буквами без пробелов.

Проверьте результат прошивки, проведя поиск новых устройств Bluetooth на смартфоне.

Листинг 8.3. Переименование робота и проверка работы Bluetooth

```
// Подключаем библиотеку для работы с Serial через дискретные порты
#include <SoftwareSerial.h>
// Создаем последовательный порт на пинах 10 и 11
SoftwareSerial BTSerial(10, 11); // RX, TX
// Переменная для приема данных по Bluetooth
char bt_input;
// Настройка
void setup()
{
  // Устанавливаем скорость передачи данных по Bluetooth
  BTSerial.begin(9600);
  // Переключаем A0 в двоичный режим работы, на передачу
  pinMode(14, OUTPUT);//Только для HC-05
  // Переводим HC-05 в режим AT-команд
  digitalWrite(14, 1); //Только для HC-05
  // Присваиваем HC-05 новое имя - MaxiBot
  BTSerial.println("AT+NAME=MaxiBot");//Для HC-05
  //Для модуля из набора HC-06 и SPP-C
  // BTSerial.println("AT+NAMEMaxiBot");
  // Устанавливаем скорость передачи данных по кабелю
  Serial.begin(9600);
  // переводим HC-05 в нормальный режим работы
  digitalWrite(14, 0); //Только для HC-05
}
void loop()
{
  // Если пришли данные по Bluetooth
  if (BTSerial.available())
  {
    // Записываем, что пришло по Bluetooth, в переменную bt_input
    bt_input = (char)BTSerial.read();
    Serial.println(bt_input);
  }
}
```

Настройка смартфона

Для управления роботом по каналу Bluetooth вам потребуется смартфон на операционной системе Android и программа удаленного управления. Удачным выбором может стать программа Arduino Bluetooth RC Car. Она хорошо документирована и неплохо зарекомендовала себя в тестах. Установите ее с Google Play Маркет на смартфон. Интерфейс программы приведен на рис. 8.14.

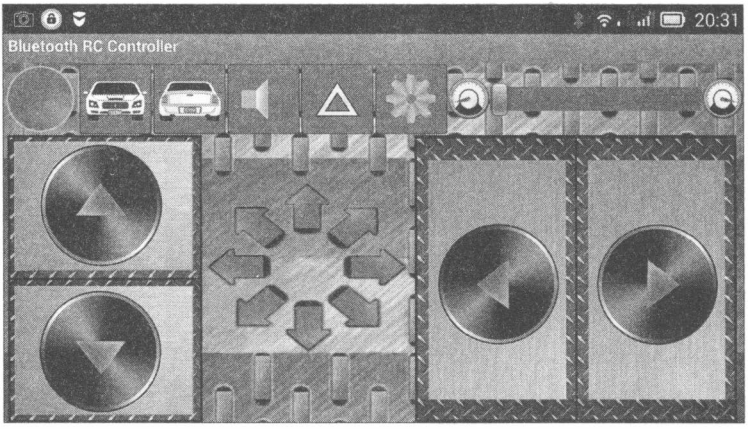


Рис. 8.14. Главное окно программы Arduino Bluetooth RC Car

Чтобы подключиться к роботу, следует нажать кнопку в виде шестеренки, активировать Bluetooth, после чего выбрать пункт **Connect**. Если робот уже прописан в устройствах смартфона, нужно будет выбрать его из меню (рис. 8.15), в противном случае произвести новое сканирование и подключить робота. Пароль по умолчанию у HC-05: `1234`, это может потребоваться!

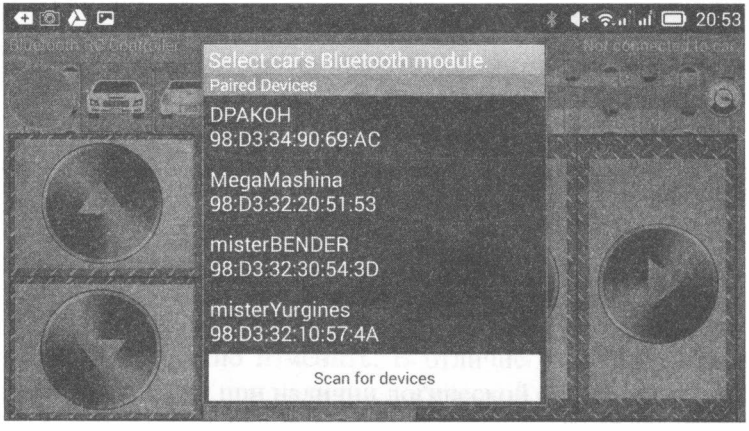


Рис. 8.15. Выбор Bluetooth-устройства

Когда робот соединится со смартфоном, интерфейс программы изменится: серая лампочка слева вверху станет зеленой, а в верхнем правом углу появится имя робота, к которому подсоединен смартфон (рис. 8.16).

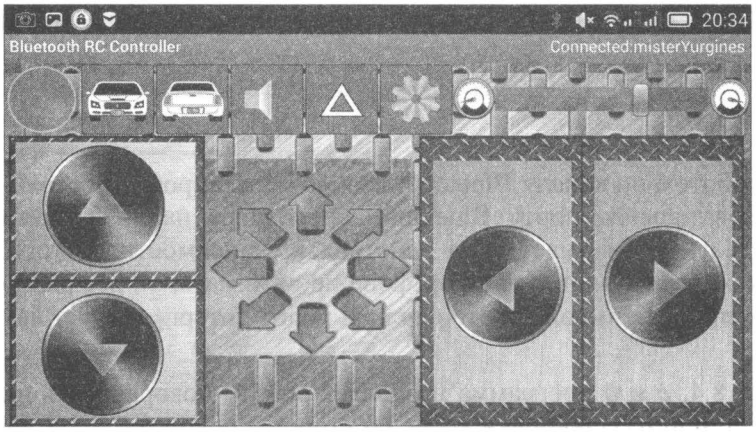


Рис. 8.16. Соединение установлено

Устранение радиопомех

Когда речь идет о работе по радиоканалу, возникают проблемы, связанные с электромагнитной совместимостью. В *главах 5* и *6* уже рассматривался вопрос защиты роботов от помех, создаваемых собственными электромоторами. Точно так же установка Bluetooth-модуля рядом с сервомотором головы, создающим радиопомехи, может приводить к частым обрывам связи. Лучше всего (не обязательно), если Bluetooth-модуль будет расположен на небольшой антенне и подальше от двигателей. При этом, если на все моторы будут установлены рекомендованные в *главах 5* и *6* защитные конденсаторы, то радиус связи вашего робота значительно увеличится, и обрывы связи сократятся.

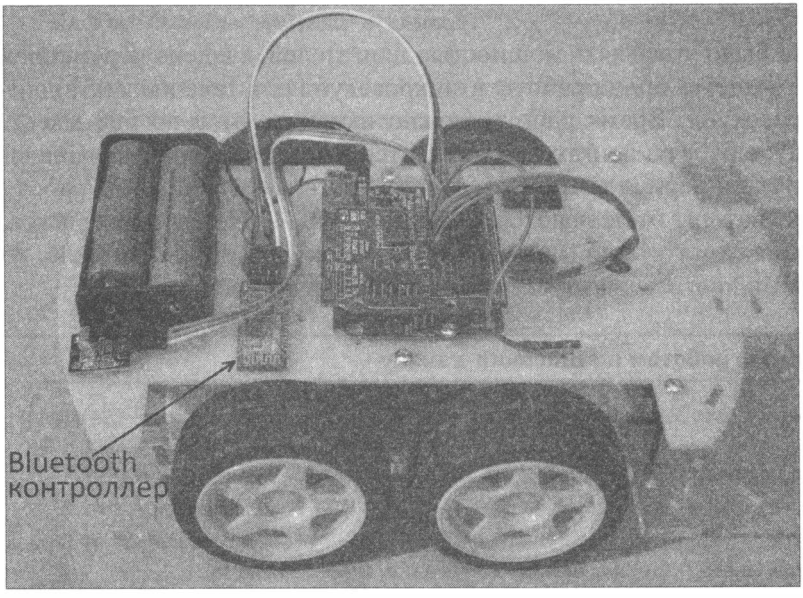


Рис. 8.17. Робот, оснащенный контроллером Bluetooth HC-05/H-06

Общий вид робота в сборе, оснащенного контроллером Bluetooth, приведен на рис. 8.17.

Программа

Программа управления роботом по каналу Bluetooth весьма проста: робот постоянно проверяет на последовательном порту Bluetooth-контроллера наличие байта команды. Если команда есть — изменяет свои действия в зависимости от того, какой байт пришел. Если никакие кнопки на смартфоне не нажимать, он все равно отправляет команду, а именно — символ `S`, которую наш робот интерпретирует как остановку.

Приведенная в листингах 8.4, *а* и *б* программа позволяет роботу поворачивать на полном ходу, используя принцип блокировки колес со стороны поворота. Если же робот стоит, то он поворачивает всеми колесами, — при этом правые и левые вращаются в разные стороны. Робот может поворачивать и при езде задним ходом. Кроме того, программа Arduino Bluetooth RC Car дает возможность управлять роботом посредством изменения ориентации смартфона в пространстве.

Кроме заявленных функций, программа Arduino Bluetooth RC Car позволяет, например, включать и отключать фары, если включение фар предусмотрено конструкцией робота.

Подробнее прочитать о том, какие символы-команды посылает смартфон роботу, можно на странице помощи в программе управления на смартфоне.

Собственно программа управления роботом по Bluetooth-каналу представлена в листинге 8.4, *а*. А для более удобной работы анализ нажатых кнопок вынесен в отдельную функцию `move_case(char bt_input)`, помещенную в файл **move_case.h** (листинг 8.4, *б*), который подключается к основной программе инструкцией `#include "move_case.h"`.

Для того чтобы можно было управлять мощностью двигателей, введена переменная `_move_time`, в которой задается время работы в микросекундах в течение такта длительностью 500 микросекунд. Время работы можно изменять от 0 до 500 мксек. Сама организация тактов работы двигателей находится в конце программы. Сначала проверяется, не истек ли период времени текущего такта, выделенный на работу моторов, а если истек, моторы отключаются командой `_stop()`. Затем проверяется, не закончилось ли время самого такта (оно равно 500 мксек), а если оно истекло, то начинается новый такт, и двигатели запускаются вызовом функции `move_case`.

Листинг 8.4, *а*. Управление роботом по Bluetooth-каналу

```
// Подключаем библиотеку для создания дополнительных последовательных (Serial)
портов.
#include <SoftwareSerial.h>
#include "motor.h"
#include "move_case.h"
// Создаем последовательный порт на пинах 13-чтение и 2-передача.
SoftwareSerial BTSerial(10, 11); // RX, TX
```

```
// Переменная для приема данных по Bluetooth.
char bt_input;
// Хранит время последнего нажатия кнопки.
unsigned long _time;
//================================//
//== Функция инициализации
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
  // Устанавливаем скорость передачи данных для HC-05 (Bluetooth-модуль).
  BTSerial.begin(9600);
  // Переключаем A0 в двоичный режим работы, на передачу,
  // если вы его еще не отключили
  pinMode(14, OUTPUT);
  // Устанавливаем скорость передачи данных по кабелю.
  // Порт компьютера
  //Serial.begin(9600);
  _time = micros();
}
// Основная программа.
void loop()
{
  if (BTSerial.available())
  {
    // Читаем команду и заносим ее в переменную. char преобразует
    // код символа команды в символ.
    bt_input = (char)BTSerial.read();
    // Отправляем команду в порт, чтобы можно было
    // их проверить в "Мониторе порта".
    //Serial.println(bt_input);
    // Вызов функции выбора действия по команде
    move_case(bt_input);
    _time = micros();
  }
  if ((micros() - _time) > _move_time)
  {
    _stop();
  }
  if ((micros() - _time) >= 500)
  {
    _time = micros();
    move_case(bt_input);
  }
}
```

Листинг 8.4, *б*. Содержимое файла move_case.h

```
// Пременная изменения скорости.
unsigned long  _move_time;
// == Выбор действий
void move_case(char bt_input)
{
  switch (bt_input) {
    // Вперед
    case 'F':
      forward();
      break;
    // Назад
    case 'B':
      backward();
      break;
    // Влево
    case 'L':
      left();
      break;
    // Вправо
    case 'R':
      right();
      break;
    // Прямо и влево
    case 'G':
      forward_left();
      break;
    // Прямо и вправо
    case 'I':
      forward_right();
      break;
    // Назад и влево
    case 'H':
      backward_left();
      break;
    // Назад и вправо
    case 'J':
      backward_right();
      break;
    // Стоп
    case 'S':
      _stop();
      break;
    // Скорость 0%
    case '0':
      _move_time = 0;
      break;
```

```
// Скорость 10%
case '1':
  _move_time = 50;
  break;
// Скорость 20%
case '2':
  _move_time = 100;
  break;
// Скорость 30%
case '3':
  _move_time = 150;
  break;
// Скорость 40%
case '4':
  _move_time = 200;
  break;
// Скорость 50%
case '5':
  _move_time = 250;
  break;
// Скорость 60%
case '6':
  _move_time = 300;
  break;
// Скорость 70%
case '7':
  _move_time = 350;
  break;
  // Скорость 80%
  _move_time = 400;
  break;
// Скорость 90%
case '9':
  _move_time = 450;
  break;
// Скорость 100%
case 'q':
  _move_time = 500;
  break;
case 'X':
  //switch_rejim = 1;
  break;
case 'x':
  //switch_rejim = 0;
  break;
```

```
    case 'D':
      _stop(); //switch_rejim = 0;
      break;
  }
}
```

Основные проблемы, которые могут возникнуть при подключении контроллера Bluetooth, связаны с электромагнитной несовместимостью и невнимательностью. Если контроллер Bluetooth не изменяет свое имя с первого раза, следует сделать повторную попытку.

Принцип управления роботом по каналу Bluetooth теперь должен быть вам понятен. Здесь не рассмотрен вопрос обратной передачи информации от робота и связи двух роботов между собой, но если вы успешно собрали и испытали рассмотренного до сих пор робота, то добьетесь остального самостоятельно.

Выводы

В этой главе проанализированы способы дистанционного управления роботами на платформе Arduino. Подробно рассмотрено управление роботом с помощью IR-канала и канала Bluetooth. Подобрана элементная база, разработаны схемы подключения и программы управления как для Bluetooth, так и для IR-канала, — теперь роботом можно управлять удаленно с любого IR-пульта (помните, что разные пульты генерируют разные последовательности).

В следующей главе мы научим робота следовать по извилистой черной линии.

ГЛАВА 9

ДВИЖЕНИЕ ПО ЧЕРНОЙ ЛИНИИ

Классической задачей для мобильных роботов является движение по линии. В нашем случае фон будет белым, а линия — черного цвета, шириной 4 см. Наш робот должен двигаться по этой черной линии, не сбиваясь с курса.

Решение этой задачи предполагает:

- рассмотрение способов детектирования черной линии Arduino;
- обоснование выбора датчиков детектирования черной линии;
- подключение датчиков к роботу;
- составление алгоритма и практическую реализацию программы.

В качестве примера приведем типичную трассу для соревнований по «биатлону» для роботов (рис. 9.1), которые традиционно проводят на робофестивалях. В продолжение эстафеты робот должен начать движение от старта и, двигаясь только по черной линии, проехать, не задев фишки, установленные в позициях 5, 6, 7 и 8;

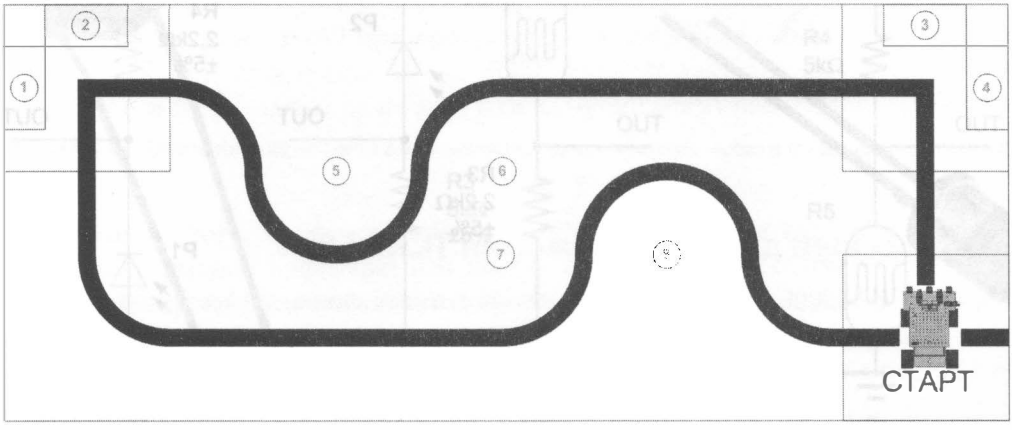


Рис. 9.1. Пример черной линии на соревнованиях роботов по «биатлону»

проезжая мимо, сбить фишки, установленные в позициях 1 и 2; забрать с собой к финишу фишки, находящиеся в позициях 3 и 4.

Такую сложную задачу мы пока ставить перед собой не будем, а ограничимся только движением по линии.

Конечно, можно попытаться запрограммировать робота таким образом, чтобы он четко знал, где ехать прямо, а где совершить поворот, но при этом малейшие непредвиденные обстоятельства: небольшая неровность, изменение шероховатости покрытия, удар о препятствие — собьют робота с курса. Намного эффективнее робот действует, если будет знать, где поверхность черная, а где светлая.

Обнаружение черной линии

Обнаружить черную линию нам помогут электронные приборы, которые реагируют на изменение освещенности. Рассмотрим четыре таких прибора, наиболее применимых для решения поставленной задачи.

Фотодиод

Фотодиод (рис. 9.2, *а*) в темноте ведет себя как обычный диод — пропускает электрический ток только в одном направлении, а при освещении генерирует небольшой ток даже без источника питания. При обратном включении фотодиода он, при достижении освещенностью определенного уровня, начинает пропускать электрической ток в обратном направлении — это позволяет использовать фотодиоды в качестве датчиков наличия освещенности.

Схемы обратного включения фотодиодов приведены на рис. 9.2, *б* и *в*. Сигнал снимается с выхода OUT, при этом для схемы 9.2, *б* наличие освещенности харак-

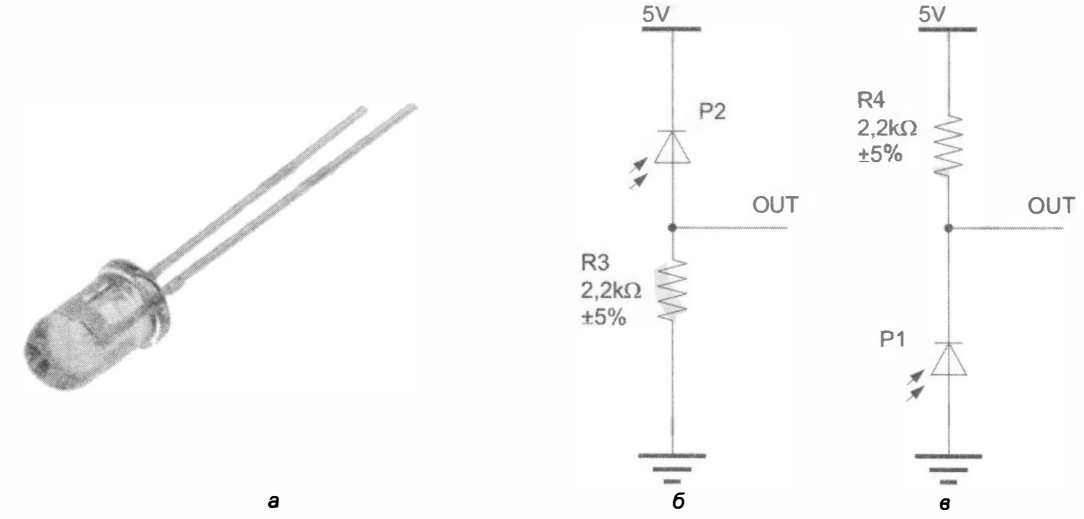


Рис. 9.2. Фотодиод (*а*) и схемы подключения фотодиода в качестве датчика наличия освещенности (*б* и *в*)

теризуется высоким выходным сигналом, а для схемы 9.2, *в* — низким. Подобные схемы не позволяют измерять уровень освещенности, а лишь фиксируют ее наличие, что говорит о необходимости подключения их к цифровым портам, которые воспринимают только два вида сигналов: высокий — 1 и низкий — 0.

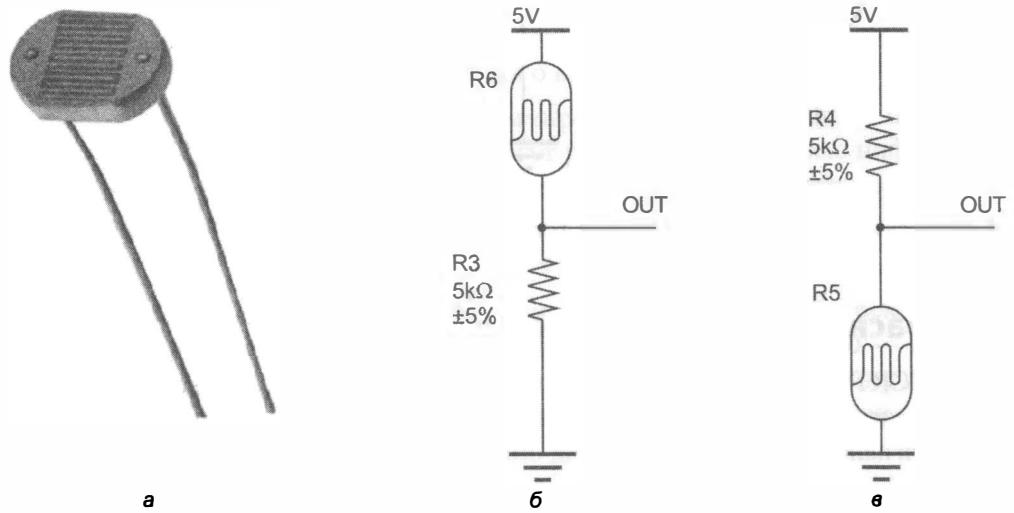
Чтобы фотодиод мог четко детектировать черную линию, нужно ее область подсветить, для чего, как правило, используется светодиод. Исходящий от него свет, отражаясь от светлой поверхности пола, попадает на фотодиод, вызывая в нем обратный ток. От черной же линии свет не отражается, что приводит к прекращению обратного тока фотодиода. Светодиод и фотодиод размещаются над исследуемой поверхностью так, чтобы отраженный от нее свет светодиода попадал на фотодиод. При этом расстояние до поверхности должно быть в пределах 1–2 см.

Фотодиоды и светодиоды производятся для разных спектров света, в том числе и невидимого, — например, инфракрасного диапазона.

Фоторезистор

Фоторезистор (рис. 9.3, *а*) под действием света изменяет свое сопротивление. При отсутствии освещения его сопротивление находится в пределах 1–200 МОм, а при освещении — уменьшается на несколько порядков. При этом фоторезистор имеет линейную зависимость сопротивления от освещенности, вследствие чего с его помощью удобно измерять освещенность в помещении.

Недостатками фоторезистора можно назвать достаточно высокое сопротивление и инертность — он не реагирует на изменение освещенности с частотой выше 1 кГц. Но при небольших скоростях движения робота использовать его все-таки можно. Схемы подключения фоторезистора приведены на рис. 9.3, *б* и *в*. Подсветка фоторезистора светодиодом и размещение их над исследуемой поверхностью организуются таким же образом, как и в случае применения фотодиода.



а *б* *в*

Рис. 9.3. Фоторезистор (*а*) и схемы подключения фоторезистора в качестве датчика освещенности (*б* и *в*)

Фототранзистор

Фототранзистор (рис. 9.4, *а*) не только преобразует световой поток в электрический ток, подобно фотодиоду или фоторезистору, но и многократно его усиливает. Визуально фототранзисторы могут не отличаться от фотодиодов, но подключаются и работают иначе. Схемы подключения фототранзисторов приведены на рис. 9.4, *б* и *в*. Основным преимуществом фототранзистора, как аналогового датчика, перед фотодиодом, представляющим собой пороговый, или цифровой, датчик, является возможность измерения уровня освещенности, что позволяет при определении цвета объекта фиксировать тона и полутона. Для этого следует подключать аналоговый выход OUT фототранзистора к аналоговым входам контроллера Arduino (A0–A5 или A0–A7 — в зависимости от модели контроллера). При этом робот должен будет экспериментально замерить уровень освещенности на белом фоне и уровень освещенности на черной линии, а затем проанализировать полученные значения в программе.

Подсветка фототранзистора светодиодом и размещение их над исследуемой поверхностью организуются таким же образом, как и в случае применения фотодиода.

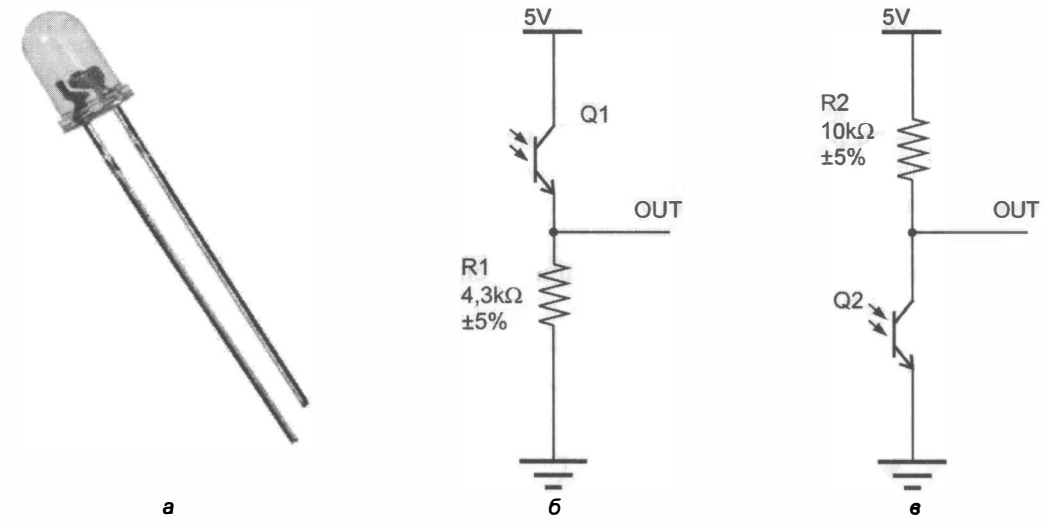


Рис. 9.4. Фототранзистор (*а*) и схемы подключения фототранзистора в качестве датчика освещенности (*б* и *в*)

Инфракрасный датчик отражения TCRT 5000

В датчике TCRT 5000 (рис. 9.5) использована готовая связка инфракрасного светодиода и фототранзистора. На его выходах можно получать как аналоговый, так и цифровой сигналы:

- *цифровой выход* датчика можно использовать для определения наличия черной линии, при этом его чувствительность настраивается переменным резистором;

- *на аналоговом выходе* датчика интенсивность отраженного сигнала зависит от типа и цвета поверхности, что также можно использовать для выявления наличия черной линии. При этом надо будет экспериментально замерить уровень освещенности на белом фоне и уровень освещенности на черной линии, а затем проанализировать полученные значения в программе.

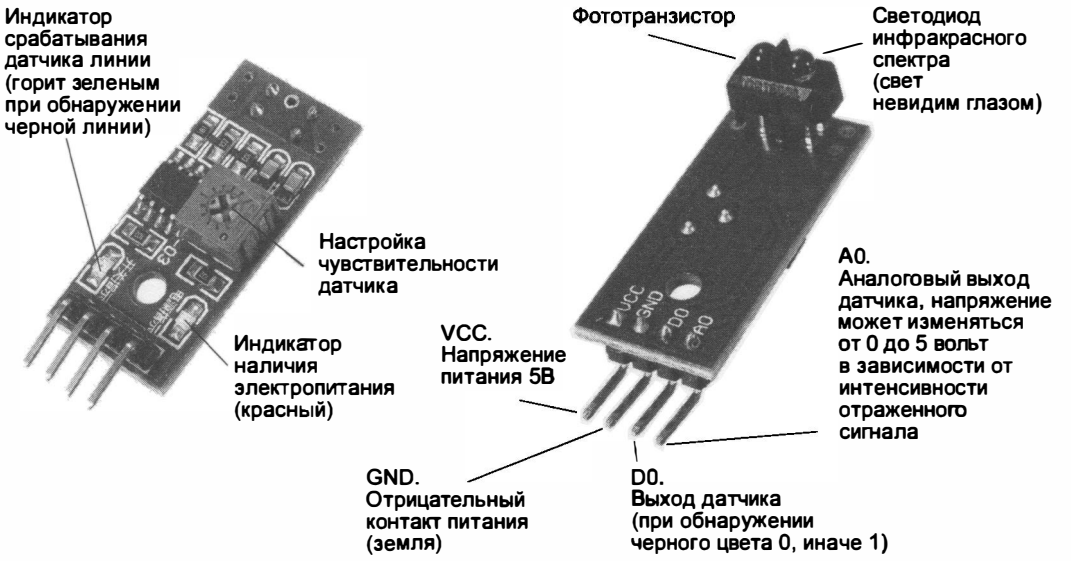


Рис. 9.5. Инфракрасный датчик отражения TCRT 5000: *а* — лицевая сторона; *б* — оборотная

Цифровой выход датчика TCRT 5000 обозначен на плате D0, а аналоговый — A0. Цифровой выход подключается к цифровому порту Arduino D0–D13, а аналоговый — к аналоговому A0–A5/A7. Контакт VCC подключается к питанию 5 В, контакт GND — соответственно, к «земле».

На плате датчика расположены два светодиода: зеленый, информирующий о наличии линии, и красный, информирующий о наличии электропитания.

Аналоговые порты A0–A7 при программировании в среде Arduino IDE имеют номера 14–21. При этом порты A0–A5 могут работать как обычные цифровые порты. Порты A6 и A7 распаяны не на всех платах, и они могут работать только как порты аналогового ввода.

* * *

В конечном итоге, для определения наличия черной линии я бы рекомендовал использовать именно инфракрасный датчик отражения TCRT 5000 — как наиболее удобное и готовое решение, для которого мы и составим программу движения робота. При этом мы воспользуемся цифровым выходом датчика, как наиболее простым с точки зрения обнаружения линии: для него в программе не потребуется анализировать уровень освещенности, а регулировка будет проходить на месте (трассе) подстройкой резистора.

Подготовка робота: установка датчиков

При движении по непрерывной линии двух датчиков — справа и слева относительно бампера робота — вполне достаточно, чтобы на этой линии удержаться. Датчики TCRT 5000, как правило, уже имеют припаянные контакты для разъемных соединений. Для подключения датчиков через сервисную плату потребуются провода с разъемами «мама-мама» (рис. 9.6).

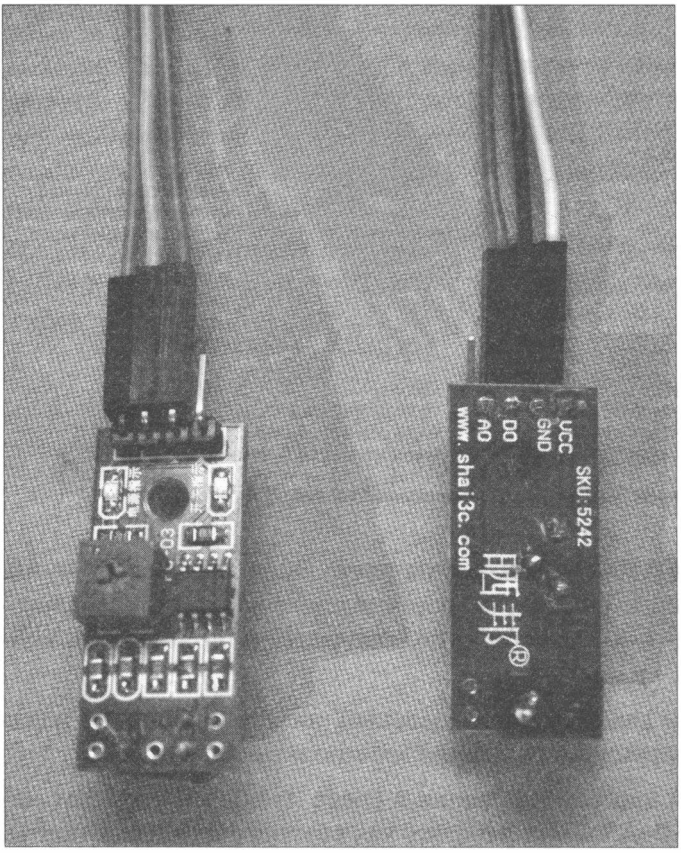
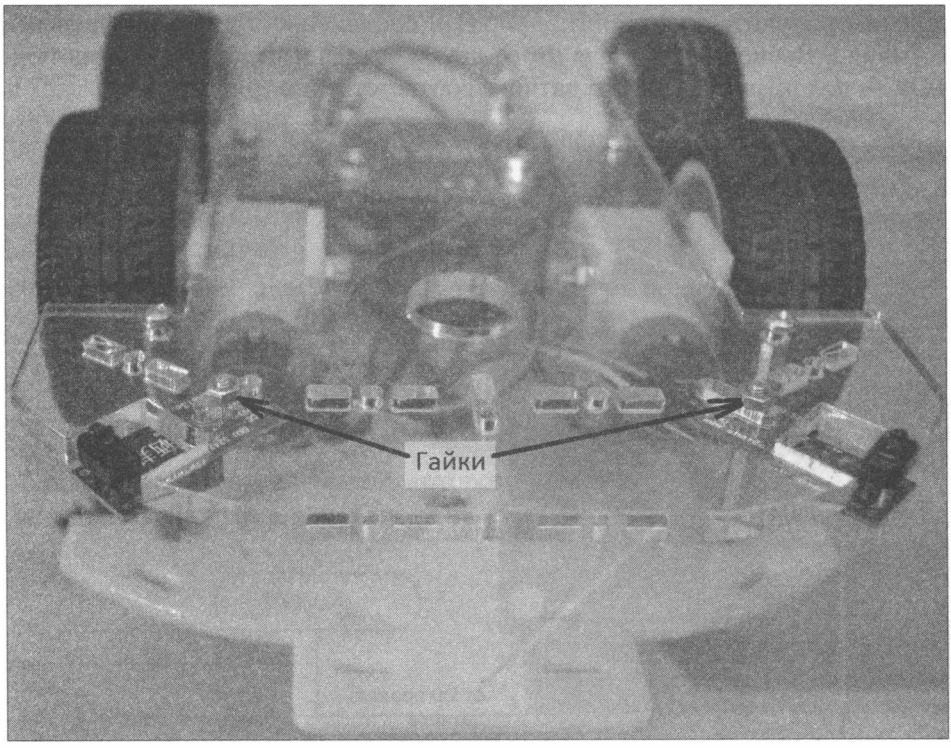


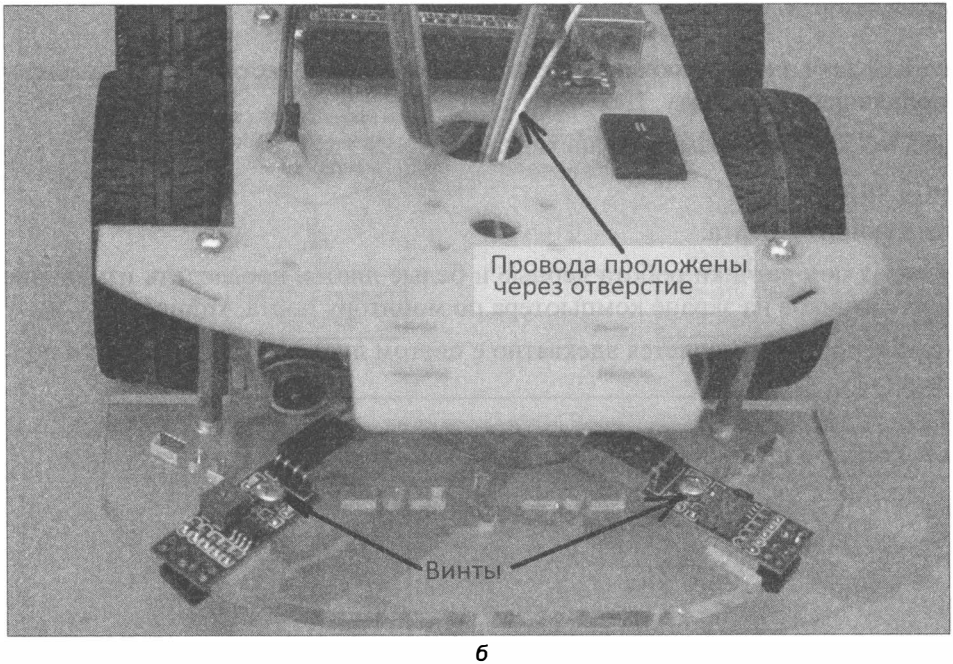
Рис. 9.6. Пара датчиков TCRT 5000, готовых к установке на робота

Подсоединим провода к датчикам и прикрепим датчики на робота. Для настройки чувствительности доступ к подстроечным резисторам датчиков должен быть свободным. Если по конструктивным особенностям вашего робота доступ не возможен, следует настроить датчики заранее, для чего подключить их к питанию, установить на рабочее расстояние над черной линией/белым фоном и добиться четкого срабатывания, подкладывая под датчики черные и белые листы.

Итак, крепим датчики к нижней панели робота 3-мм винтом с гайкой (рис. 9.7, *а*), предварительно проложив провода до платы Arduino Sensor Shield (рис. 9.7, *б*).



а



б

Рис. 9.7. Робот с установленными датчиками TCRT 5000: *а* — вид на робота снизу; *б* — вид на робота сверху

Датчики должны получить электропитание 5 В и соединение с двумя цифровыми выводами Arduino. Воспользуемся для этого двумя оставшимися свободными портами D8 и D9 — для правого и левого датчика соответственно (рис. 9.8).

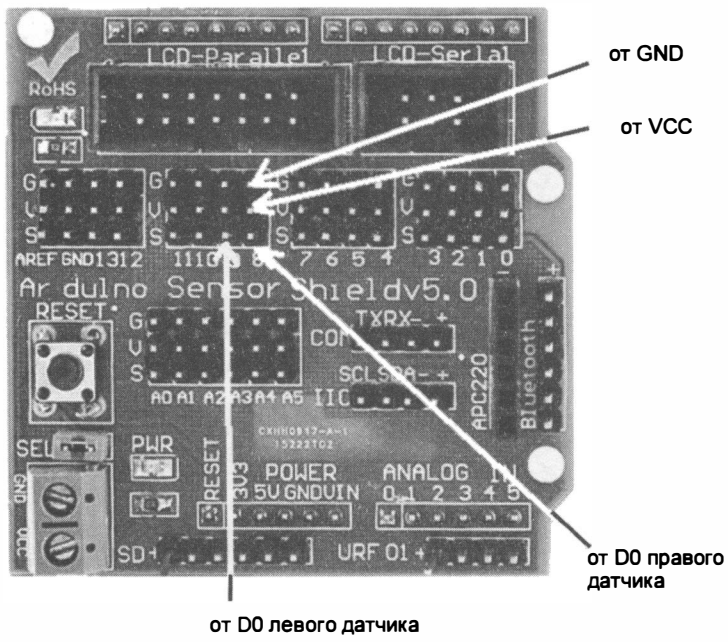


Рис. 9.8. Порядок подключения датчиков TCRT 5000 к Arduino Sensor Shield v5.0

После того как датчики подключены, желательно провести тестирование правильности их подключения к роботу. Для этого следует:

1. Загрузить в робота программу из листинга 9.1.
2. Включить питание.
3. Запустить монитор порта.
4. Подкладывая под датчики робота черные и белые листы, проследить изменение состояния датчиков на экране компьютера по монитору порта Arduino.

Если состояние портов изменяется адекватно с цветом листов, значит, датчики подключены верно.

Листинг 9.1. Тестовая программа проверки подключения датчиков линии

```
// Переменные с номерами пинов, к которым подключены датчики.
int left_sensor_line = 9;
int right_sensor_line = 8;
void setup()
{
  // Перевод пинов в состояние ввода данных.
  pinMode(left_sensor_line, INPUT);
  pinMode(right_sensor_line, INPUT);
```

```
  // Устанавливаем скорость порта связи с ПК.
  Serial.begin(9600);
}
// Основная программа.
void loop()
{
    Serial.print("Left sensor =");
    Serial.println(digitalRead(left_sensor_line));
    Serial.print("Right sensor =");
    Serial.println(digitalRead(right_sensor_line));
    Serial.println("=====================================");
    delay(500);
}
```

Теперь рассмотрим алгоритм, которого необходимо придерживаться при следовании по извилистой линии (рис. 9.9). Считаем, что в начале движения линия нахо-

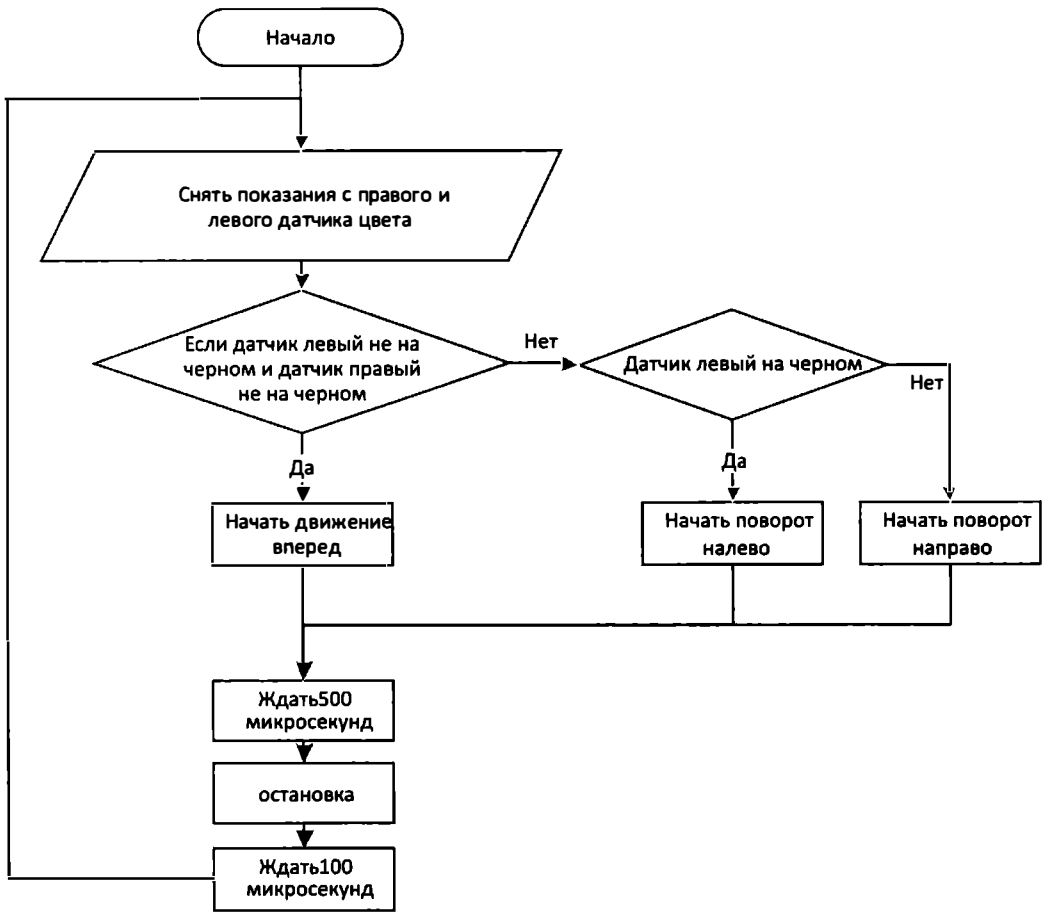


Рис. 9.9. Простейший алгоритм движения по черной линии

дится между правым и левым датчиком, т. е. робот установлен на линию. Пока оба датчика показывают состояние «светлый фон», робот будет ехать прямо. Если левый датчик показал «черный фон», робот должен повернуть налево. Если правый датчик показал «черный фон», робот должен повернуть направо.

Робот довольно тяжелый и может по инерции проскочить поворот и сбиться с линии. Поэтому следует отрегулировать скорость его движения, для чего в программу добавлены периодические остановки робота. Кроме регулировки скорости, подобные микроостановки позволяют более точно снимать показания с датчиков.

Программа движения робота с учетом указанных обстоятельств приведена в листинге 9.2.

Периоды работы двигателей и остановок следует подобрать экспериментально, поскольку они должны быть различными для разных моторов и массы робота.

Листинг 9.2. Движение робота по черной линии

```
#include "motor.h"
unsigned long _time;
//==================================
int left_sensor_line = 9;
int right_sensor_line = 8;
//==================================
void setup()
{
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); // Двигатели остановлены.
  // Сенсорные пины переводим в состояние ввода данных.
  pinMode(left_sensor_line, INPUT);
  pinMode(right_sensor_line, INPUT);
  _time = millis();
  // Устанавливаем скорость порта связи с ПК.
  Serial.begin(9600);
}
// Основная программа.
void loop()
{
  while (true)
  {
    bool ls = digitalRead(left_sensor_line);
    bool rs = digitalRead(right_sensor_line);
    //   Serial.println(ls);
    //   Serial.println(rs);
    //   Serial.println("=====================================");
    //  delay(500);
    // Готов к приему.
```

```
    if ((!ls) && (!rs))
    {
      forward();
    }
    else
    {
      if (ls)
      {
        left();
      }
      else
      {
        right();
      }
    }
    delayMicroseconds(500);
    _stop();
    delayMicroseconds(100);
  }
}
```

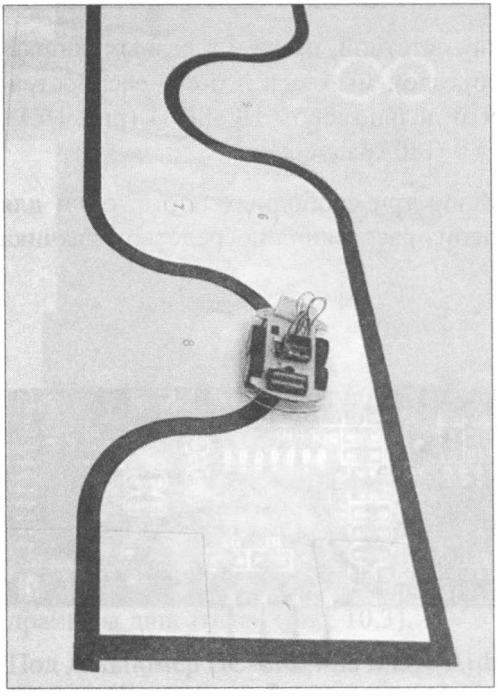


Рис. 9.10. Робот, двигающийся по черной линии

Выводы

В этой главе мы рассмотрели ряд датчиков освещенности, выбрали наиболее подходящий и оснастили им нашего робота. Теперь он может самостоятельно двигаться по черной линии (рис. 9.10).

А мы переходим к следующему проекту, которому посвящена *глава 10*, — разберемся с работой ультразвукового датчика расстояния и научим робота вращать головой, как локатором.

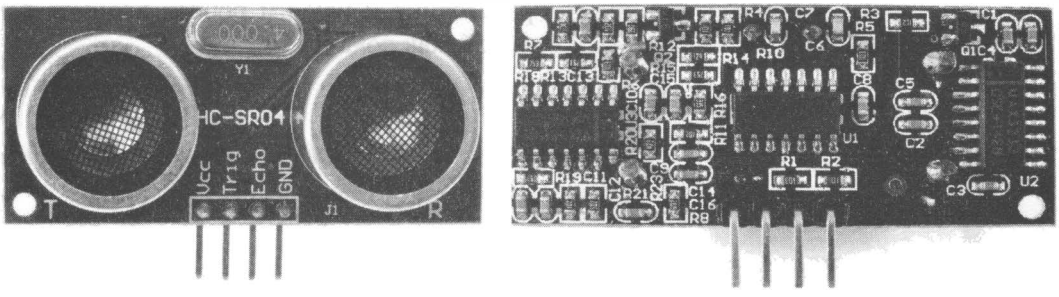
ГЛАВА 10
ПОВОРОТНАЯ ГОЛОВА

Сделаем робота более интеллектуальным, поместив на него дальномер и научив реагировать на изменение его показаний.

Ультразвуковой дальномер HC-SR04

Чтобы робот мог определять расстояния до препятствий, причем в разных направлениях (а это может потребоваться при их объезде), мы оснастим его распространенным, простым и надежным ультразвуковым дальномером HC-SR04 (рис. 10.1), установив его на сервомотор с углом поворота 0–180 градусов.

В этой задаче мы задействуем на плате Arduino три свободных порта: один для управления сервоприводом и два для определения расстояния посредством датчика HC-SR04.



а *б*

Рис. 10.1. Ультразвуковой дальномер HC-SR04: *а* — вид спереди; *б* — вид сзади

Схема подключения

Монтажная схема подключения дальномера и поворотного сервомотора к плате Arduino без использования коммутационной платы Arduino Sensor Shield v5.0 приведена на рис. 10.2.

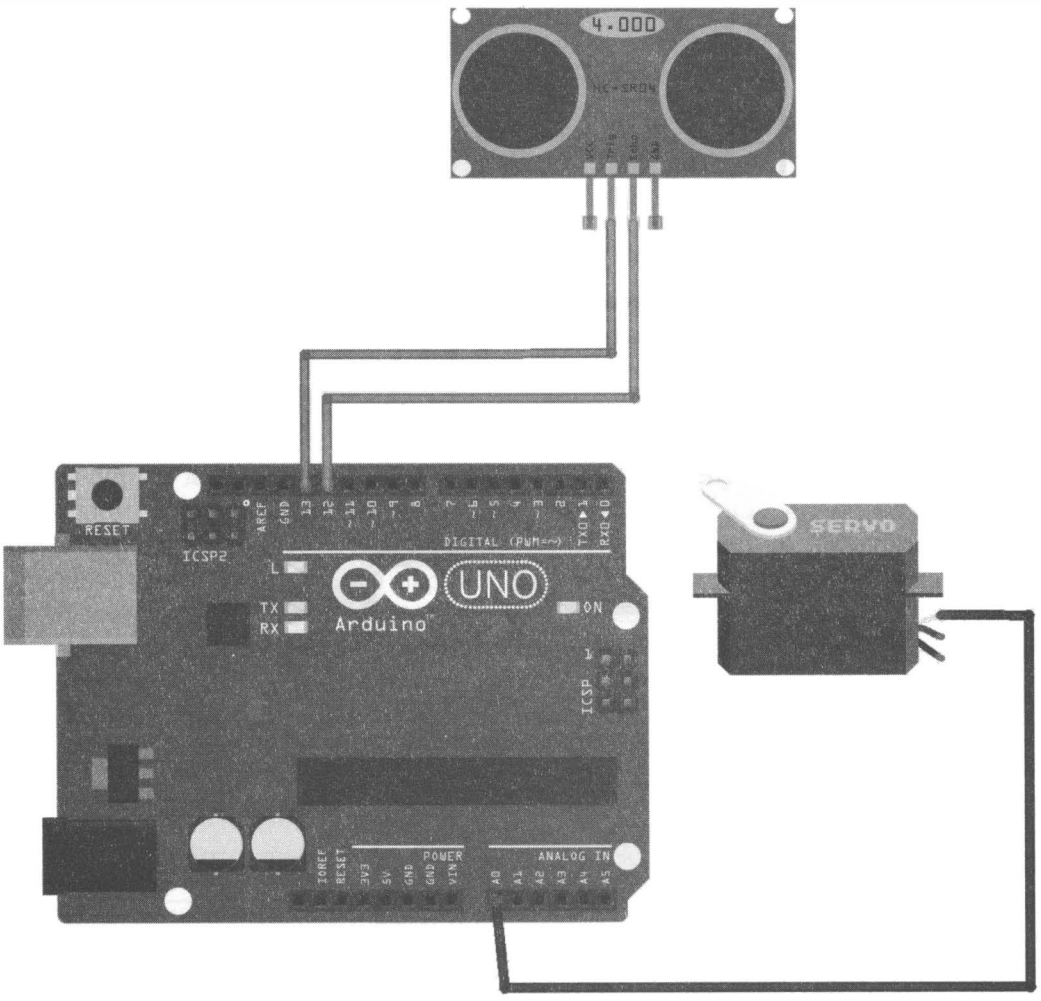


Рис. 10.2. Монтажная схема подключения сервомотора и дальномера

Питание 5 вольт для сервомотора и дальномера следует взять с контакта +5V платы драйвера двигателей (рис. 10.3).

Под дальномер HC-SR04 на плате Arduino будут заняты два порта: D13 (Trig) и D12 (Echo). Управление сервомотором, поворачивающим голову, будет производиться с порта A0 платы Arduino. Порт A0 хоть и является аналоговым, но позволяет работать и в цифровом (двоичном) режиме ввода/вывода, что и требуется в данном случае.

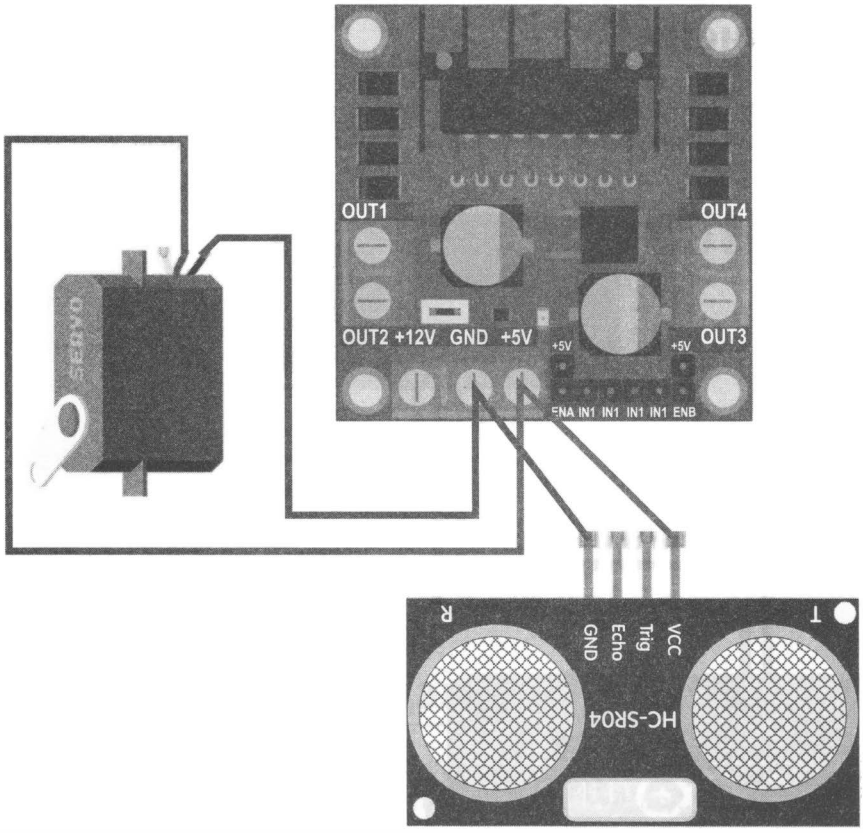


Рис. 10.3. Монтажная схема цепей питания сервомотора и дальномера

Рис. 10.4. Подключение дальномера и сервомотора к Arduino Sensor Shield v5.0

При использовании Arduino Sensor Shield v5.0 (рис. 10.4) потребуется четыре коммутационных провода с разъемами «мама-мама» для подключения дальномера, сервомотор же не нуждается в дополнительных проводах, поскольку его стандартная колодка как раз подходит к контактам коммутационной платы. Но стоит обратить внимание на то, чтобы не перевернуть колодку в обратную сторону, перепутав G (GND) и S (Signal), — это может привести к порче либо мотора, либо контроллера Arduino.

Измерение расстояния

Ультразвуковой дальномер посылает импульс, этот импульс отражается от препятствия и возвращается через некоторое время. Время между импульсом и его возвратом, умноженное на скорость звука и поделенное на два, даст расстояние.

Выделим для работы с ультразвуковыми функциями отдельный файл — это позволит не путаться в функциях и быстро находить нужное, дадим ему имя sonar.h (листинг 10.1). Объявим в этом файле две глобальные переменные: `Trig` и `Echo` — для хранения номеров портов подключения одноименных контактов дальномера. Там же создадим функцию `Sonar_init(int Tr, int Ec)`, которая будет запоминать значения портов для входов дальномера и задавать им требуемые режимы работы. Измерением займется функция `int Sonar(unsigned long Limit)` — она будет обращаться к сонару и по его данным вычислять расстояние до препятствия. У этой функции всего один входной параметр, влияющий на время ожидания отражения от препятствия, — это максимальное измеряемое расстояние в сантиметрах, при превышении которого функция будет возвращать 0. Ограничение измеряемого расстояния позволит ускорить работу функции, иначе она может ждать возврата ультразвукового эха от препятствия целую секунду.

Файл sonar.h следует скопировать в рабочий каталог текущей программы и подключить инструкцией `#include "sonar.h"`.

Листинг 10.1. Содержимое файла sonar.h

```
// Номера пинов сонара.
int Trig;
int Echo;
//=Режим пинов/портов ==============================
Sonar_init(int Tr, int Ec)
{
   Trig=Tr;
   Echo=Ec;
   // Задаем режимы работы пинов.
   pinMode(Trig, OUTPUT);
   pinMode(Echo, INPUT);
}
```

```
//= Измерение расстояния ============================
 int Sonar(unsigned long Limit)
{
  int Long_cm;
  // Переводим лимит из сантиметров в относительные величины (микросек.)
  unsigned long Lim=Limit*58;
  // Генерируем импульс
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);
  // Ждем отражения импульса.
  // Пересчитываем время в расстояние (по скорости звука).
  Long_cm = int(pulseIn(Echo, HIGH, Lim)/58);
  // Если не дождались отражения,
  // присваиваем значение максимально измеряемому.
  if(Long_cm==0) return int(Limit);
  return Long_cm;
}
```

Подключим сонар HC-SR04 к роботу, а робота — к компьютеру и протестируем возможности измерения расстояния. Для этого подойдет программа из листинга 10.2. В ней дальномеру назначаются порты 13 и 12, создается связь с ПК, а в основной программе с периодом 1,5 секунды измеряется расстояние до препятствия и полученное значение передается в порт ПК. В оболочке Arduino IDE следует открыть монитор порта, тогда в окне монитора будет отображаться расстояние (рис. 10.5). Максимальное измеряемое расстояние в примере — 300 см.

Листинг 10.2. Программа измерения расстояний

```
// Подключаем библиотеку, управляющую дальномером.
#include "sonar.h"
void setup()
{
  // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
  // При инициализации задаем скорость порта для связи с ПК.
  Serial.begin(9600); // start the serial port
}
void loop()
{
  // В цикле loop отправляем значение,
  // полученное с дальномера, в порт через 1,5 с.
  // получаем дистанцию в сантиметрах с лимитом 300 см.
  int prepyatstvie = Sonar(300);
```

```
  Serial.print("Distance="); // оформляем вывод.
  Serial.print(prepyatstvie); // выводим дистанцию.
  Serial.println(" cm."); // оформляем вывод.
  delay(1500); // приостанавливаем программу.
}
```

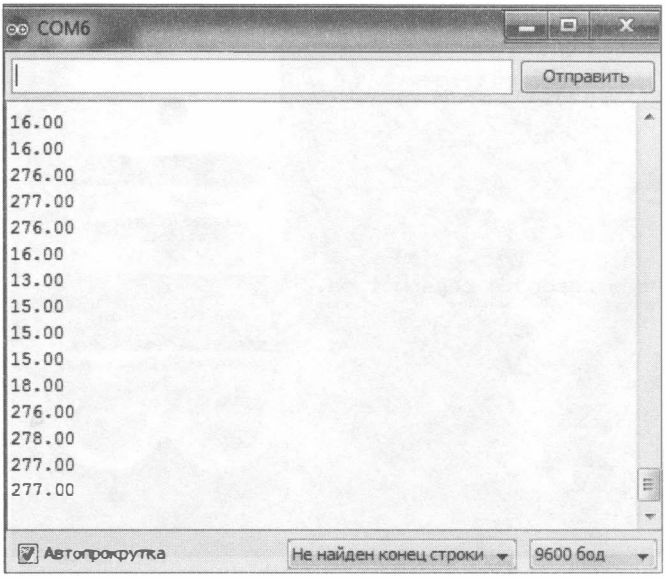


Рис. 10.5. Отображение измеренного расстояния в окне монитора порта

Управление сервомотором

Сервомотор требуется, чтобы робот поворачивал голову на шее. Для работы с сервомоторами в Arduino IDE имеется отдельная библиотека, при помощи которой порт контроллера может генерировать программный ШИМ на любом порту (кроме A6 и A7) с частотой 50 Гц. Принцип работы контроллера в режиме генерации ШИМ с частотой 50 Гц очень похож на рассмотренный в *главе 5*, но для генерации ШИМ не требуются порты с аппаратной широтно-импульсной модуляцией, — подойдет любой цифровой порт, так как генерация ШИМ будет производиться программно. От длины положительного фронта импульса при широтно-импульсной модуляции зависит угол поворота сервомотора. Простейшая программа, реализующая поворот вала сервомотора в разные стороны, представлена в листинге 10.3. Вал поворачивается рывками по 10 градусов сначала в одну, а затем в другую сторону. Задействуются значения углов поворота от 10 до 170 градусов (большие или меньшие значения на простых сервомоторах могут привести к заклиниванию).

> ***Берегите шестерни сервомоторов!***
>
> Сервомоторы SG90 имеют довольно хрупкие шестерни. Не следует с усилием вращать вал сервомотора руками — это может привести к разрушению шестерней! Сервомоторы с металлическими редукторами таких недостатков лишены.

Листинг 10.3. Программа поворота головы робота в разные стороны

```
// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Создаем сервомотор поворота головы.
Servo neck;
void setup()
{
  // Инициализируем сервомотор, управление 14-м портом.
  neck.attach(14);
}
void loop()
{
  int i;
  // Создаем цикл для равномерного поворота сервомотора.
  for(i=10; i<=170; i=i+10)
  {
    // Поворачиваем вал на угол i.
    neck.write(i);
  // приостанавливаем программу.
  delay(100);
  }
  // Возвращаем голову назад.
  for(i=170; i>=10;i=i-10)
  {
    // Поворачиваем вал на угол i.
    neck.write(i);
  // приостанавливаем программу.
  delay(100);
  }
}
```

Монтаж головы

Если голова еще не сидит на роботе, то пора ее установить. Варианты крепления головы могут различаться, требуемые в нашем примере детали приведены на рис. 10.6: это сонар, сервомотор с крестообразным рычажком и крепежным винтом, пластина крепления сервомотора, четыре шестигранных стойки с гайками, конструкционные детали крепления сонара, четыре 3-мм винта и два 2-мм винта с гайками для крепления сервомотора.

Установку головы следует проводить в следующем порядке:

1. Установим четыре шестигранных стойки и закрепим их снизу гайками, как показано на рис. 10.7.
2. Вставим сервомотор в прямоугольное отверстие пластины крепления сервомотора и зафиксируем двумя 2-мм винтами с гайками.

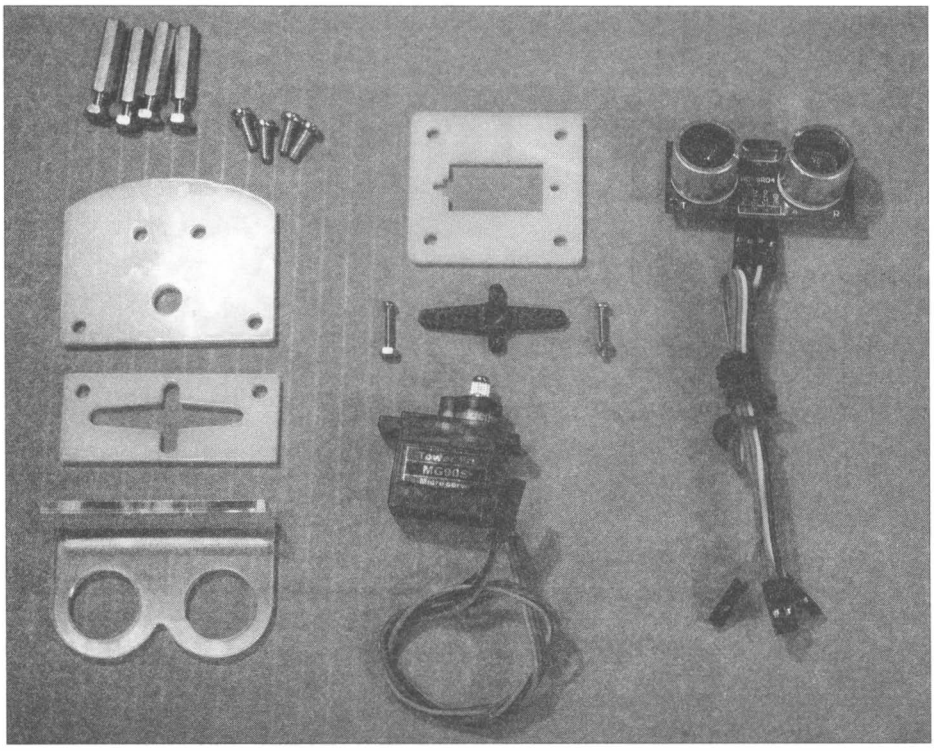


Рис. 10.6. Детали для сборки головы

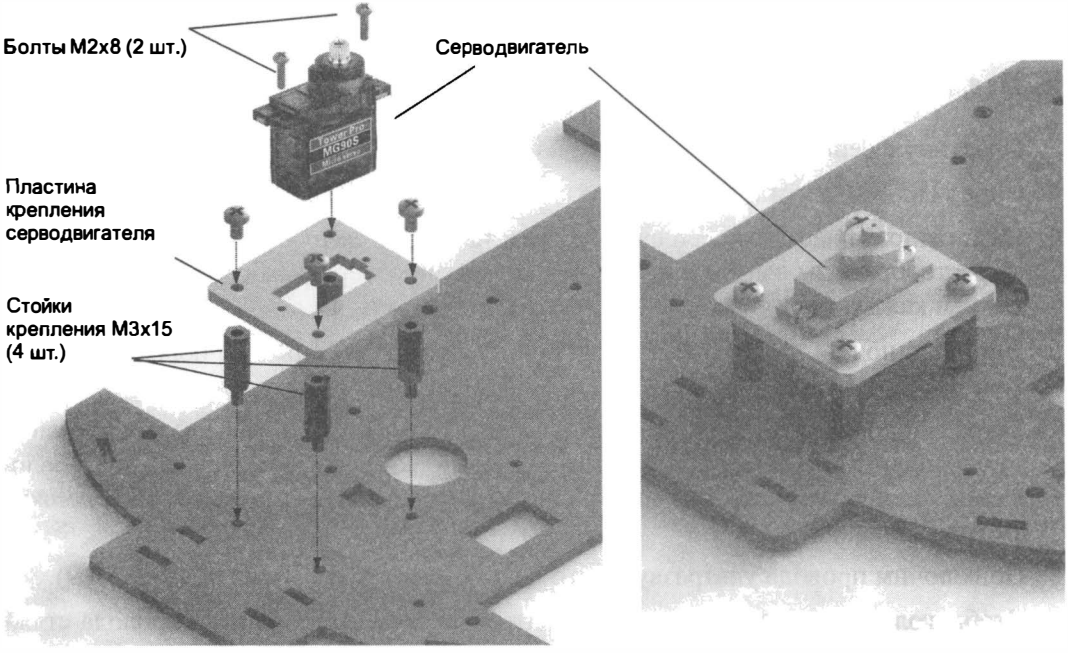


Рис. 10.7. Установка сервомотора

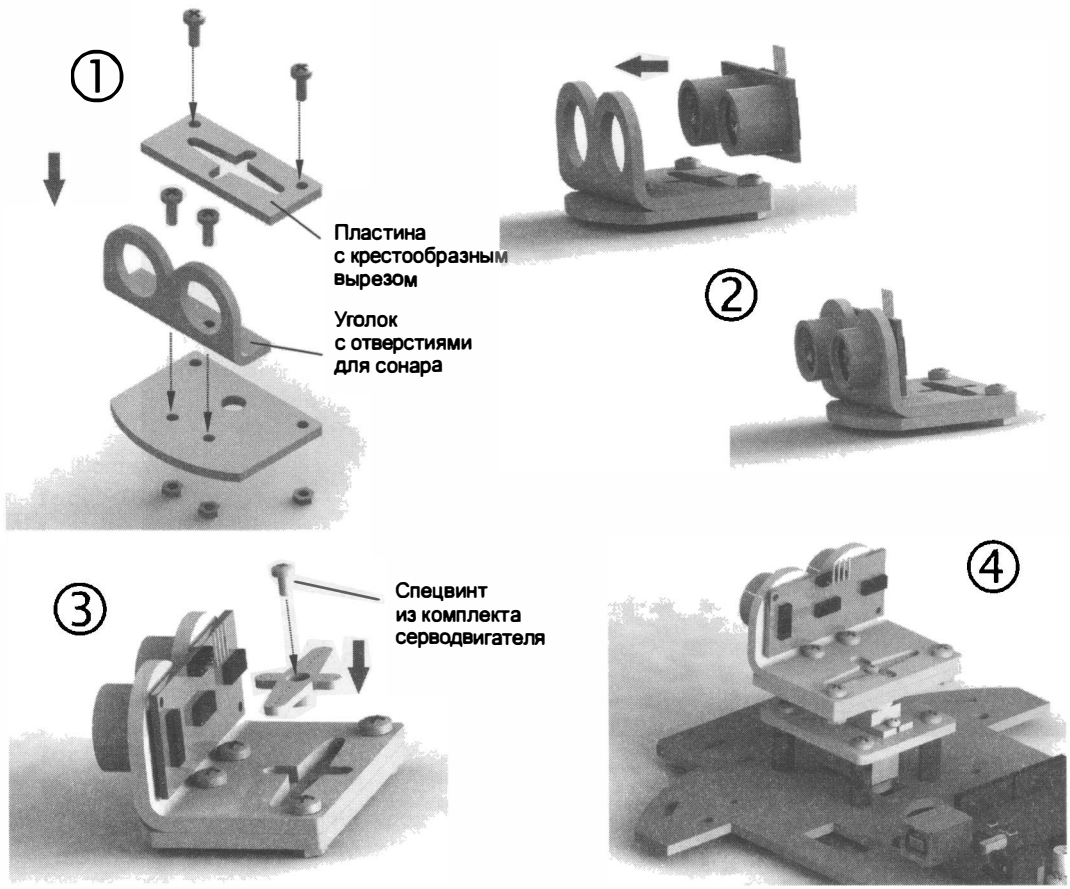


Рис. 10.8. Установка головы на робота

3. Пластину с сервомотором закрепим на стойках четырьмя 3-мм винтами.
4. Сонар HC-SR04, установленный на серводвигатель, показан на рис. 10.8.
5. На нижнюю пластину крепления сонара наложим сверху две детали: пластину с крестообразным вырезом и уголок с круглыми отверстиями для сонара; зафиксируем каждую деталь двумя винтами М3 с гайками.
6. Программно (`neck.write(90);`) установим сервомотор в среднее положение (листинг 10.4).
7. Закрепим собранную конструкцию на оси сервомотора спецвинтом из комплекта серводвигателя. После программной установки сервомотора в среднее положение голову следует установить ровно в центральное положение (робот должен смотреть вперед).
8. Подключим провода ультразвукового дальномера к плате Arduino (рис. 10.9).
9. Чтобы голове ничего не мешало вращаться, стянем болтающиеся провода стяжками. Стянутые в пучки провода реже подвергаются изгибам и реже переламываются. Теперь робот готов (рис. 10.10).

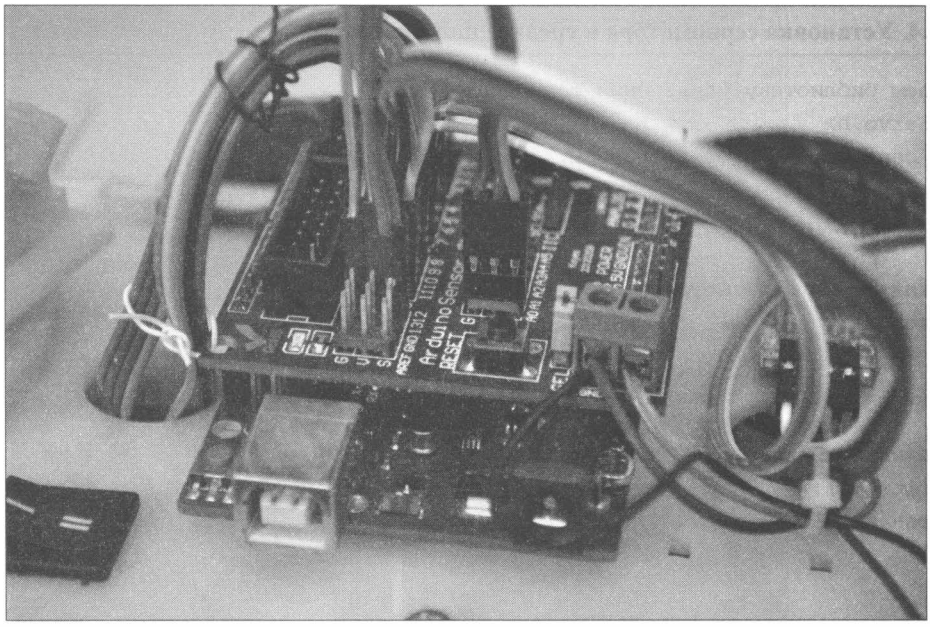


Рис. 10.9. Подключение проводов дальномера HC-SR04 к плате Arduino Sensor Shield v5.0

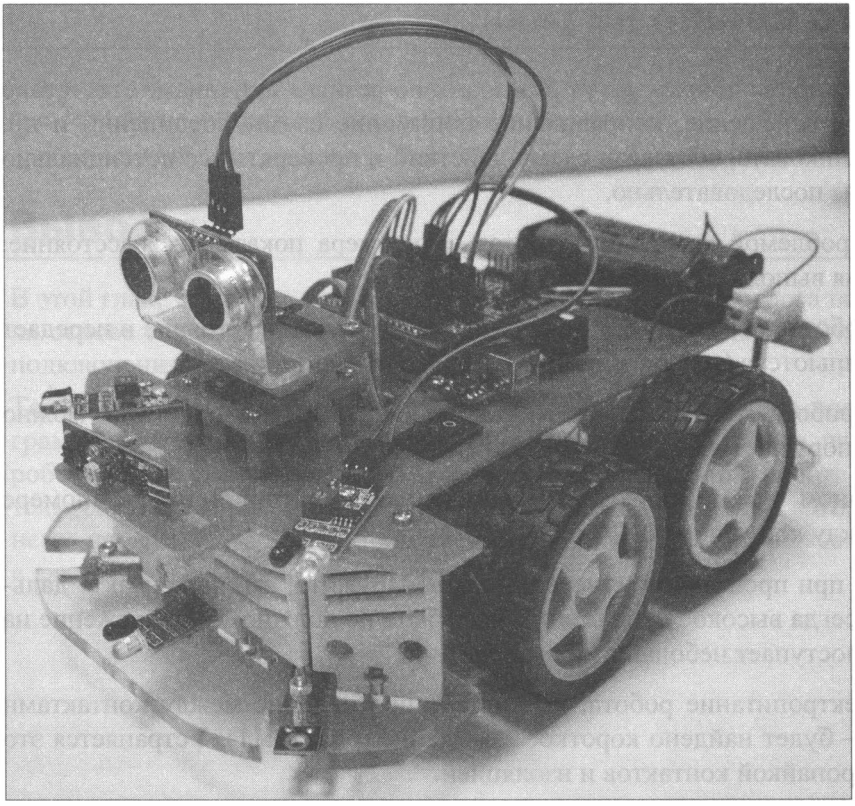


Рис. 10.10. Робот с установленной головой

Листинг 10.4. Установка сервомотора в среднее положение

```
// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Создаем сервомотор поворота головы.
Servo neck;
void setup()
{
  // Инициализируем сервомотор, управление 9-м портом.
  neck.attach(9);
}
void loop()
{
  // Поворачиваем вал на угол 90.
    neck.write(90);
  // приостанавливаем программу.
  delay(100);
}
```

Если что-то пошло не так...

Неисправности во время сборки могут возникать по разным причинам: отсутствие внимания, плохое освещение, неправильное понимание схемы соединений и пр. Главное — не паниковать, составить схему действий и проверять все потенциально неисправные узлы последовательно.

Так, основной проблемой может стать отказ дальномера показывать расстояние. Для ее устранения выполните следующие шаги:

1. Загрузите в робота программу, которая только определяет расстояние и передает его в порт компьютера (см. листинг 10.2).
2. Подключите робота к компьютеру, в Arduino IDE откройте порт. Как можно видеть, через порт поступают нулевые значения.
3. При включенном роботе проверьте наличие электропитания на дальномере HC-SR04 (между контактами VCC и GND).
4. Скорее всего, при проведении измерений вы обнаружите, что на контакте дальномера Trig всегда высокое напряжение, чего быть не должно, — напряжение на этот контакт поступает небольшими импульсами.
5. Отключив электропитание робота, измерьте сопротивление между контактами VCC и Trig — будет найдено короткое замыкание (рис. 10.11). Устраняется это тщательной пропайкой контактов и изоляцией.
6. Все, неисправность устранена, дальномер HC-SR04 начал работать правильно.

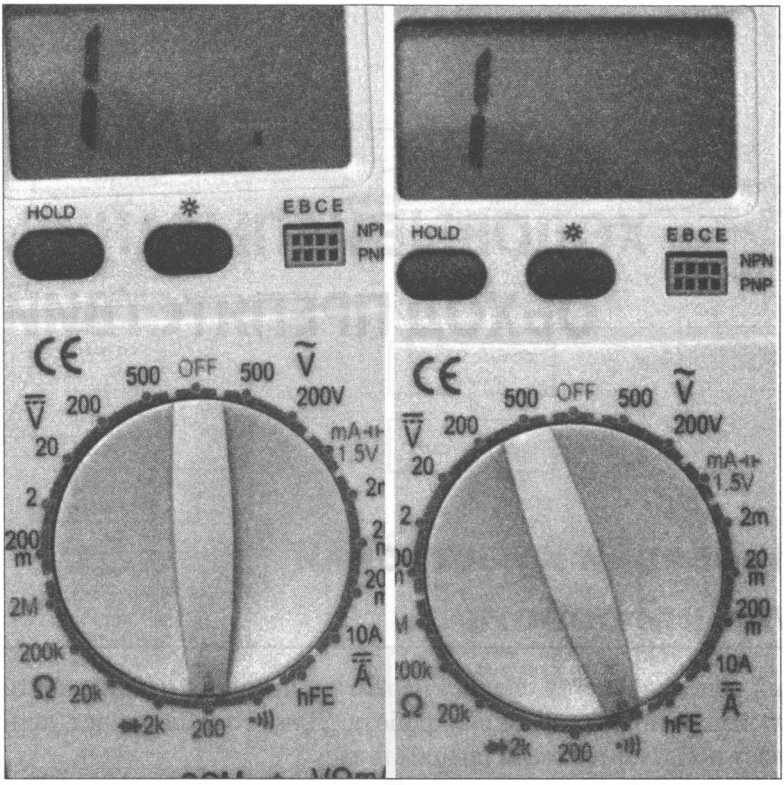


Рис. 10.11. Мультиметр при определении короткого замыкания

Выводы

В этой главе подробно рассмотрен процесс установки на робота поворотной головы на основе сервомотора с ультразвуковым дальномером HC-SR04, а также порядок подключения этих компонентов.

Теперь робот имеет возможность объезжать препятствия, но пока в нем нет программы, которая анализирует информацию о расстоянии до препятствия и дает роботу соответствующие команды, — проделанная нами работа была подготовительной. Пора переходить к ходовым испытаниям: составить алгоритм, написать по нему программу, а затем произвести настройку параметров, чем мы и займемся в следующей главе.

ГЛАВА 11

ХОДОВЫЕ ИСПЫТАНИЯ: ОБХОД ПРЕПЯТСТВИЙ

Программа проверки и настройки основных функций робота

После устранения явных неисправностей следует перейти к проверке правильности работы и тонкой настройке основных функций робота. Для этого подойдет программа, которая позволяет роботу объезжать препятствия.

Придумаем простой и понятный алгоритм объезда препятствий:

1. Вращаем головой и измеряем расстояние до препятствий слева, справа и впереди.
2. Сравниваем расстояния, выбираем наибольшее.
3. Туда и едем.
4. Если ехать нельзя, тупик — разворачиваемся.

Подробная схема алгоритма приведена на рис. 11.1.

Реализуем этот алгоритм в виде программы (листинг 11.1), опираясь на опыт разработки программ в *главах 7–10*. Листинг 11.1 снабжен комментариями и соответствует алгоритму, приведенному на рис. 11.1.

Следует заметить, что наша программа не является идеальной программой обхода препятствий — в частности, ее эффективность зависит от размеров робота. Программа тестировалась на роботе размером 17×14 см. Возможно, что большой робот будет застревать, заезжая в узкие проходы. В этом случае стоит самостоятельно поэкспериментировать с программой — например, добавить в нее проверку на возможность разворота, а если окажется, что разворот невозможен, научить робота включать задний ход. В программу заложено условие, разрешающее движение, если расстояние до препятствия превышает 10 см. Попробуйте это расстояние увеличить до 20–30 см. Подойдите к процессу тестирования робота и программирования его настроек творчески!

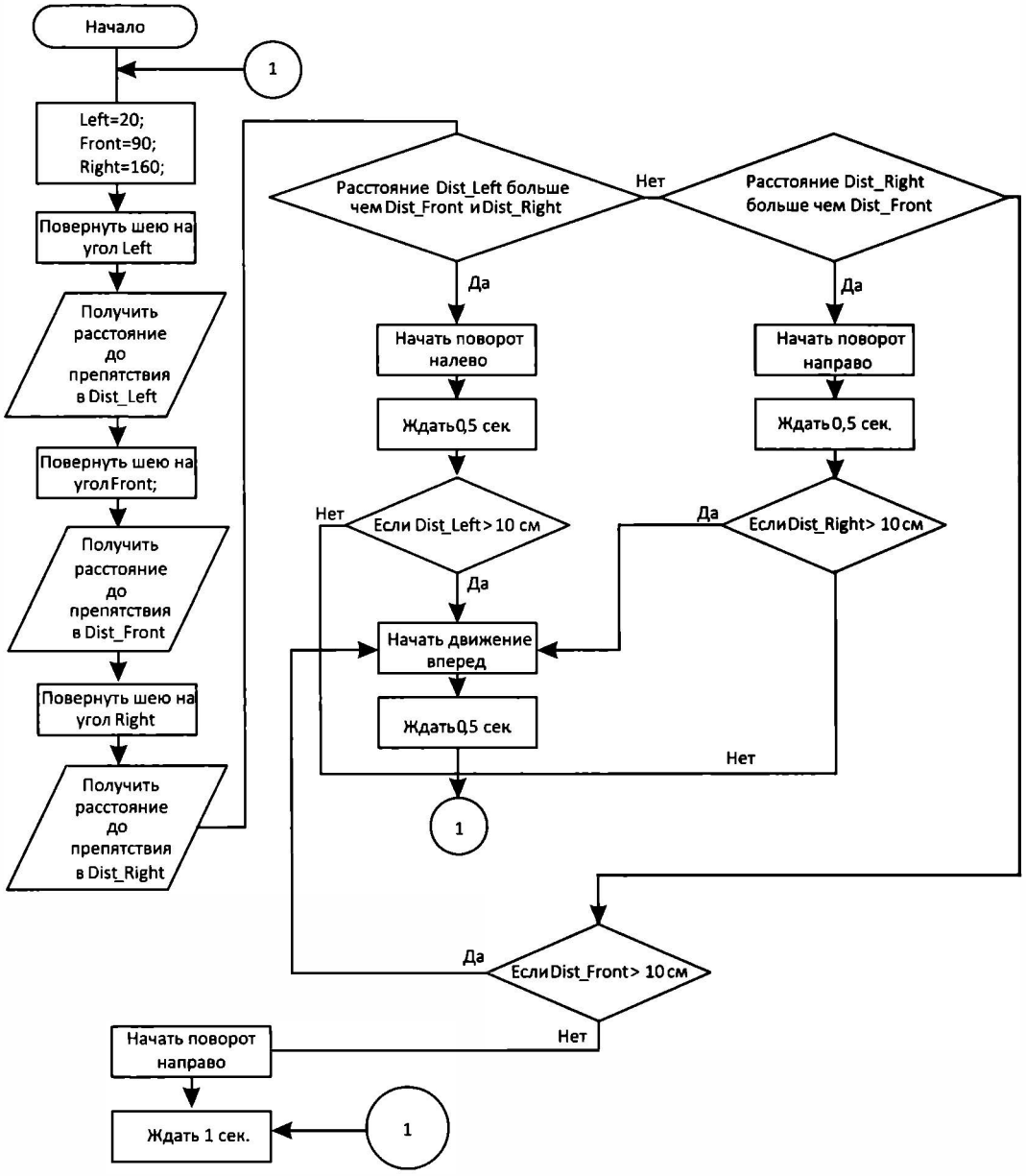


Рис. 11.1. Алгоритм программы обхода препятствий роботом

Листинг 11.1. Программа обхода препятствий роботом

```
// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Подключаем библиотеку, управляющую моторами.
#include "motor.h"
```

```
// Подключаем библиотеку, управляющую дальномером.
#include "sonar.h"

// Создаем сервомотор поворота головы.
Servo neck;
// Константы - постоянные значения для уточнения углов.
const int left_ang = 168;
const int front_ang = 98;
const int right_ang = 28;
// Временные константы служат для точного задания времени на поворот,
// разворот, движение вперед
// в миллисекундах.
const int time_90 = 390;
const int time_180 = 750;
const int time_10cm = 220;
void setup()
{
  // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
  // Инициализируем сервомотор, управление 9-м портом.
  neck.attach(14);
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
}
// Основная программа.
void loop()
{
  _stop();
  // Создаем переменные для хранения трех дистанций - слева, впереди, справа.
  int Dist_left, Dist_front, Dist_right;
  // Поворачиваем голову налево.
  neck.write(left_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(150);
  // Записываем расстояние до препятствия слева.
  Dist_left = Sonar(400);
  // Поворачиваем голову прямо вперед.
  neck.write(front_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(150);
  // Записываем расстояние до препятствия впереди.
  Dist_front =  Sonar(400);
  // Поворачиваем голову направо.
  neck.write(right_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(150);
```

```
  // Записываем расстояние до препятствия впереди.
  Dist_right = Sonar(400);
  neck.write(left_ang);
  // Если расстояние до препятствия слева наибольшее
  if ((Dist_left > Dist_front) && (Dist_left > Dist_right))
  {
    left(); // поворачиваем налево 0,5 секунд.
    delay(time_90);
    if (Dist_left > 10)
    {
      forward(); // едем вперед 0,5 секунды.
      delay(time_10cm);
    }
  }
  else
  {
    if (Dist_right > Dist_front)
    {
      right(); // поворачиваем направо 0,5 секунд.
      delay(time_90);
      if (Dist_right > 10)
      {
        forward(); // едем вперед 0,5 секунды.
        delay(time_10cm);
      }
    }
    else
    {
      if (Dist_front > 10)
      {
        forward(); // едем вперед 0,5 секунды.
        delay(time_10cm);
      }
      else
      {
        right(); // разворачиваемся.
        delay(time_180);
      }
    }
  }
}
```

Электронный архив

В папке listing_11_1m сопровождающего книгу электронного архива вы найдете модифицированную (альтернативную) версию этой программы — она немного отличается логикой (см. *приложение 2*).

Константы и постоянные времени

Для тонкой настройки программы нам потребуется экспериментально определить ряд значений:

1. Значения углов поворота сервомотора для прямого, правого и левого взглядов.

   Довольно сложно установить голову на вал точно прямо, но можно экспериментально подобрать и задать в программе нужный угол. Правый и левый повороты — это повороты головы направо и налево на 60–75 градусов, чего достаточно, чтобы робот смог определить расстояние до препятствия справа и слева соответственно. Высокая точность здесь не нужна, поскольку луч сонара довольно широкий, — около 30 градусов. Поверните сервомотор программно на 90 градусов. Если при этом робот смотрит прямо, примите значение угла для прямого взгляда `front_ang` равным 90 градусов. В противном случае уточните значение `front_ang` экспериментально.

2. Значения времени, за которые робот поворачивает на 90 градусов и на 180 градусов.

   Здесь экспериментально определяется время, за которое робот выполняет эти движения. На разных типах поверхности это время может быть различным.

3. Время, за которое с места робот проезжает 10 сантиметров.

Полученные значения заносятся в программу как константы с интуитивно понятными названиями (табл. 11.1).

Таблица 11.1. *Переменные и константы для программы обхода препятствий*

№ п.п.	Переменная или константа	Назначение	Содержимое функции `loop()`
1.	`left_ang`	Угол поворота сервопривода головы робота для поиска препятствий слева — 70° от прямого положения	`neck.write(left_ang);` `delay(1000);`
2.	`front_ang`	Угол поворота сервопривода головы робота для поиска препятствий прямо по ходу робота	`neck.write(front_ang);` `delay(1000);`
3.	`right_ang`	Угол поворота сервопривода головы робота для поиска препятствий справа — 70° от прямого положения	`neck.write(right_ang);` `delay(1000);`

4.	`time_90`	Время, за которое робот поворачивает на 90° из состояния покоя	`left();`
			`delay(time_90);`
			`_stop();`
			`delay(10000);`

5.	`time_180`	Время, за которое робот поворачивает на 180° из состояния покоя	`left();`
			`delay(time_180);`
			`_stop();`
			`delay(10000);`

Таблица 11.1 (окончание)

№ п.п.	Переменная или константа	Назначение	Содержимое функции loop()

6.	time_10cm	Время, за которое робот из состояния покоя преодолевает расстояние 10 см и полностью останавливается	forward();
			delay(time_10cm);
			_stop();
			delay(10000);

Значения функции `loop()`, содержащей основные действия программы из листинга 11.1, можно для своего робота уточнить, если удалить из этой функции все строки, а затем поочередно вставлять в нее строки, представленные в 4-м столбце табл. 11.1. Изменяя в программе значения указанных параметров, нужно добиться удовлетворительного результата.

Например, если команда `neck.write(front_ang)` приводит к тому, что робот смотрит немного вбок, то следует, понемногу увеличивая или уменьшая значение `const int front_ang=90`, добиться того, чтобы робот смотрел прямо. Полученное значение может отличаться от 90, и его следует запомнить и применить в программе. Аналогично надо поступить для поворота головы влево и вправо.

Для поворотов робота на 90 градусов нужно определить время, за которое робот совершает поворот на указанный угол. Стоит заметить, что это время будет отличаться для разного вида поверхностей, по которым движется робот. Если робот поворачивается на угол больший, чем 90°, следует это время уменьшать, а если угол поворота мал, то увеличивать. В программе используется поворот на 90 градусов, но можно применять и меньшие значения углов.

Время движения вперед/назад на заданное расстояние определяется аналогично. В программе это расстояние задано величиной 10 см, но вы можете его увеличить.

Отладка программы

Проверку правильности функционирования робота рекомендуется производить в следующем порядке:

1. Снять с робота колеса.
2. Загрузить программу из листинга 11.1 и дополнить ее следующими тестовыми командами согласно листингу 11.2 (требуемые изменения в листинге 11.2 выделены полужирным шрифтом):
   - в разделе `setup()` организовать вывод в порт;
   - в основной программе `loop()` вставить в интересующих нас местах вывод служебной информации.
3. Запрограммировать робота и оставить его соединенным кабелем с ПК.
4. В Arduino IDE открыть окно **Монитор порта**.

5. Создавая перед роботом различные варианты препятствий, соответствующие определенной ситуации в программе, проверить правильность реакции робота по выводу сообщений в окно **Монитор порта** и по вращению колес, — робот при этом будет выдавать в порт расстояния до препятствий и этапы работы программы (какой блок выполняется).
6. Если робот действует не верно, проанализировать его работу и внести в программу необходимые правки.
7. Можно также скорректировать время, которое требуется роботу для поворота головы (в листинге 11.2 это время сильно увеличено до 1,5 сек.), — этого времени должно быть достаточно для поворота головы, но не приводить к простою робота. В то же время, если голова еще вращается, а робот уже начал измерять расстояние до препятствия, результат измерения окажется неверным.

Листинг 11.2. Отладочная (неполная) программа обхода препятствий с выводом в порт

```
// Отладочная программа.
void setup()
{
  // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
  // Инициализируем сервомотор, управление 9-м портом.
  neck.attach(14);
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); // Двигатели остановлены.
  Serial.begin(9600);
}
// Основная программа.
void loop()
{
  _stop();
  // Создаем переменные для хранения трех дистанций - слева, впереди, справа.
  int Dist_left, Dist_front, Dist_right;
  // Поворачиваем голову налево.
  neck.write(left_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(1500);
  // Записываем расстояние до препятствия слева.
  Dist_left = Sonar(400);
  Serial.print("Dist_left="); // оформляем вывод.
  // выводим дистанцию.
  Serial.print(Dist_left);
  Serial.println(" см."); // оформляем вывод.
```

```
  // приостанавливаем программу.
  delay(500);
  // Поворачиваем голову прямо вперед.
  neck.write(front_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(1500);
  // Записываем расстояние до препятствия впереди.
  Dist_front = Sonar(400);
  Serial.print("Dist_front="); // оформляем вывод.
  // выводим дистанцию.
  Serial.print(Dist_front);
  Serial.println(" cm."); // оформляем вывод.
  // приостанавливаем программу.
  delay(500);
  // Поворачиваем голову направо.
  neck.write(right_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(1500);
  // Записываем расстояние до препятствия впереди.
  Dist_right = Sonar(400);
  Serial.print("Dist_right="); // оформляем вывод.
  // выводим дистанцию.
  Serial.print(Dist_right);
  Serial.println(" cm."); // оформляем вывод.
  // приостанавливаем программу.
  delay(500);
  neck.write(front_ang);
  // оформляем вывод.
  Serial.println(" ------------------------.");
  // Если расстояние до препятствия слева наибольшее.
  if ((Dist_left > Dist_front) && (Dist_left > Dist_right))
  {
    // оформляем вывод.
Serial.println("(Dist_left>Dist_front)&&(Dist_left>Dist_right)POVOROT LEFT");
    // приостанавливаем программу.
    delay(500);
    left(); // поворачиваем налево 0,5 секунд.
    delay(time_90);
    if (Dist_left > 10)
    {
      // оформляем вывод.
      Serial.println(" Dist_left>10 VPERED 0.5 SEC");
      // приостанавливаем программу.
      delay(500);
      forward(); // едем вперед 0,5 секунды.
      delay(time_10cm);
    }
  }
```

```
  else
  {
    if (Dist_right > Dist_front)
    {
      // оформляем вывод.
      Serial.println(" Dist_right>Dist_front POVOROT RIGHT");
      // приостанавливаем программу.
      delay(500);
      right(); // поворачиваем направо 0,5 секунд.
      delay(time_90);
      if (Dist_right > 10)
      {
        // оформляем вывод.
        Serial.println(" Dist_right>10 VPERED 0.5 SEC");
        // приостанавливаем программу.
        delay(500);
        forward(); // едем вперед 0,5 секунды.
        delay(time_10cm);
      }
    }
    else
    {
      if (Dist_front > 10)
      {
        // оформляем вывод.
        Serial.println(" Dist_front>10 VPERED 0.5 SEC");
        // приостанавливаем программу.
        delay(500);
        forward(); // едем вперед 0,5 секунды.
        delay(time_10cm);
      }
      else
      {
        // оформляем вывод.
        Serial.println(" right **********RAZVOROT**************");
        // приостанавливаем программу.
        delay(500);
        right(); // разворачиваемся.
        delay(time_180);
      }
    }
  }
}
```

Пример вывода сообщений в окно монитора приведен на рис. 11.2.

Здесь наглядно видно, что когда слева 12 см, прямо 22 см, а справа 15 см, то самое большое расстояние впереди (и оно больше 10 см), и, соответственно, выполняется

команда «Ехать вперед». Удостоверьтесь, что при этом моторы действительно вращаются вперед.

Когда же слева 61 см, прямо 14 см, а справа 13 см, то самое большое расстояние слева (и оно больше 10 см), и, соответственно, выполняются команды «Повернуть влево» и «Ехать вперед». Удостоверьтесь, что при этом моторы вращаются согласно командам.

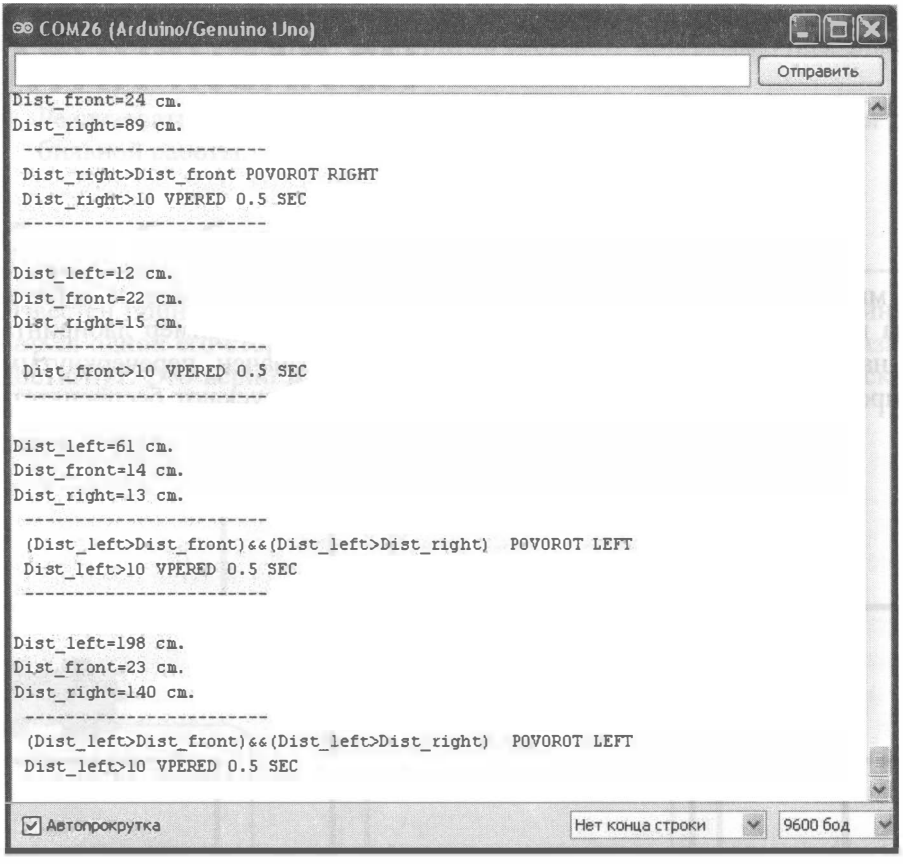


Рис. 11.2. Пример вывода сообщений в окно монитора

Выводы

Нами создана программа проверки и настройки основных функций робота, она же представляет собой простейшую программу, позволяющую роботу двигаться, объезжая препятствия. Введены специальные константы, отвечающие за параметры робота, дан их список и объяснено назначение. Описаны функции движения робота и их назначение. Приведен пример отладки программы.

На следующем этапе нам предстоит модернизация собранного робота под решение конкретных задач.

ГЛАВА 12

РОБОТ, НАХОДЯЩИЙ ВЫХОД ИЗ ЛАБИРИНТА

В этой главе мы поставим себе следующую цель — создать автономного робота, способного за наименьшее время найти выход из лабиринта. Пример лабиринта представлен на рис. 12.1, где выход из лабиринта обозначен перечеркнутым кругом. Все проходы лабиринта проходимы по ширине и допускают беспрепятст-

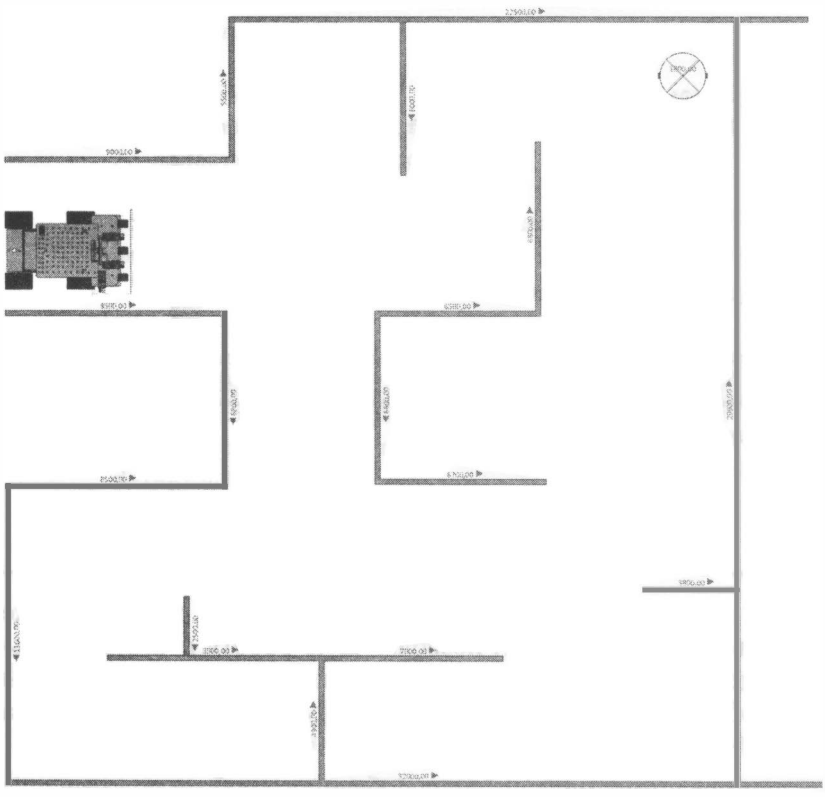


Рис. 12.1. Робот в лабиринте

венную возможность разворота робота. В поисках выхода робот не видит лабиринт целиком и не имеет о нем дополнительной информации.

Для достижения поставленной цели нам потребуется решить следующие задачи:

1. Разобраться, как можно установить наличие препятствий (с помощью каких датчиков и решений).
2. С учетом выбранных датчиков и принятых технических решений создать алгоритм движения, гарантирующий прохождение лабиринта.
3. На основании созданного алгоритма составить программу, позволяющую роботу преодолеть лабиринт.
4. Реализовать разработанного робота, провести его тестирование и добиться стабильной работы.

Способ обхода лабиринта

Известен принцип, согласно которому, если при движении в лабиринте придерживаться одной его стороны (стенки): левой или правой, то выход обязательно будет достигнут. Это верно для лабиринтов с выходом наружу. Графически возможный путь робота показан на рис. 12.2 (вдоль правой стенки лабиринта) и 12.3 (вдоль его левой стенки).

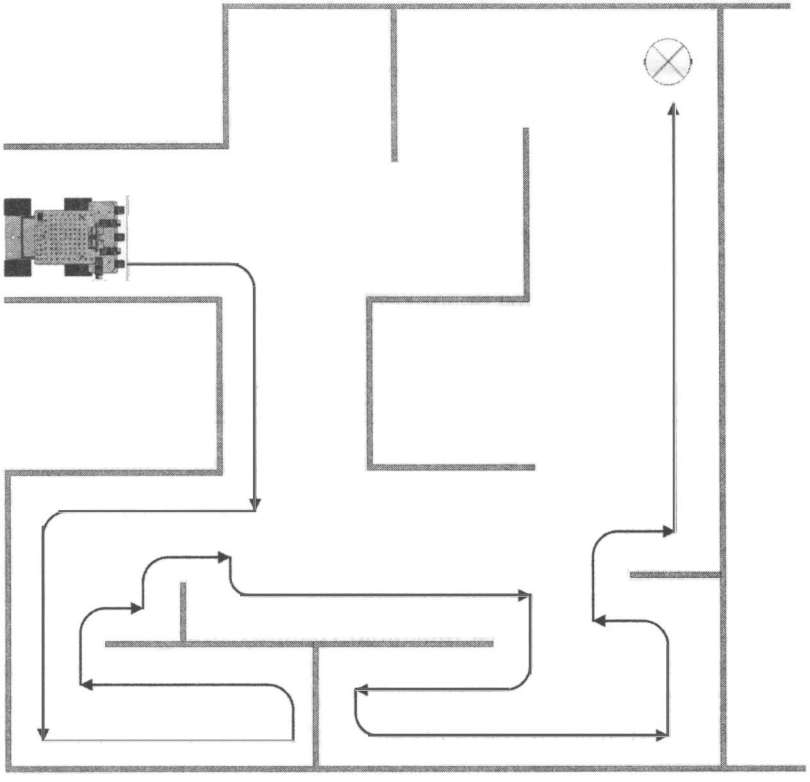


Рис. 12.2. Путь робота вдоль правой стенки

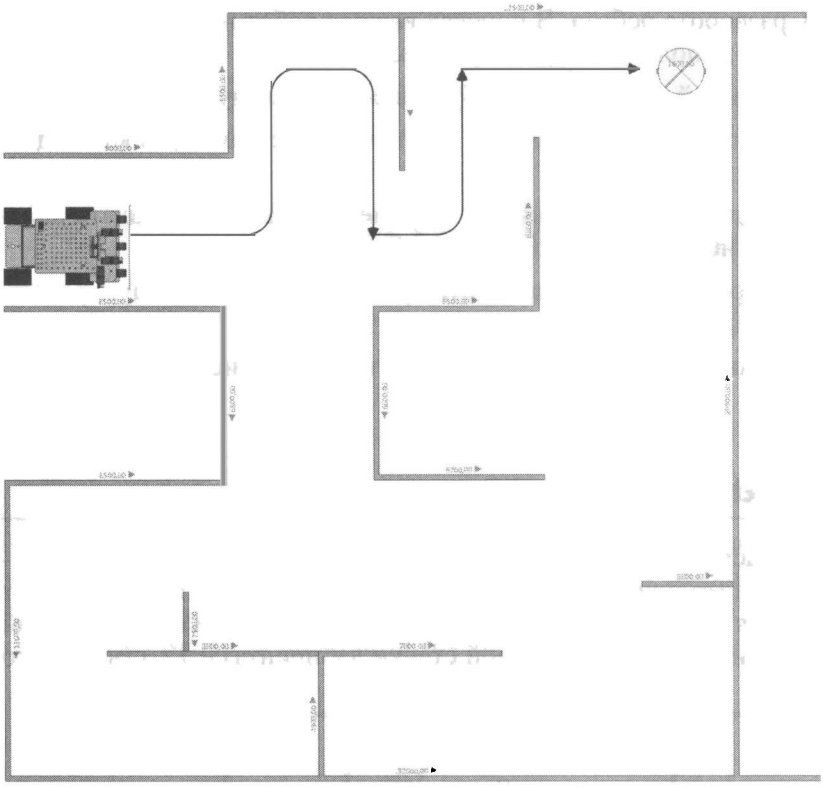


Рис. 12.3. Путь робота вдоль левой стенки

Следует обратить внимание, что по пути, показанному на рис. 12.3, робот достигнет выхода быстрее, но он не может знать заранее, где выход, и какой путь короче. Конечно, это может знать оператор и выбрать для робота выигрышный режим «направо» или «налево» перед входом в лабиринт. Впрочем, как правило, в соревнованиях роботов такая подсказка оператора недопустима.

Обход лабиринта без модернизации робота

Составим алгоритм обхода лабиринта для робота, оснащенного уже знакомым нам ультразвуковым датчиком, установленным на его поворотной голове (рис. 12.4). Пусть робот в этот раз будет совершать обход лабиринта по правой его стороне. При этом он должен всегда держаться на небольшом расстоянии (3–4 см) от правой стенки и адекватно реагировать на изменение обстановки. Добавим к этому расстоянию половину ширины робота и получим нижнюю границу `Dist_Right`. Робот измеряет расстояния с двух сторон: справа и прямо, затем анализирует полученные результаты, выполняет соответствующие движения и снова останавливается для сбора данных. Если расстояние до препятствия справа находится в пределах от 5 до 7 см, значит, подворот не нужен, и он немного проезжает вперед (если при этом нет препятствия спереди).

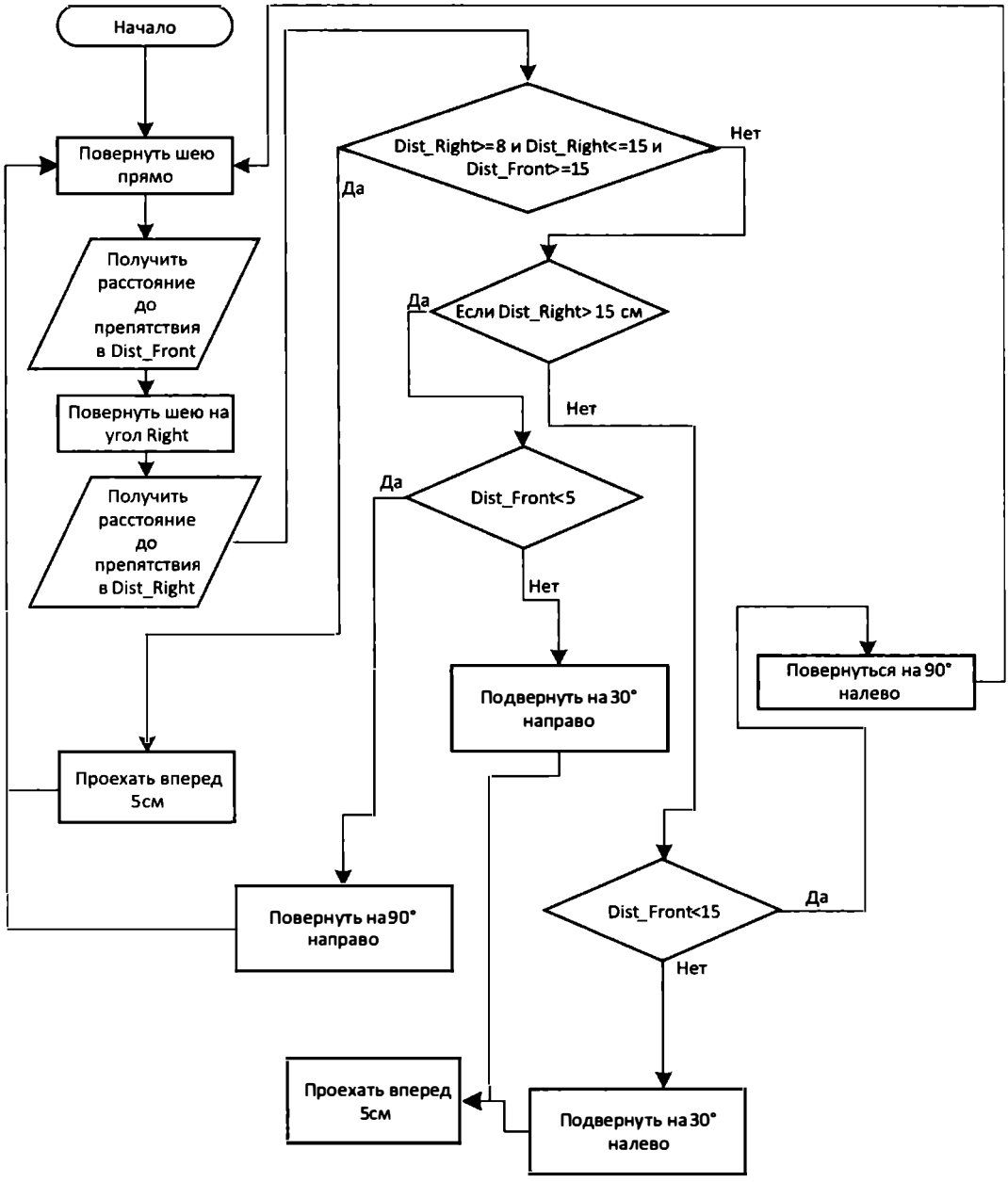


Рис. 12.4. Алгоритм движения вдоль правой стенки с помощью ультразвукового датчика

Если дистанция справа стала больше 7 см, значит, робот стал отходить от правой стены, и нужно начать подворот вправо. Если робот уперся в стену, ему следует повернуть на месте влево и снова получить данные.

Программа

Соответствующая рассмотренному алгоритму программа представлена в листинге 12.1. В ней много настраиваемых значений, которые определяются экспериментально. Это, например, углы поворота головы. Дело в том, что механически установить голову так, чтобы значение 90° соответствовало точно переднему положению головы, довольно трудно, поэтому мы определяем этот угол экспериментально, изменяя значения, передаваемые на сервомотор, и следя за положением головы робота. Временны́е константы, связанные с движением и поворотами, зависят от дорожного покрытия и заряда аккумуляторов. Например, поворот на ковровом покрытии займет больше времени, чем на линолеуме или ламинате, а на гладком и скользком полу возможно инерционное скольжение робота. Следите за зарядом аккумуляторов — если они сильно разрядятся, робот станет передвигаться медленнее, а это приведет к неверным значениям поворотов. В любом случае, значения переменных `left_ang`, `front_ang` и `right_ang`, отвечающих за поворот головы, а также значения `time_90`, `time_180`, `time_10cm` и `time_7cm`, отвечающих за время, требуемое на производство роботом простейших движений, настраиваются самостоятельно.

Листинг 12.1. Программа движения вдоль правой стенки с помощью ультразвукового датчика

```
/// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Подключаем библиотеку, управляющую моторами.
#include "motor.h"
// Подключаем библиотеку, управляющую дальномером.
#include "sonar.h"
// Создаем сервомотор поворота головы.
Servo neck;
// Константы - постоянные значения для уточнения углов.
const int left_ang=168; // Без поправки 160
const int front_ang=98; // Без поправки 90
const int right_ang=28; // Без поправки 20
// Временные константы служат для точного задания времени на поворот, разворот,
движение вперед.
// в миллисекундах.
const int time_90=390;
const int time_180=750;
const int time_10cm=220;
const int time_7cm=120;
void setup()
{
   // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
```

```
  // Инициализируем сервомотор, управление 9-м портом.
  neck.attach(14);
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); // Двигатели остановлены.
}
// Основная программа.
void loop()
{
  // Создаем переменные для хранения двух
  // дистанций до препятствий спереди, справа.
  int Dist_left, Dist_front, Dist_right;
  _stop();
  // Ждем, т. к. поворот занимает небольшое время.
  delay(250);
  // Записываем расстояние до препятствия впереди.
  Dist_front = Sonar(40);
  // Поворачиваем голову направо.
  neck.write(right_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(250);
  // Записываем расстояние до препятствия впереди.
  Dist_right = Sonar(40);
  neck.write(front_ang);
  // Если условия позволяют двигаться прямо
  if((Dist_right>=8)&&(Dist_right<=15)&&(Dist_front>=15))
  {
    forward(); // едем вперед.
    delay(time_10cm);
  }
  else
  {// если появился поворот направо
     if(Dist_right>15)
     {
       // если абсолютно уперлись в стену!
       // узкий правый поворот.
       if(Dist_front<5)
       {
        right(); // поворачиваем направо.
        delay(time_90);
       }
       else // нормальный правый поворот.
       { // поворот направо в движении.
       right();
       delay(time_90/3);
       forward(); // едем вперед 7 см.
```

```
      delay(time_90/3);
      }
    }
    else
    // если уперлись в стену.
    if(Dist_front<15)
    {
      left(); // поворачиваем налево.
      delay(time_90);
    }
    else
   { // подворот налево в движении.
    left();
    delay(time_90/3);
    forward(); // едем вперед 5 см.
    delay(time_90/3);
   }
  }
}
```

Сравнение и выбор датчиков

Как можно видеть, наш робот, определяя расстояния с помощью ультразвукового датчика расстояния HC-SR04, часто останавливается и находит выход из лабиринта относительно медленно. Что ж, можно попытаться ускорить его работу, задействовав дополнительные датчики.

В качестве таких датчиков могут быть использованы кратко рассмотренные в *главе 1* детекторы препятствия (см. рис. 1.4) и инфракрасные датчики расстояния (см. рис. 1.6).

Ультразвуковой датчик HC-SR04

Взглянем еще раз на используемый нами ультразвуковой датчик HC-SR04 (подробно он рассматривался в *главах 10 и 11*). Как там и отмечалось, для управления им должно быть задействовано два порта Arduino. Датчик определяет расстояние от 2 до 400 сантиметров, а функция `Sonar(unsigned long Limit)`, которую мы разработали в *главе 10*, может настраиваться на максимально измеряемое расстояние, которое возвращается в сантиметрах.

Время, которое требуется для измерения, изменяется от 1 до 200 миллисекунд и увеличивается с увеличением измеряемого расстояния. Между последовательными измерениями для подавления эхо-эффекта требуется время в 50 миллисекунд. В итоге минимальное количество замеров в секунду — 4, максимальное — 19. Это не много, особенно с учетом того, что для каждого измерения роботу приходится останавливаться, а для измерения расстояния с разных сторон еще и вращать голо-

вой. Неудивительно, что прохождение лабиринта при этом происходит относительно медленно.

Инфракрасный детектор препятствия

Инфракрасный датчик (детектор) препятствия, показанный на рис. 1.4, определяет наличие препятствия по интенсивности отраженного света: если она выше пороговой, на выходе появляется низкий уровень напряжения, а если препятствия нет, то уровень напряжения на выходе высокий. Это важно не перепутать! Датчик не информирует о расстоянии до препятствия, он только сбрасывается в ноль при его наличии.

Расстояние до препятствия, на котором срабатывает датчик, можно регулировать в диапазоне от 3 до 80 см, вращая бегунок переменного резистора (в центре датчика). Плата снабжена двумя информационными светодиодами: красный — наличие питания, зеленый — наличие препятствия. В передней части платы датчика расположены инфракрасный излучатель и приемник (рис. 12.5).

Недостатком этого детектора является зависимость расстояния срабатывания от цвета объекта. Фактически это говорит нам о том, что датчики нужно подстраивать под цвет проходимого лабиринта. Датчик также может срабатывать на несуществующее препятствие при наличии посторонних источников инфракрасного освещения — к таковым относится прямой или отраженный от близлежащих объектов солнечный свет.

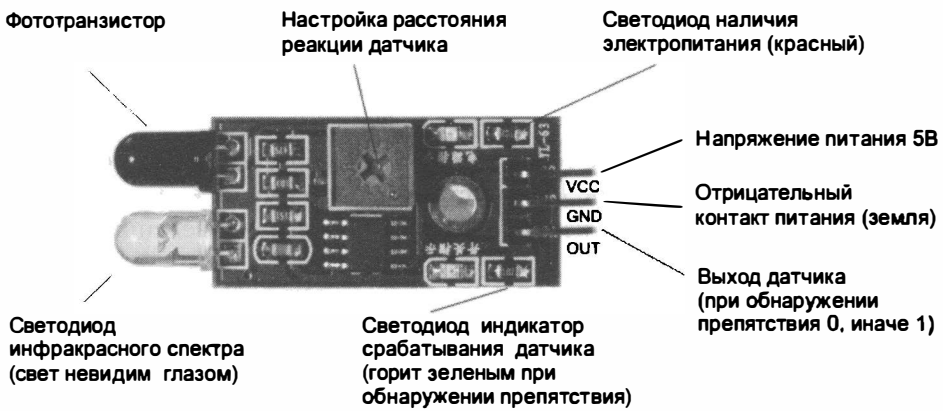


Рис. 12.5. Инфракрасный детектор препятствия

Инфракрасный датчик Sharp GP2Y0A21YK

Инфракрасный датчик расстояния Sharp GP2Y0A21YK, показанный на рис. 1.6, отличается от прочих тем, что выдает данные о расстоянии в виде аналогового сигнала, — его выходное напряжение изменяется в зависимости от расстояния. При этом измеряемое расстояние лежит в диапазоне от 10 до 80 см. Для измерений требуется

подключить датчик к питанию 5 вольт, а его выход — к аналоговому входу Arduino, например, A0. Следует учесть, что зависимость показаний датчика от расстояния нелинейная, поэтому для расчета расстояния в сантиметрах нужно использовать специальную функцию обработки. Пример такой функции приведен в листинге 12.2. Время, которое требуется для измерения, составляет около 0,1 миллисекунды, паузы между замерами не нужны. Максимальное количество замеров в секунду — около 10 000. Основные недостатки таких датчиков заключаются в их относительно высокой стоимости и малом диапазоне измеряемых расстояний.

Листинг 12.2. Функция обработки показаний датчика расстояния Sharp GP2Y0A21YK

```
int IR_Ranging(int nk)
{
  if((nk>13)&&(nk<22))
  {
    int vole=analogRead(nk);
    if(vole<18) vole=18;
    return 5461/(vole - 17) - 2;
  }
  else return 1000;
}
```

Обоснование выбора датчиков препятствия

В связи с тем, что ультразвуковой датчик уже используется в проекте, а инфракрасные детекторы препятствия примерно в 10 раз дешевле и чаще встречаются в продаже, чем инфракрасный датчик расстояния Sharp GP2Y0A21YK, то мы воспользуемся ими.

Впрочем, датчик Sharp GP2Y0A21YK со счетов мы тоже сбрасывать не станем, поскольку он практически не дает сбоев, и, если стоимость робота не критична, рекомендуем использовать именно его, немного модифицировав приведенный далее код.

Модернизация робота

Итак, поскольку наш робот часто останавливается и находит выход из лабиринта относительно медленно, мы попытаемся ускорить его работу, используя дополнительные датчики — детекторы препятствия.

Монтаж детекторов препятствия

Установим на передний бампер робота три детектора (датчика) препятствия, чего для наших целей вполне достаточно (рис. 12.6–12.9). У нашего робота два детектора будут смотреть вперед и один — вправо. Наличие двух детекторов, направлен-

ных вперед, улучшает качество определения препятствий спереди, поскольку сектор определения одного детектора не захватывает всю переднюю зону.

Крепятся датчики на шасси робота при помощи винтов. Перед установкой датчиков на шасси следует подсоединить или припаять к ним провода. Датчики линии, используемые ранее, следует демонтировать, их контакты будут использованы под детекторы препятствия и потребуется провести еще три провода и задействовать один дополнительный порт/пин Arduino.

Желательно также защитить сенсоры от ударов, иначе от столкновений с препятствиями они быстро погнутся и выйдут из строя.

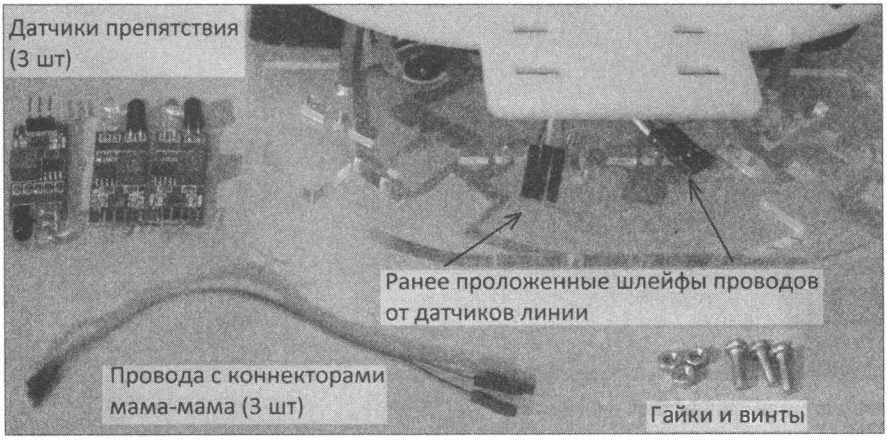


Рис. 12.6. Робот, три датчика препятствия и крепежные винты

Рис. 12.7. Три датчика препятствия размещены на верхней плате шасси

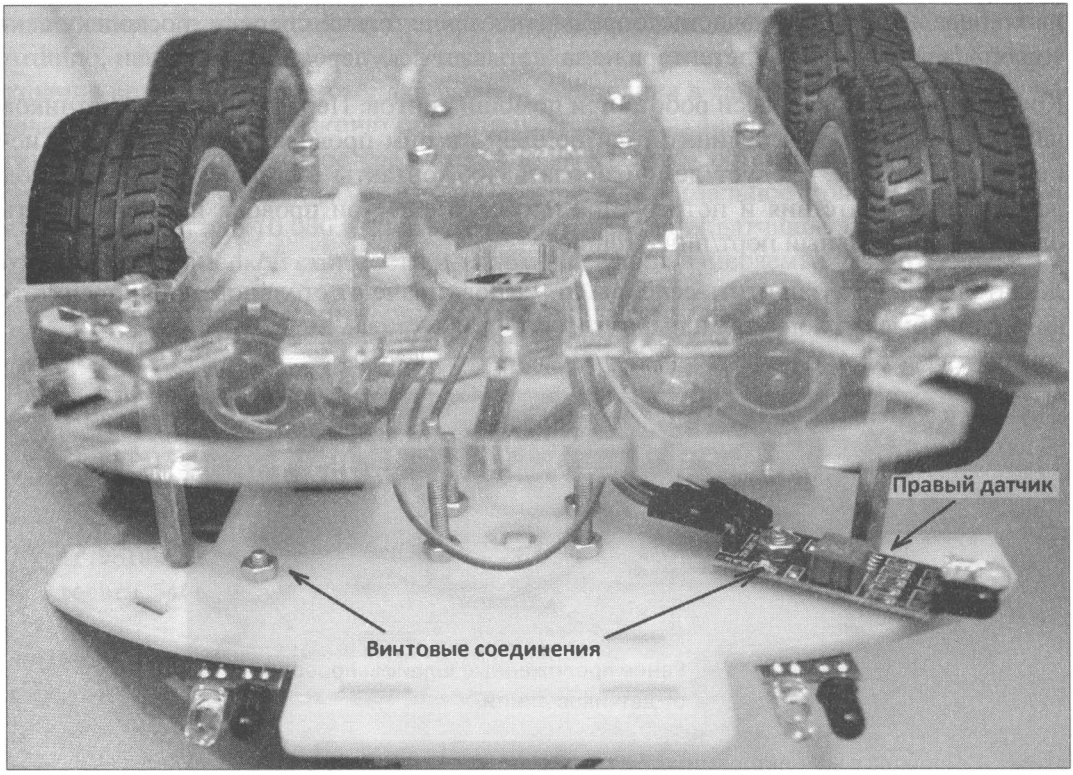


Рис. 12.8. Крепление датчиков препятствия винтами

Рис. 12.9. Пример альтернативной установки датчиков при помощи термоклея

Полную схему робота здесь приводить не имеет смысла — она уже была приведена в предшествующих главах. Изменения коснулись только порядка подключения детекторов. Электропитание +5V они в общем случае получают от модуля драйвера двигателей L298N (рис. 12.10), а для нашего робота — через плату Arduino Sensor Shield v5.0. «Земля» (контакт GND) берется с любого доступного места — например, от того же L298N. Сигнальные выводы подключаются к 7, 8 и 9-му портам Arduino (рис. 12.11). Получать питание 5 В датчики могут также от Arduino, но на

некоторых платах Arduino стоят слабые стабилизаторы напряжения, которые не следует перегружать, поэтому 5 В и подается на датчики от L298N.

Фронтальные датчики мы подключаем к портам 8 (датчик S2) и 9 (датчик S3), а датчик, расположенный с правой стороны, к 7-му порту (датчик S1).

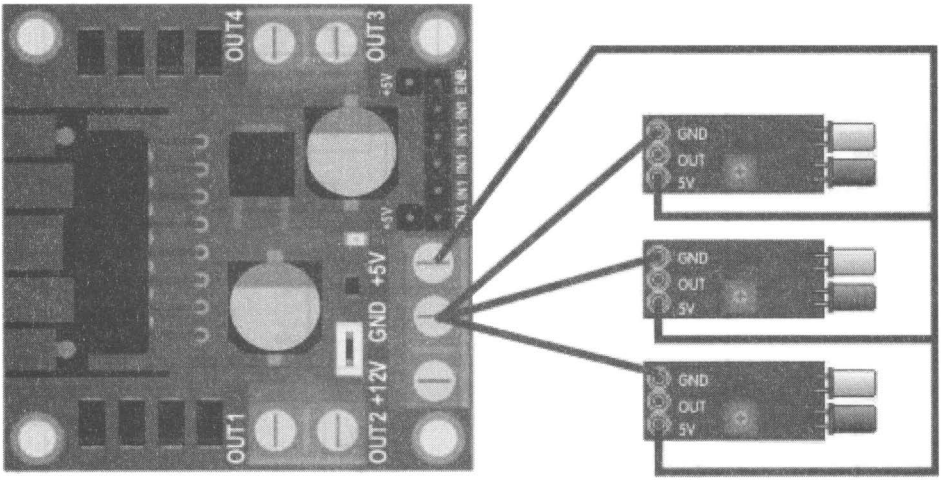


Рис. 12.10. Электропитание датчиков

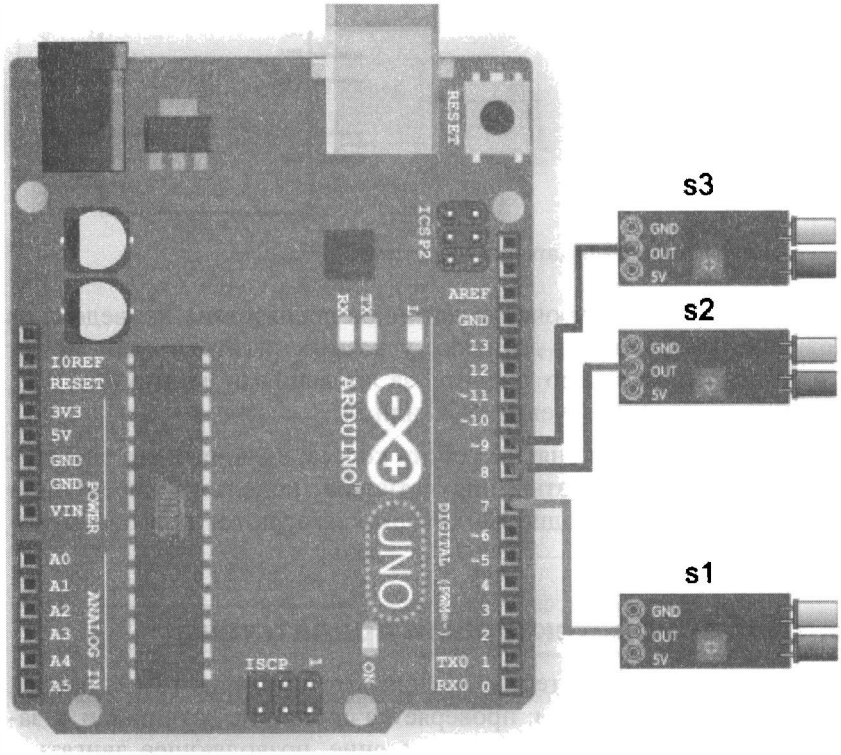


Рис. 12.11. Подключение датчиков к Arduino UNO

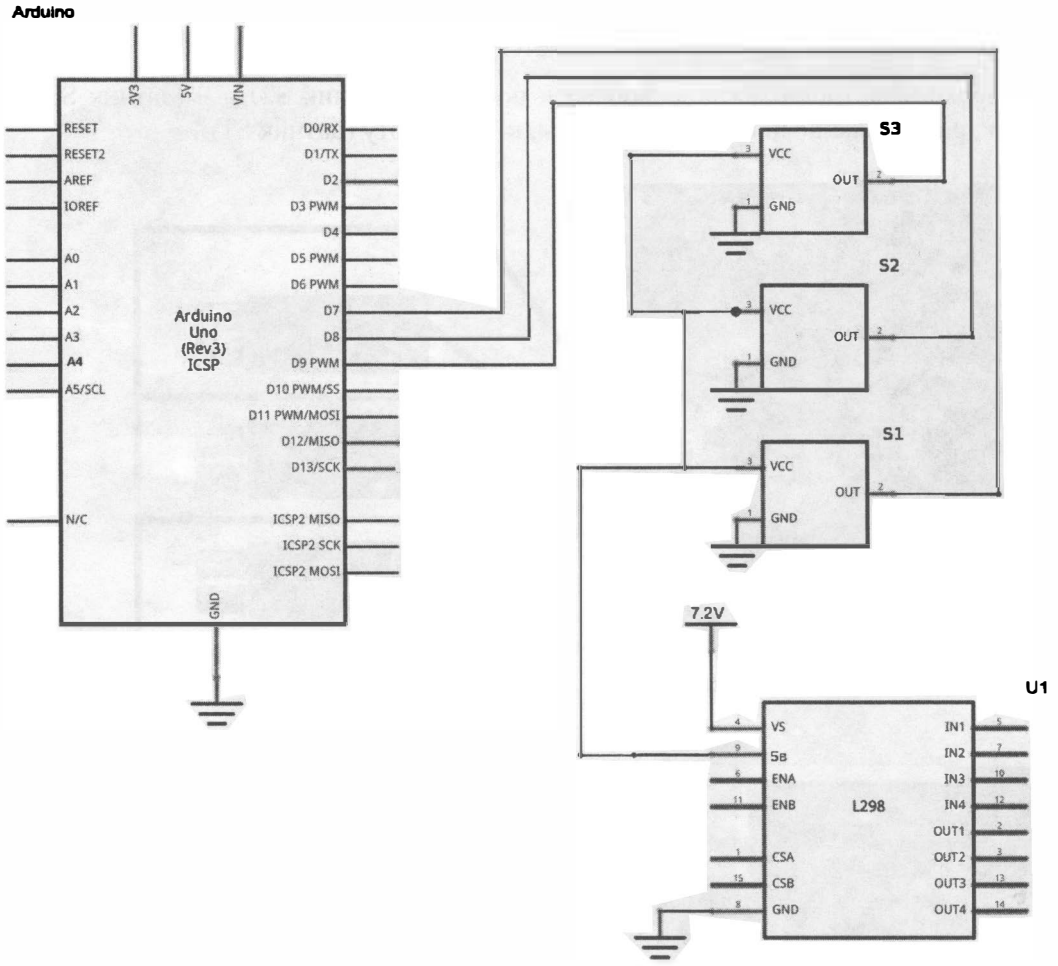


Рис. 12.12. Принципиальная схема подключения датчиков препятствия

Итоговая принципиальная схема подключения датчиков препятствия приведена на рис. 12.12. Расположение контактов на датчиках у разных изготовителей может быть различным, поэтому внимательно следите за названиями контактов. Так, питающий контакт может иметь обозначение VCC или 5V.

После подключения датчиков следует снять с робота колеса, включить его электропитание и настроить расстояние срабатывания датчиков, используя переменные резисторы и зеленые сигнальные светодиоды, которые загораются при обнаружении препятствия.

Программа для робота с детекторами препятствия

Алгоритм движения робота с тремя детекторами препятствия (рис. 12.13) проще, чем алгоритм с ультразвуковым датчиком: проверяется состояние датчиков и, в зависимости от полученных данных, принимается решение, позволяющее двигаться вдоль правой стены.

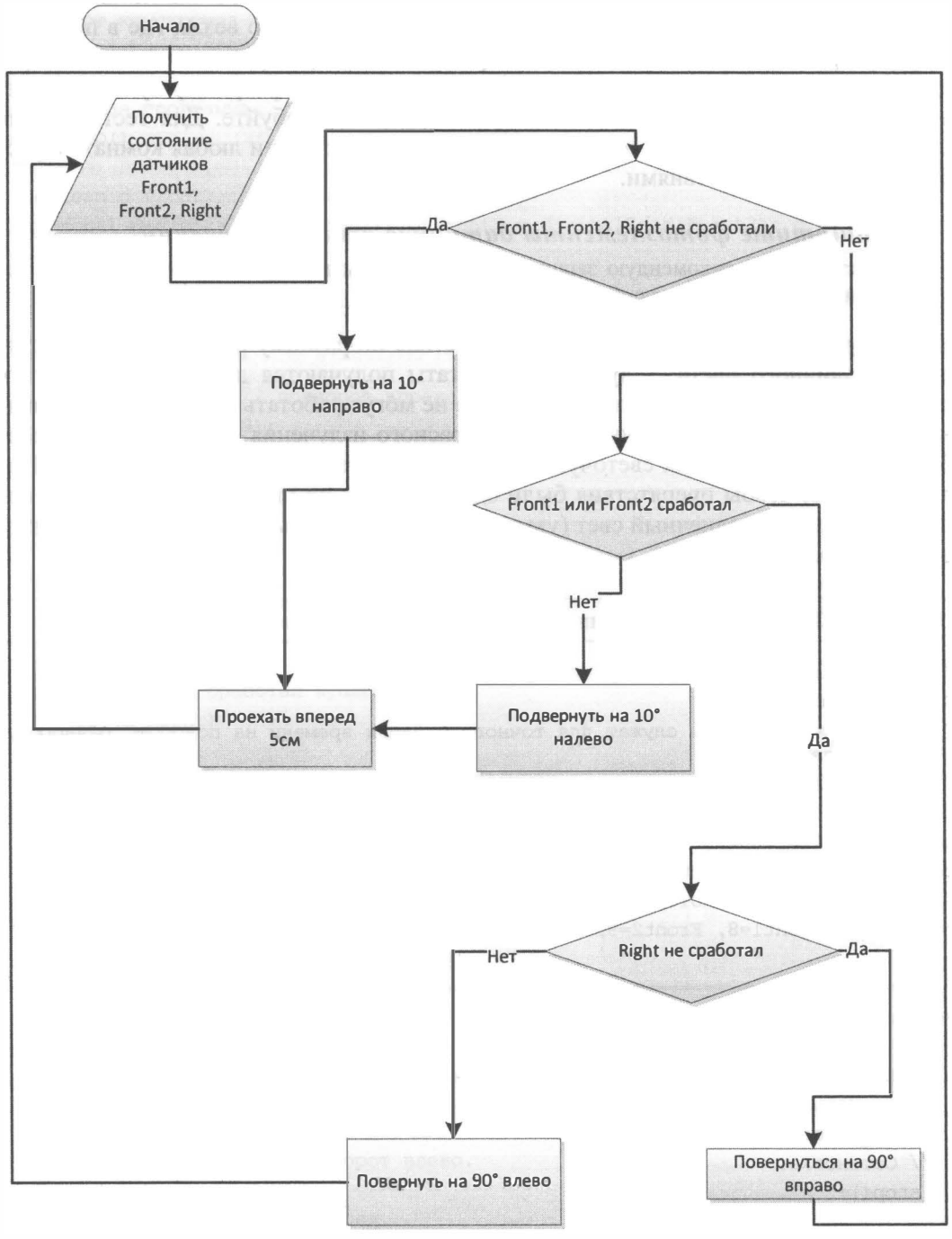


Рис. 12.13. Алгоритм движения робота с тремя детекторами препятствия

Загрузите программу (листинг 12.3). Поэкспериментируйте со значениями переменных времени для подбора точных поворотов. Собственно везде, где в программе стоит функция `delaly()`, создается пауза, в течение которой и выполняется предыдущая команда. Если вам кажется, что можно создать программу лучше, чем та, что приведена в листинге, смело программируйте и пробуйте. Для тестирования программы не обязательно наличие лабиринта — подойдет и любая комната с расставленными препятствиями.

> ***Защитите фотоэлементы датчиков от столкновений!***
>
> Настоятельно рекомендую защитить фотоэлементы датчиков от повреждений в результате столкновения робота с препятствиями.

Используемые в проекте детекторы препятствия по-разному реагируют на препятствия различного цвета — лучшие результаты получаются для белых и светлых препятствий. Детекторы препятствия также не могут работать при наличии солнечного света и источников сильного инфракрасного излучения — это связано с особенностью используемых светочувствительных элементов (фототранзисторов). Поэтому важно, чтобы препятствия были однородными по цвету, и в комнату не попадал бы прямой солнечный свет (увы, приходится считаться с этими недостатками детекторов).

Листинг 12.3. Программа движения робота с тремя детекторами препятствия

```
// Подключаем библиотеку, управляющую моторами.
#include "motor.h"
// Временные константы служат для точного задания времени на поворот, разворот,
движение вперед
// в миллисекундах.
const int time_90=390;
const int time_180=750;
const int time_10cm=220;
// Номера портов, к которым подключены датчики препятствия.
const int  Front1=8, Front2=9, Right=7;

//=======================================
void setup()
{
  // Заносим в переменные номера контактов (пинов) Arduino.
 // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Остановка.
  _stop();

  // Инициализируем порты датчиков препятствия.
  pinMode(Front1, INPUT);
  pinMode(Front2, INPUT);
  pinMode(Right, INPUT);
```

```
  // Устанавливаем скорость передачи данных по кабелю.
  // Порт компьютера
  //Serial.begin(9600);
}
// Основная программа.
void loop()
{
  boolean d_Front1, d_Front2, d_Right;
  d_Front1 = digitalRead(Front1); d_Front2 = digitalRead(Front2); d_Right =
digitalRead(Right);

  // Если ни один датчик не сработал.
  if (d_Front1 && d_Front2 && d_Right)
  {
    forward_right();// подворот вправо.
    delay(time_90 / 9);
    forward(); // едем вперед.
    delay(time_10cm / 2);
  }
  else
  {
    // Если сработал один из передних датчиков и не сработал правый.
    if ((!d_Front1) || (!d_Front2))
    {
      // Если не сработал правый датчик.
      if (d_Right)
      {
        // поворачиваем направо на 90 градусов.
        right();
        delay(time_10cm);
      }
      else
      {
        // поворачиваем налево на 90 градусов.
        left();
        delay(time_10cm);
      }
    }
    else
    { // Если сработал правый датчик.
      forward_left();// подворот влево.
      delay(time_90 / 9);
      forward(); // едем вперед.
      delay(time_10cm / 2);
    }
  }
}
```

Модернизируем программу

Следующий листинг (12.4) демонстрирует модернизированную программу прохода лабиринта с использованием датчиков препятствия. Здесь применены повороты в движении, что позволило максимально упростить программу. Также организовано управление мощностью двигателей (закомментировано в конце функции `avtoroute`). Вся процедура, отвечающая за поведение робота, перенесена из основной программы в функцию `avtoroute()` — это позволит переключать режимы работы робота. Например, если есть команды с пульта, то робот реагирует на них, иначе запускается прохождение лабиринта.

В программе применена стандартная функция `delayMicroseconds()`, являющаяся аналогом функции `delay()`. Отличие этой функции в том, что величина задержки принимается в микросекундах, что позволяет создавать задержки выполнения программы менее 1 миллисекунды (в примере, это 0,5 миллисекунды).

Листинг 12.4. Модернизированная программа прохода лабиринта с использованием датчиков препятствия

```
/ Подключаем библиотеку, управляющую моторами.
#include "motor.h"
// Временные константы служат для точного задания времени на поворот, разворот,
движение вперед
// в миллисекундах.
const int time_90=190;
const int time_10=20;
const int time_180=380;
const int time_10cm=100;
// Номера портов, к которым подключены датчики препятствия.
const int  Front1=8, Front2=9, Right=7;

void avtoroute()
{
  boolean d_Front1, d_Front2, d_Right;
  d_Front1=digitalRead(Front1); d_Front2=digitalRead(Front2);
d_Right=digitalRead(Right);
  // Ищем правую сторону,
  // если ни один датчик не сработал.
  if(d_Front1&&d_Front2&&d_Right)
  {
    forward_right();// подворот вправо.
    delay(3);//time_10cm/5);
    forward(); // едем вперед.
    delayMicroseconds(500);
  }
  else
  {
    // Если сработал один из передних датчиков.
    if(((!d_Front1)||(!d_Front2))&&d_Right)
```

```
    {
      // поворачиваем направо на 90 градусов.
      right();
      //delay(time_10);
      delayMicroseconds(500);
    }
    else
    // Если сработал один из передних датчиков и правый.
    if((((!d_Front1)||(!d_Front2))&&(!d_Right))
    {
      // поворачиваем налево на 90 градусов.
      left();
      //delay(time_10);
      delayMicroseconds(500);
    }
    else
    { // Отклоняемся от правой стороны,
      // если сработал правый датчик.
      forward_left();// подворот влево.
      delay(3);//time_90/5);
      forward(); // едем вперед.
      delayMicroseconds(500);
    }
  }
// Регулировка скорости робота.
//  _stop();
//  delayMicroseconds(100);

}

void setup()
{
  // Заносим в переменные номера контактов (пинов) Arduino.
 // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Остановка.
  _stop();

  // Инициализируем порты датчиков препятствия.
  pinMode(Front1, INPUT);
  pinMode(Front2, INPUT);
  pinMode(Right, INPUT);

  // Устанавливаем скорость передачи данных по кабелю.
  // Порт компьютера
  //Serial.begin(9600);
}
```

```
// Основная программа.
void loop()
{
    avtoroute();
}
```

Сервомоторы постоянного вращения

В лабиринтах хорошо показывают себя роботы, собранные на сервомоторах постоянного вращения, — они практически не ошибаются в любом лабиринте. Это достигается за счет того, что скорость вращения сервомотора изменяется очень точно и плавно, вследствие чего не нужно использовать вставки с вызовом функции `delay()`. Также для повышения точности можно использовать датчики расстояния Sharp. Построение такого робота выходит за рамки этой книги, но вы можете заняться подобным усовершенствованием самостоятельно.

Выводы

В этой главе рассмотрены различные датчики препятствий и выбраны ультразвуковой и инфракрасный. Обсуждены алгоритмы их работы. Составлены соответствующие программы. Показаны способы крепления датчиков препятствий на шасси робота. Приведена принципиальная схема подключения детекторов. Даны рекомендации по подбору параметров, от которых зависит поведение робота в лабиринте, а также советы по самостоятельной доработке конструкции.

Пришло время научить нашего робота использовать электронный компас — и двигаться в заданном направлении с его помощью.

ГЛАВА 13

РОБОТ, ДЕРЖАЩИЙ НАПРАВЛЕНИЕ ПО ЭЛЕКТРОННОМУ КОМПАСУ

Цель, которую мы ставим себе в этой главе, — научиться применять электронный компас в собственных проектах.

Для достижения поставленной цели нам потребуется решить следующие задачи:

1. Понять, как работает электронный компас.
2. Определиться с выбором электронного компаса.
3. Практически применить электронный компас для нахождения северного магнитного полюса.
4. Реализовать робота, который способен держать направление движения, используя данные электронного компаса.

О компасе подробнее

Компас — это магнитный прибор. Традиционный компас представляет собой намагниченную стрелку, посаженную на ось вращения с малым трением. Магнитная стрелка, свободно вращаясь, одним концом поворачивается к северному магнитному полюсу. При этом сам компас нужно держать в руке так, чтобы магнитной стрелке ничего не мешало вращаться. Перестав колебаться, стрелка укажет направление на северный магнитный полюс.

> ***О несовпадении магнитных и географических полюсов***
>
> Магнитный и географический северные полюса не совпадают. В настоящее время северный магнитный полюс находится в районе канадской Арктики и ежегодно смещается примерно на 10 километров.

Электронный компас

Подобным образом работает и электронный компас, почти так, но не совсем. Электронный компас состоит из магниторезисторов, изменяющих свое сопротивление под воздействием магнитной силы, или элементов Холла, построенных на основе кремниевых пластин, чувствительных к изменению положения объекта относительно магнитного поля Земли.

Для использования в самодельных роботах лучше всего подойдет компас, микросхема которого уже распаяна на плате. Подобных решений много, самые распространенные строятся на основе микросхемы HMC5883L. Эти решения недорогие и, как правило, не совсем точные. Более дорогие решения на основе микросхем HMC5983 или AK8975 имеют автоматическую калибровку и температурную компенсацию. Впрочем, для непромышленного, в том числе и нашего, применения вполне достаточно и HMC5883L.

> ***Описание микросхемы HMC5883L***
>
> Полное описание микросхемы HMC5883L от производителя можно найти по адресу: **http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf**.

Подключение

Плата HMC5883L (рис. 13.1) содержит стабилизатор питания на 3,3 В и подтягивающие резисторы, что позволяет подключать ее напрямую к плате Arduino и к питанию 5 В или 3,3 В. Обмен данными между HMC5883L и Arduino осуществляется по шине I^2C, которая представляет собой последовательный интерфейс. К шине I^2C могут быть подключены одновременно несколько устройств, одно из которых должно являться ведущим, а остальные — ведомыми. В нашем случае ведущим

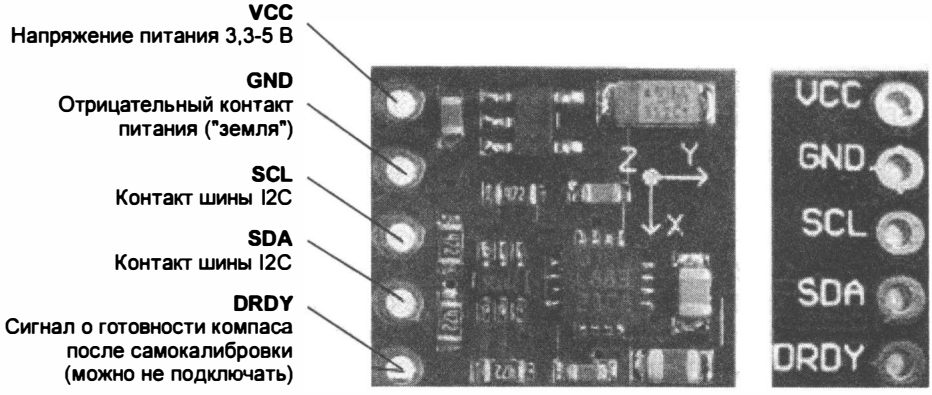


Рис. 13.1. Электронный компас HMC5883L на плате

будет контроллер Arduino, а модуль HMC5883L — ведомым. Каждое устройство на шине I^2C имеет уникальный адрес. Когда ведущий начинает передачу, он передает по шине адрес устройства, к которому выполняется обращение. Устройства, подключенные к шине, проверяют этот адрес и в случае совпадения начинают обмен информацией.

Шина I^2C имеет 2 контакта: SCL — вывод тактового сигнала, его задает ведущий, и SDA — передача/прием данных в обоих направлениях. Для работы по шине I^2C потребуется соединить контакт SCL компаса с контактом SCL Arduino (порт A5) и контакт SDA компаса с контактом SDA Arduino (порт A4), как показано на рис. 13.2. Кроме обязательных контактов, плата HMC5883L снабжена пятым, необязательным, контактом DRDY, на котором формируется положительный сигнал, когда компас прошел самокалибровку и готов к работе. В нашем роботе мы его использовать не будем, и чтобы уменьшить риск перепутать контакты, впаивать в него разъем не следует.

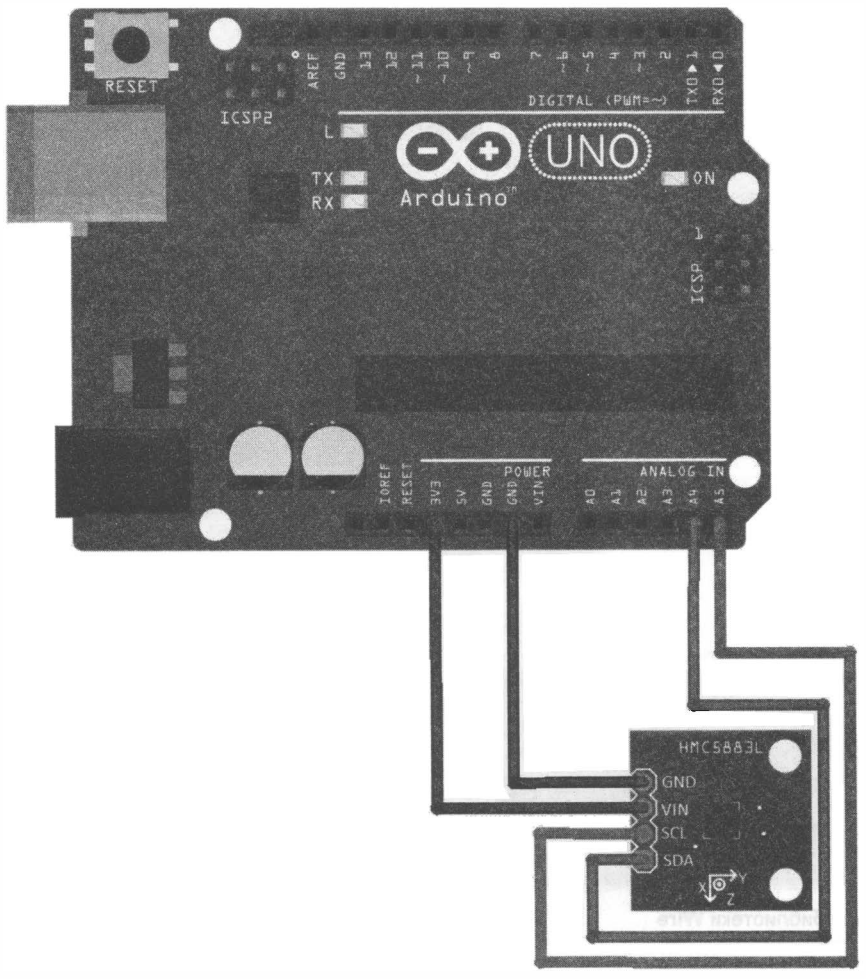


Рис. 13.2. Порядок подключения платы HMC5883L к Arduino UNO

Способы крепления проводов к плате компаса

Как правило, подобные платы поставляются в комплекте с разъемами, к которым следует припаять провода, если вы собираетесь пользоваться разъемными соединениями. Если же вам по душе неразъемное соединение пайкой, то провода припаивать надо непосредственно к контактам платы.

Организация обмена данными

Физического подключения платы HMC5883L к Arduino UNO недостаточно, следует активировать библиотеку `Wire`, отвечающую за обмен данными между устройствами по протоколу I^2C. Библиотека `Wire` входит в стандартный пакет библиотек Arduino IDE и может быть подключена непосредственно после установки среды программирования (рис. 13.3).

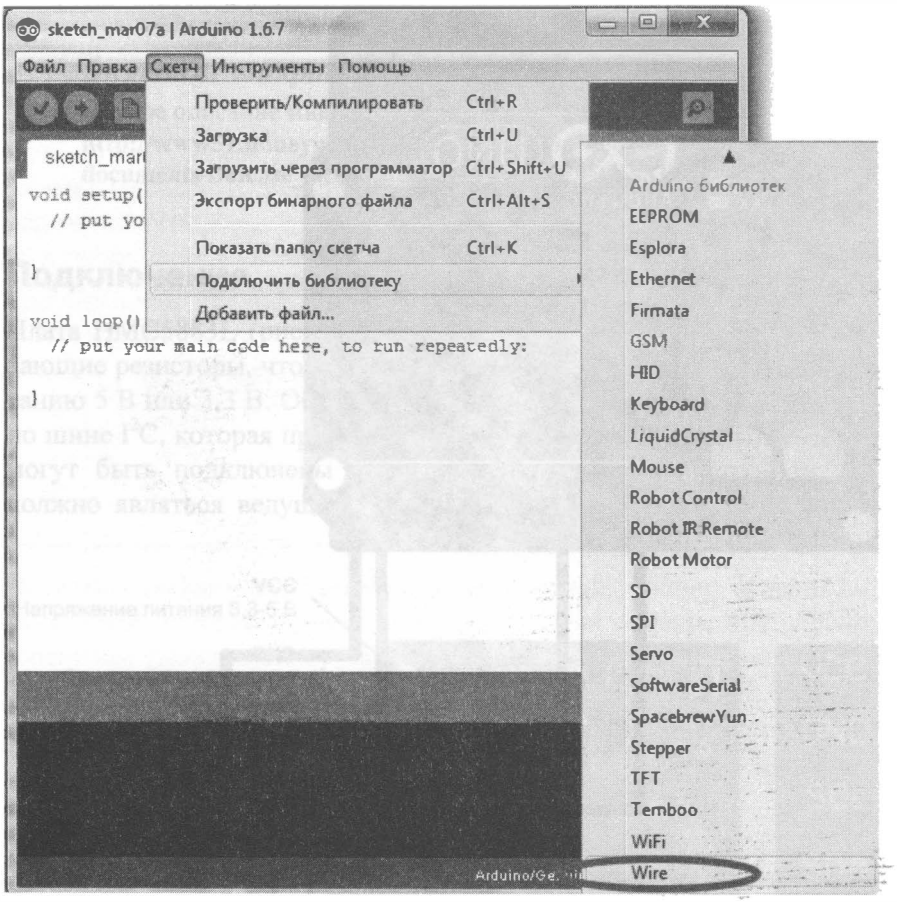


Рис. 13.3. Подключение библиотеки **Wire**

Модернизация робота

Схема подключения модуля магнитометра (компаса) HMC5883L к роботу приведена на рис. 13.2. Согласно этой схеме, питание HMC5883L берет от контакта 3,3 В Arduino, но это не обязательно — можно запитать модуль и от 5 В.

При использовании платы Arduino Sensor Shield v5.0 потребуются четыре провода с клеммами «мама-мама» (рис. 13.4). На Arduino Sensor Shield v5.0 можно будет задействовать либо контакты A4, A5 и питание от соответствующих контактов V (+5 В) и G («земля»), либо контактную площадку IIC (рис. 13.5).

В схеме подключения моторов обязательны керамические конденсаторы емкостью 100 нФ — они должны быть установлены на клеммах всех четырех моторов.

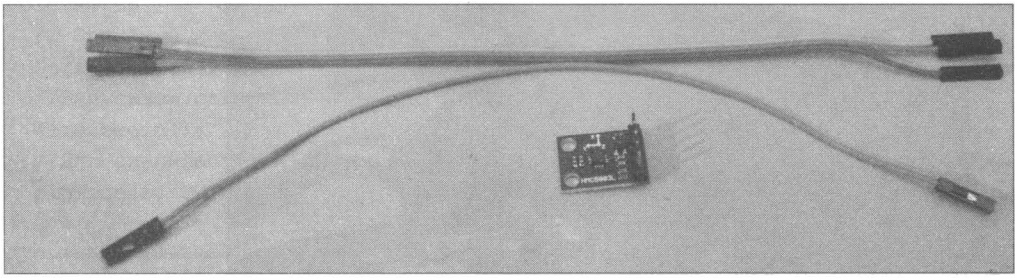
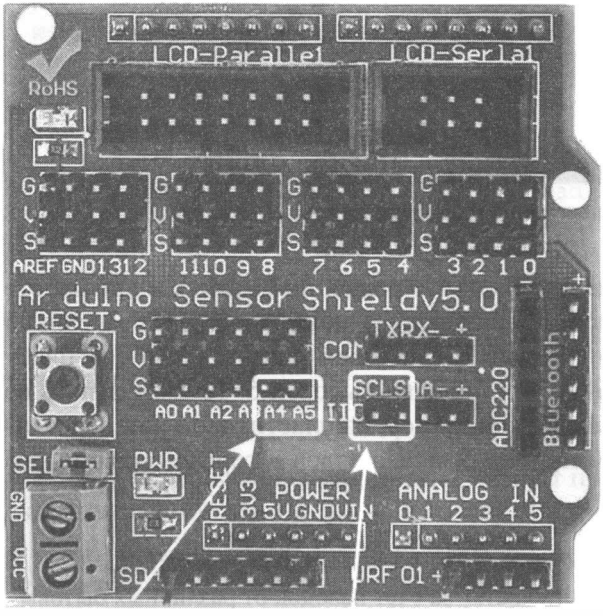


Рис. 13.4. Провода для установки компаса



Компас можно подключить к A4 (SDA) и A5 (SCL)
или к разъему IIC на соответствующие клеммы

Рис. 13.5. Порядок подключения магнитометра с I^2C-интерфейсом к Arduino UNO посредством платы Arduino Sensor Shield v5.0

Желательно установить подобный конденсатор и в цепь питания сервомотора (если это модели sg90 или mg90s) как можно ближе к самому мотору, а лучше всего разобрать сервомотор поворота головы робота и припаять подобный конденсатор прямо на двигатель постоянного тока, который расположен внутри сервомотора (рис. 13.6), — электронный компас будет избавлен от большей части электромагнитных помех, которые будут «сводить его с ума» в противном случае.

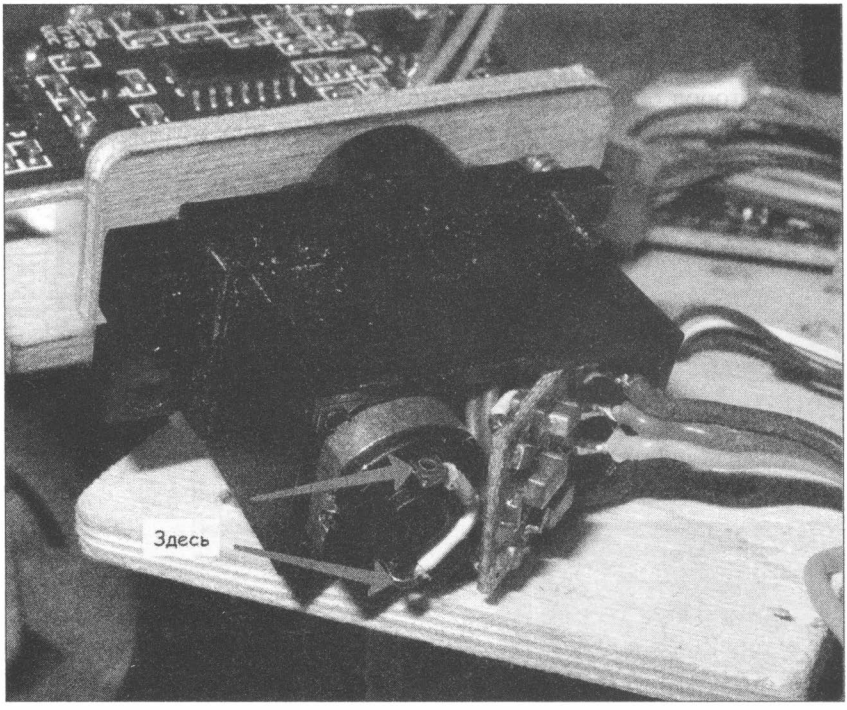


Рис. 13.6. Место установки керамического конденсатора в сервомоторе

Если же проведенные мероприятия не помогли (что редко, но бывает), и при работе двигателей робота (колеса сняты, робот неподвижен) показания прибора сильно колеблются (более 3-х градусов), то с целью дальнейшей защиты от помех плату HMC5883L следует поместить на мачту или полочку высотой 5–10 см. Сделать ее можно из любых подручных материалов — например, из пластиковой или деревянной линейки. Важно, чтобы при этом прибор был закреплен в горизонтальном положении.

Проблемы могут не исчезнуть...

И даже после этого HMC5883L может «врать» за счет наводок от различных бытовых и промышленных приборов, в которых работают мощные электрические двигатели и генерируется электромагнитное поле. Впрочем, и классический магнитный компас в этих ситуациях будет ошибаться.

Получение данных от HMC5883L

Программа приема данных от HMC5883L (листинг 13.1) считывает данные напряженности магнитного поля по различным осям прибора и выводит полученную информацию в порт компьютера.

Листинг 13.1. Прием данных от HMC5883L и передача их на ПК

```
//Подключаем библиотеку I2C.
#include <Wire.h>
////////////////////////////////////////////////////////////
// адрес i2c компаса.
#define address 0x1E
void setup() {
  // Серийный порт для связи с ПК.
  Serial.begin(9600);
  //Запускаем связь по шине I2C.
  Wire.begin();
  //Для анализа окончания процесса будем использовать светодиод.
  pinMode(6, OUTPUT);
}
//====================================
void loop() {
  int x, y, z; //triple axis data
  Wire.beginTransmission(address);
  Wire.write(0x03); //select register 3, X MSB register
  Wire.endTransmission();
  Wire.requestFrom(address, 6);
  if (6 <= Wire.available()) {
    x = Wire.read() << 8;    x |= Wire.read();
    z = Wire.read() << 8;    z |= Wire.read();
    y = Wire.read() << 8;    y |= Wire.read();
  }
  Serial.print("x: ");  Serial.print(x);
  Serial.print("  y: ");  Serial.print(y);
  Serial.print("  z: ");  Serial.println(z);
  delay(500);
}
```

На рис. 13.7 приведены результаты работы программы при испытаниях типичного экземпляра компаса. Испытания показали:

- при вращении прибора показания по оси Y никогда не становились меньше нуля;
- показания по оси X принимали как положительные, так и отрицательные значения;
- показания по оси Z принимали как положительные, так и отрицательные значения.

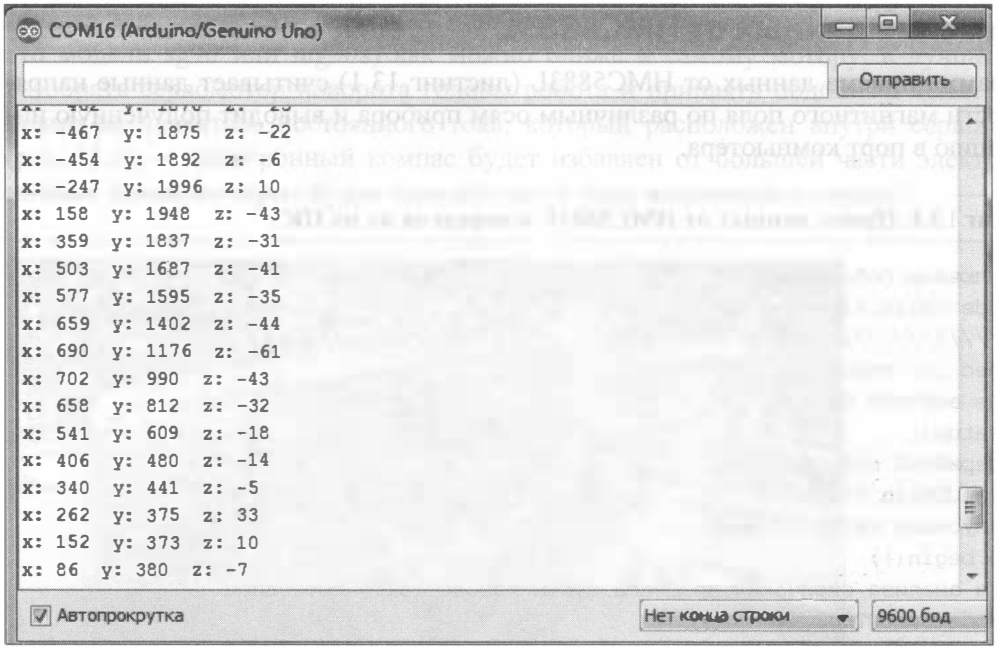


Рис. 13.7. Данные, получаемые от электронного компаса

Следовательно, прибор, как минимум, нуждается в калибровке, в противном случае невозможно будет определить величину отклонения от «севера» даже с некоторой погрешностью.

В листинге 13.2 приведена программа калибровки, которая в течение некоторого времени снимает показания с прибора и запоминает экстремальные значения. В это время робот будет вращаться и искать магнитные экстремумы по осям X и Y. Предварительно робота нужно установить на ровную горизонтальную поверхность.

Робот должен найти четыре экстремума, где напряженность магнитного поля по осям X и Y прибора максимальна или минимальна. Эти значения записываются в энергонезависимую память контроллера Arduino и затем могут быть использованы в программе для корректировки расчета угла отклонения от «севера». Для индикации работы программы удобно использовать сигнальный светодиод, предварительно установленный в 6-й порт. Светодиод горит, пока снимаются данные, и гаснет, когда расчет завершится.

Значения, записанные в энергонезависимую память, не стираются после отключения питания и будут храниться там до тех пор, пока вы их не перезапишете (используются ячейки с 0 по 19).

Листинг 13.2. Основная программа калибровки компаса

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
// Подключаем функции управления моторами
#include "motors.h"
```

```
//Работа с компасом
#include "compas.h"
//======================================
void setup() {
  // Заносим в переменные номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Двигатели остановлены.
  _stop();
  delay(10);

  Serial.begin(9600);
   //Запуск обмена по шине I2C
    Wire.begin();
  //Настройка регистров компаса.
  compas_setup_a();
  delay(100);
  pinMode(6, OUTPUT);
  //Включение светодиода.
  digitalWrite(6, HIGH);
//Запуск поиска экстремумов роботом
  compas_extremum_to_eeprom();
  //Выключение светодиода.
  digitalWrite(6, 0);
  _stop();
  delay(10);
}
//=========================================
void loop()
{
}
```

Для работы программы калибровки подготовлен файл compas.h. Его текст в тексте книги не приводится, т. к. он получился довольно громоздким. Сам же этот файл вы найдете в электронном архиве, сопровождающем книгу (см. *приложение 2*). Далее описаны основные функции, входящие в файл compas.h:

- файл compas.h содержит переменные: `dX`, `dY`, `dZ`, `scaleX` и `scaleY`, которые используются в расчетах калибровочных констант;
- функция `write_int_to_eeprom()` записывает в энергонезависимую память два байта: первый параметр — адрес, второй — значение;
- функция `read_int_from_eeprom()` читает из энергонезависимой памяти два байта, ее единственный параметр — это адрес ячейки памяти;
- функция `compas_max_min_to_eeprom()` ищет магнитные экстремумы, а затем выбранные экстремальные значения запоминает в энергонезависимой памяти контроллера Arduino, при этом она заставляет робота вращаться;
- функция `compas_setup_a()` готовит компас к работе;

- функция `compas_setup()` дополнительно обрабатывает информацию из энергонезависимой памяти с калибровочными параметрами;
- функция `get_Compass_ang()` возвращает угол в градусах от –180 до 180, используя результаты калибровки;
- функция `compas_read(int &x, int &z, int &y)` записывает в глобальные переменные `x`, `y` и `z` значения напряженностей по осям, считанные из электронного компаса.

В листинге 13.3 приведена программа, демонстрирующая расчет угла отклонения от магнитного «севера».

Листинг 13.3. Программа, демонстрирующая расчет угла отклонения от магнитного «севера»

```
//Подключаем библиотеку I2C.
#include <Wire.h>
////////////////////////////////////////////////////////////
//Подключаем функции управления моторами
#include "motors.h"
//Подключаем функции HMC5883L
#include "compas.h"
//=============================================
void setup() {
  //Серийный порт для связи с ПК.
  Serial.begin(9600);
  //Запускаем связь по шине I2C.
  Wire.begin();
  //Компас
  compas_setup();       //инициализация HMC5883L
  //Для анализа окончания процесса будем использовать светодиод.
  pinMode(6, OUTPUT);
}
//=============================================
void loop() {
  float heading = get_Compass_ang();
  Serial.println(heading);
  digitalWrite(6, HIGH);
  delay(500);
  digitalWrite(6, 0);
}
```

На рис. 13.8 приведены результаты работы программы при изменении угла от –180° до 180°.

На практике удобно сориентировать робота в направлении на север и отслеживать отклонения от нуля в сторону положительных и отрицательных показаний.

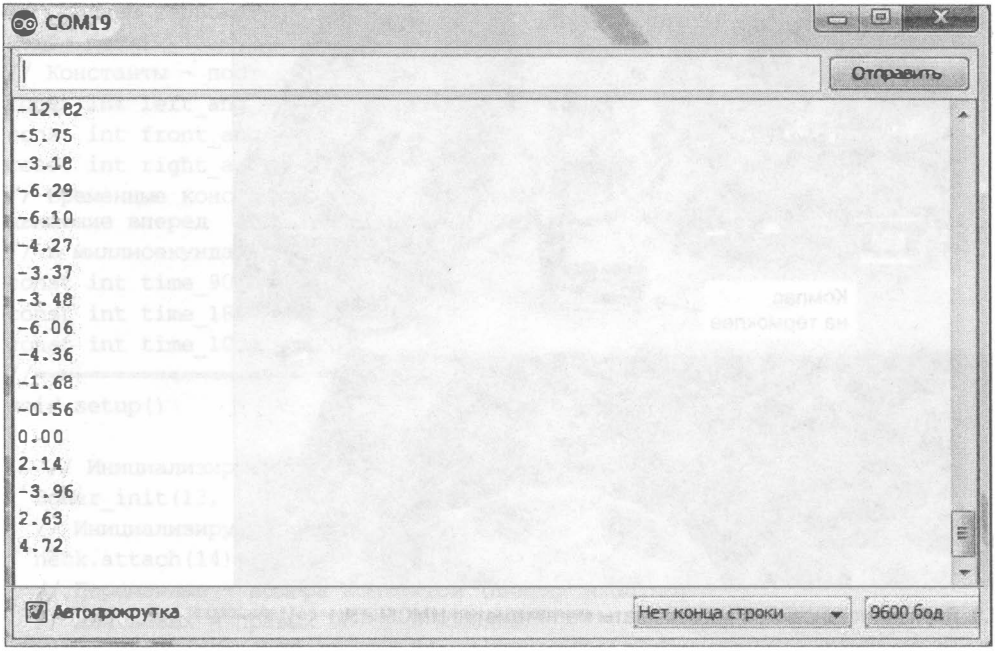


Рис. 13.8. Работа программы при настройке углов от –180° до 180°

Правильная установка магнитометра

При установке на робота компас (магнитометр) должен быть точно ориентирован. Для правильной его ориентации нужно знать точное направление на северный магнитный полюс. Получить его можно с помощью обычного компаса, или электронного компаса в смартфоне, или от самого HMC5883L.

Установка модуля компаса на робота осуществляется в несколько этапов:

1. Запустить на выполнение программу (см. листинг 13.1) и открыть порт для просмотра результатов работы магнитометра.
2. Поместить магнитометр на место предполагаемой установки на роботе.
3. Сориентировать робота по направлению на магнитный «север».
4. Вращением магнитометра добиться положения, близкого к нулю.
5. Закрепить магнитометр на роботе так, чтобы исключить возможность его смещения (рис. 13.9).

Теперь робот будет четко ориентироваться по сторонам света.

> ***Не крепите магнитометр металлическими винтами!***
>
> Несмотря на наличие на плате компаса технологических отверстий для винтового крепления, пользоваться винтами не следует, т. к. металлические винты внесут дополнительные помехи в работу прибора. Закреплять HMC5883L следует при помощи клея или двустороннего скотча.

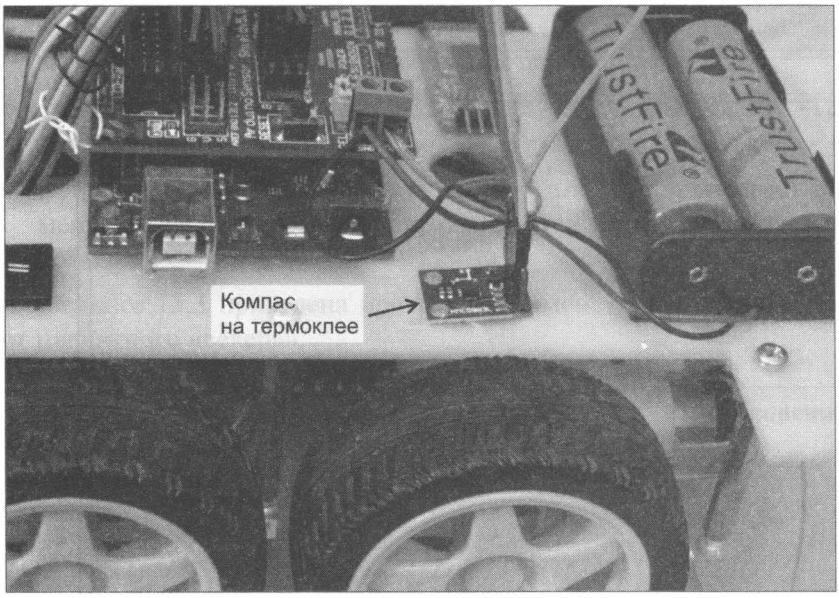


Рис. 13.9. Вариант установки на робота платы магнитометра HMC5883L

Программа

Теперь можно приступить к разработке программы движения робота. На рис. 13.10 приведен алгоритм, отвечающий за движение робота в направлении на север, — робот ищет северный магнитный полюс и разворачивается к нему.

В листинге 13.4 приведена программа, реализующая этот алгоритм. В ней дополнительно предусмотрено, что робот должен объезжать препятствия: робот движется на север, пока нет препятствий, а при появлении препятствия пытается его объехать и вернуться к намеченному маршруту.

> ***Откалибруйте компас робота!***
>
> Перед запуском этой программы компас робота должен быть обязательно откалиброван (см. листинг 13.2).

Листинг 13.4. Программа движения робота в северном направлении

```
#include "motor.h"
#include <Wire.h>
//////////////////////////////////////////////////////////
// Подключаем функции HMC5883L
#include "compas.h"
// Подключаем библиотеку для создания дополнительных Serial-портов.
#include <SoftwareSerial.h>
// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Подключаем библиотеку, управляющую дальномером.
#include "sonar.h"
```

```
// Создаем сервомотор поворота головы.
Servo neck;
// Константы - постоянные значения для уточнения углов.
const int left_ang = 168;
const int front_ang = 98;
const int right_ang = 28;
// Временные константы служат для точного задания времени на поворот, разворот,
движение вперед
// в миллисекундах.
const int time_90 = 390;
const int time_180 = 750;
const int time_10cm = 220;
//=============================================
void setup()
{
   // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
  // Инициализируем сервомотор, управление 9-м портом.
  neck.attach(14);
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
  Wire.begin();
  //Компас
  compas_setup();       // инициализация HMC5883L
  // Устанавливаем скорость передачи данных по кабелю.
  // Порт компьютера
  //Serial.begin(9600);
}
// Основная программа.
void loop()
{
  // Создаем переменные для хранения трех дистанций - слева, впереди, справа.
  int Dist_left, Dist_front, Dist_right;
  float Compass_ang;
   _stop();
   // Перед получением данных от компаса останавливаем моторы.
   delay(150);
  Compass_ang = get_Compass_ang();
  while ((Compass_ang > 5) && (Compass_ang < 355))
  {
    if (Compass_ang <= 180)
    {
      left(); //Влево.
    }
    else
    {
      right();//Вправо.
    }
```

```
  delay(50);
  _stop();
  // Перед получением данных от компаса останавливаем моторы.
  delay(150);
  Compass_ang = get_Compass_ang();
}
// Анализ обстановки на наличие препятствий.
// Поворот головы налево.
neck.write(left_ang);
// Ждет, т. к. поворот занимает небольшое время.
delay(150);
// Записывает расстояние до препятствия слева.
Dist_left = Sonar(40);
// Поворачивает голову прямо вперед.
neck.write(front_ang);
// Ждет, т. к. поворот занимает небольшое время.
delay(150);
// Записывает расстояние до препятствия впереди.
Dist_front = Sonar(40);
// Поворачивает голову направо.
neck.write(right_ang);
// Ждет, т. к. поворот занимает небольшое время.
delay(150);
// Записывает расстояние до препятствия справа.
Dist_right = Sonar(40);
neck.write(left_ang);
if (Dist_front > 20) {
  forward(); // едем вперед.
}
else if ((Dist_left > 20) || (Dist_right > 20))
{
  (Dist_left > Dist_right)
  {
    left(); // поворачиваем налево 0,5 секунд.
    delay(time_90);
  }
  else
  {
    right(); // поворачиваем направо 0,5 секунд.
    delay(time_90);
    forward(); // едем вперед 0,5 секунды.
    delay(time_10cm * 2);
  }
  else // Заехал в тупик.
  {
    backward();
    delay(time_10cm);
    _stop();
    neck.write(left_ang);
```

```
      // Ждет, т. к. поворот занимает небольшое время.
      delay(150);
      // Записывает расстояние до препятствия слева.
      Dist_left = Sonar(40);
      // Поворачивает голову направо.
      neck.write(right_ang);
      // Ждет, т. к. поворот занимает небольшое время.
      delay(200);
      // Записывает расстояние до препятствия справа.
      Dist_right = Sonar(40);
    }
  }
}
```

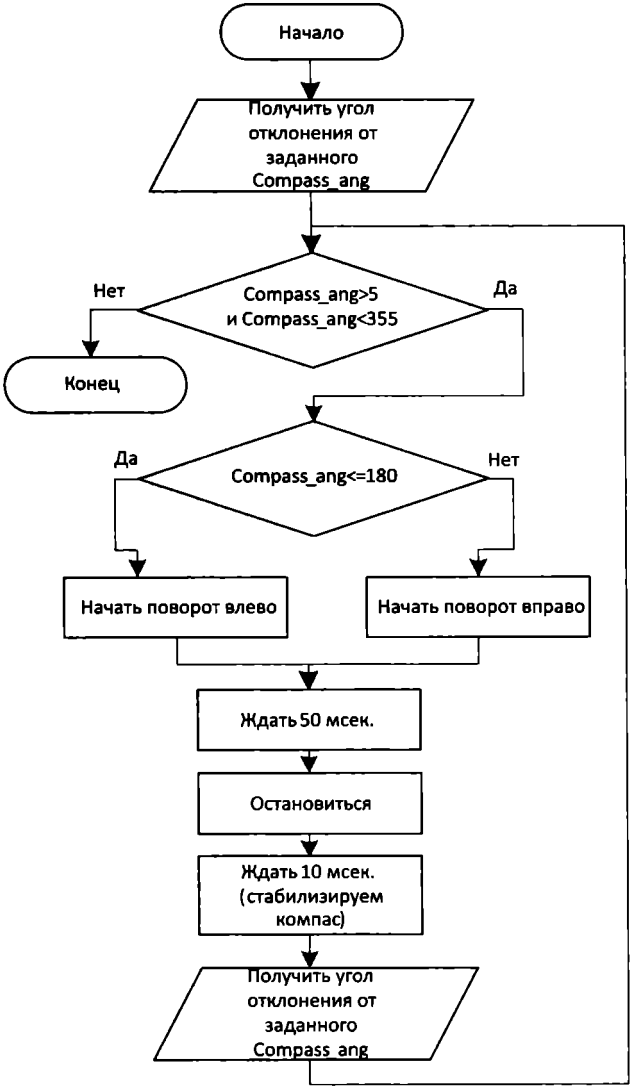


Рис. 13.10. Алгоритм поворота робота на северный магнитный полюс

Дополнительные материалы по калибровке

Хотя магнитометр HMC5883L не является высокоточным прибором, его, тем не менее, устанавливают в большинство современных смартфонов. Увеличить точность показаний магнитометра можно, проведя его калибровку (корректировку).

> ***Интересная статья про калибровку***
>
> По адресу **http://www.vectornav.com/support/library/magnetometer** находится интересная статья на этот счет — она на английском языке, но довольно легко читается.

Показания магнитометра нужно корректировать с учетом, к примеру, температуры. В то же время подобными корректировками можно не только исправить ошибки определения, но и внести новые. Например, вами внесены корректировки, исходя из магнитной обстановки дома, а запускать робота планируется в другом месте, с другой магнитной обстановкой (другими магнитными помехами), — в этом случае сделанные корректировки еще больше запутают электронный компас робота.

В *главе 14* мы рассмотрим гироскоп-акселерометр MPU-6050, имеющий температурный датчик, который подходит для температурной корректировки магнитометра. Однако такие процедуры уже выходят за рамки нашей книги.

Выводы

В этой главе нами изучена работа электронного компаса. Для его реализации выбран магнитометр HMC5883L, создано его подключение к Arduino и организован обмен данными между ними. Модернизированный таким образом робот способен держать направление движения на север, используя показания электронного компаса.

Следующий проект (см. *главу 14*) предоставляет нам еще большую свободу для творчества. Он посвящен интересной теме ориентации робота в пространстве с использованием гироскопа — более точного и быстродействующего прибора.

ГЛАВА 14

РОБОТ, ДЕРЖАЩИЙ НАПРАВЛЕНИЕ ПО ЭЛЕКТРОННОМУ ГИРОСКОПУ-АКСЕЛЕРОМЕТРУ

С помощью электронного компаса не всегда удается получить стабильно хороший результат по удержанию курса. Существенно лучшие результаты можно получить, используя электронный гироскоп и акселерометр.

Мы оснастим робота комплексным электронным прибором MPU-6050, который включает в себя гироскоп и акселерометр, а попутно раскроем принцип действия этих приборов.

Для достижения поставленной цели нам потребуется решить следующие задачи:

1. Понять принцип работы гироскопа.
2. Понять принцип работы акселерометра.
3. Разобраться в различиях электронного и классического гироскопов.
4. Подключить электронный гироскоп-акселерометр MPU-6050 к Arduino.
5. Научиться получать данные с гироскопа-акселерометра MPU-6050.
6. Написать для робота программу, которая, используя данные, полученные с MPU-6050, помогает роботу поворачивать на точные углы, держать направление при езде и ориентироваться в пространстве.

Гироскоп

Гироскоп — это прибор, способный реагировать на изменение ориентации объекта. Простейшим примером гироскопа является волчок. Если волчок раскрутить на перемещаемой поверхности — например, на доске (рис. 14.1, *а*), а затем наклонять эту доску (рис. 14.1, *б* и *в*), то волчок будет пытаться сохранить свою ориентацию.

Гироскоп был изобретен в начале XIX века, и в конце того же века его впервые применили на практике для стабилизации курса торпед. Более подробно о гироскопах можно узнать из специализированной литературы. Нам же пока необходимо

знать о нем лишь то, что гироскоп позволяет отследить углы отклонения устройства, на котором он установлен, от первоначального положения. Гироскопы бывают как одноосевыми, так и трехосевыми — позволяющими получать данные сразу по трем осям, различаются они по типам на роторные (классические гироскопы), вибрационные (сохраняющие направления своих колебаний) и оптические.

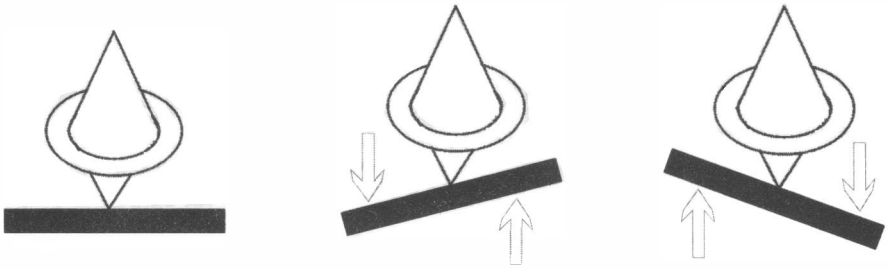


Рис. 14.1. Эффект гироскопа на примере волчка

На рис. 14.2 показано, как можно измерить отклонение машинки от первоначального направления с помощью гироскопа. Черная стрелка — вектор движения, штрихованная — показания гироскопа. В позиции *б* машинка повернула, но стрелка гироскопа осталась на месте и указывает на заданное при включении гироскопа направление. Зная ее отклонение, без труда можно скорректировать направление, но только направление, — гироскоп ничего не знает о пройденном расстоянии и о том, по какому маршруту это расстояние было пройдено.

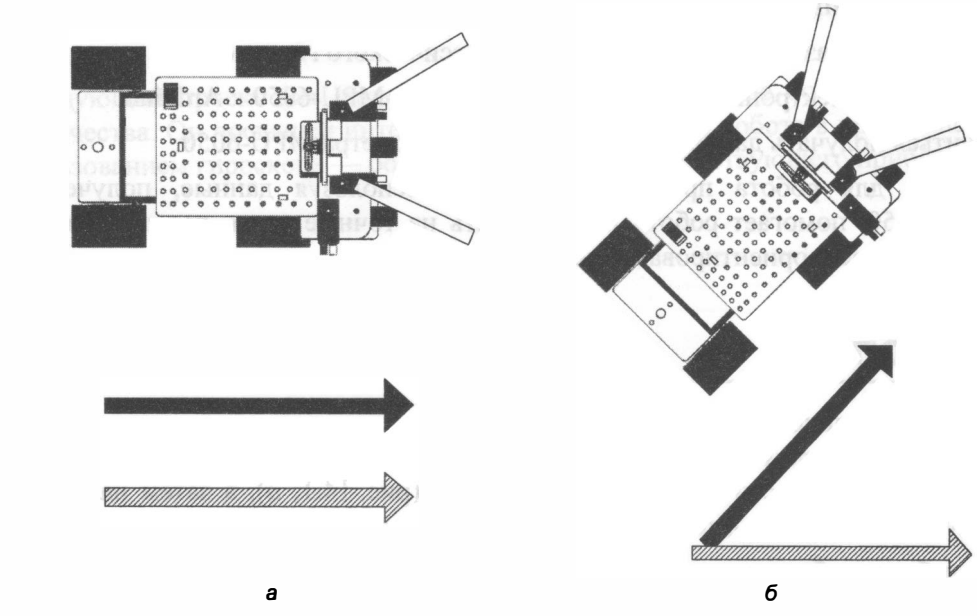


Рис. 14.2. Применение гироскопа для измерения угла поворота

Акселерометр

Акселерометр служит для измерения ускорения. С его помощью можно измерять ускорение движущегося тела и, используя простые формулы, определять положение тела в пространстве. Например, если знать начальную скорость и считывать ускорение в равные малые промежутки времени, несложно вычислить пройденное расстояние (рис. 14.3). Рассмотрим это на примере прямолинейного движения:

$$L_2 = L_1 + \Delta t \times V_1 + \frac{a_2 \times \Delta t^2}{2}, \qquad (14.1)$$

где L_1 — расстояние, пройденное ранее, L_2 — итоговое расстояние, Δt — приращение времени (время движения машинки от А до Б), V_1 — начальная скорость (в точке А), a_2 — полученное от акселерометра ускорение (полагаем на отрезке А–Б ускорение постоянным). Точность измерения расстояния зависит от частоты сбора данных и от вычислительных способностей процессора.

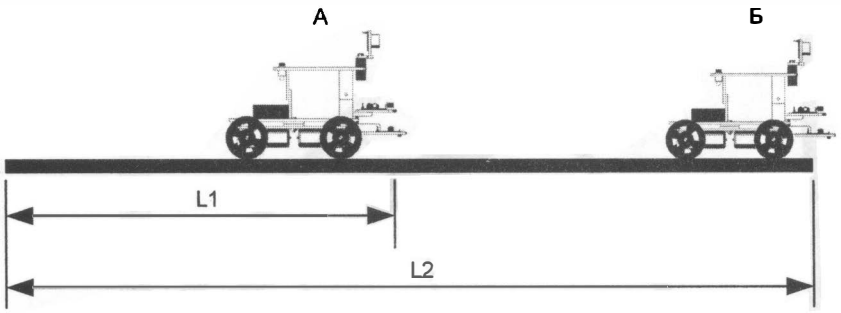


Рис. 14.3. Применение акселерометра для измерения пройденного расстояния

Второе предназначение акселерометра — это нахождение углов отклонения объекта от вертикали. Для точного измерения объект должен быть неподвижен или двигаться прямолинейно без ускорения. В этом случае на него действует только сила притяжения, направленная к центру Земли.

На рис. 14.4 изображен неподвижный объект (робот) на наклонной плоскости. Координатная ось Y акселерометра робота направлена вперед, ось Z — вниз под прямым углом к наклонной плоскости, ось X — к наблюдателю. Искомый угол отклонения объекта от вертикали Ang_{YZ} находится по формуле:

$$Ang_{YZ} = \text{arctg}\left(\frac{a_Y}{a_Z}\right), \qquad (14.2)$$

где a_Y — проекция ускорения свободного падения на ось Y акселерометра, a_Z — проекция на ось Z. Можно также использовать формулы:

$$Ang_{YZ} = \arcsin\left(\frac{a_Y}{g}\right)$$

или

$$Ang_{YZ} = \arccos\left(\frac{a_Z}{g}\right),$$

где g — ускорение свободного падения, но для электронного прибора, который будет рассмотрен далее, рекомендуется использовать формулу (14.2), оперирующую только относительными величинами.

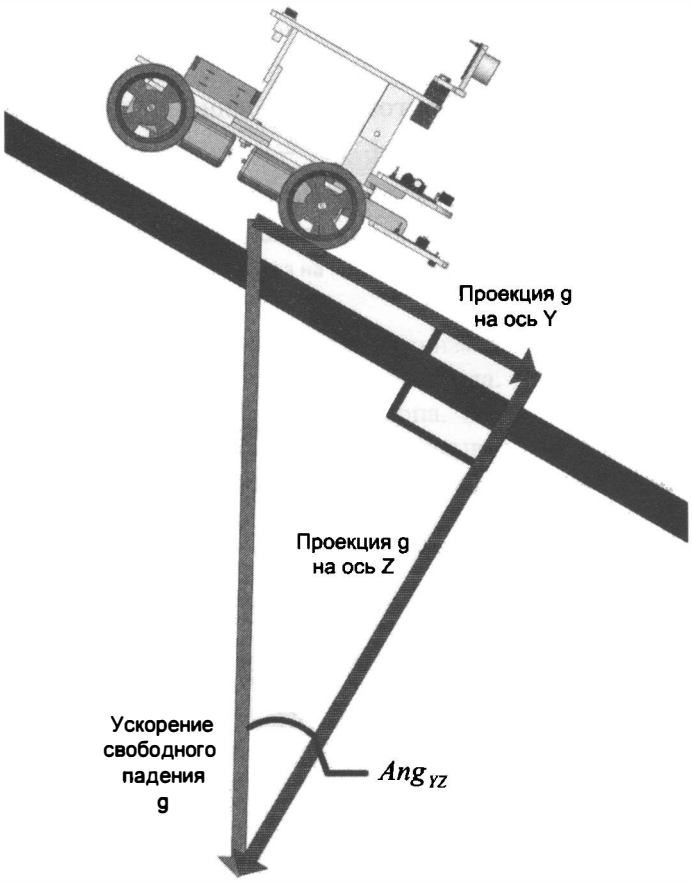


Рис. 14.4. Нахождение угла отклонения от вертикали (робот неподвижен)

Электронный гироскоп

Электронный гироскоп серьезно отличается от классического — он вообще не показывает угол отклонения от заданного направления. Электронный гироскоп — это прибор, регистрирующий и возвращающий текущую угловую скорость (скорость вращения). Зная мгновенную угловую скорость, можно вычислить угол, на который повернется объект за некоторый промежуток времени:

$$Ang_2 = Ang_1 + \Delta t \times V_{ang}, \quad (14.3)$$

где Ang_1 — начальный угол, рассчитанный ранее; Δt — малый промежуток времени, на котором можно считать, что угловая скорость не менялась; V_{ang} — полученное от электронного гироскопа в течение Δt значение угловой скорости; Ang_2 — рассчитанный угол, на который повернулся объект вокруг исследуемой оси с начала поворота. На рис. 14.5 вращение происходит вокруг оси *Z*. Показания гироскопа именно по этой оси следует использовать в расчетах.

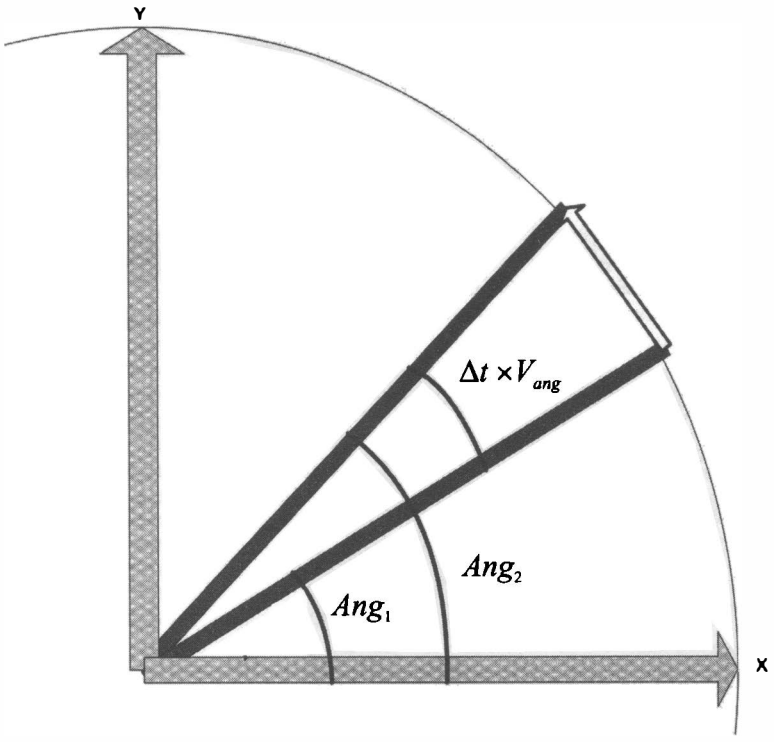


Рис. 14.5. Расчет угла поворота по показаниям электронного гироскопа

Подключение гироскопа-акселерометра MPU-6050

На рис. 14.6 изображена микросхема гироскопа-акселерометра MPU-6050, установленная на печатную плату, что позволяет удобно его подключать. На плате имеется 8 контактов, но для работы достаточно подключить 4 из них: VCC — питание 3,3–5 В (возьмем от платы Arduino), GND — «земля» (подключим к любому нулевому проводу), SDA и SCL — контакты шины I^2C, знакомой нам по *главе 12* (подключаем их к соответствующим портам Arduino: A4 — SDA и A5 —SCL). Имеется на плате также информационный красный светодиод, сигнализирующий о наличии питающего напряжения.

Схема подключения MPU-6050 к Arduino приведена на рис. 14.7. Она схожа со схемой подключения цифрового магнитометра-компаса HMC5883L. Оба эти устройства можно использовать одновременно и даже подключить их параллельно на

Рис. 14.6. Микросхема гироскопа-акселерометра MPU-6050, смонтированная на плате

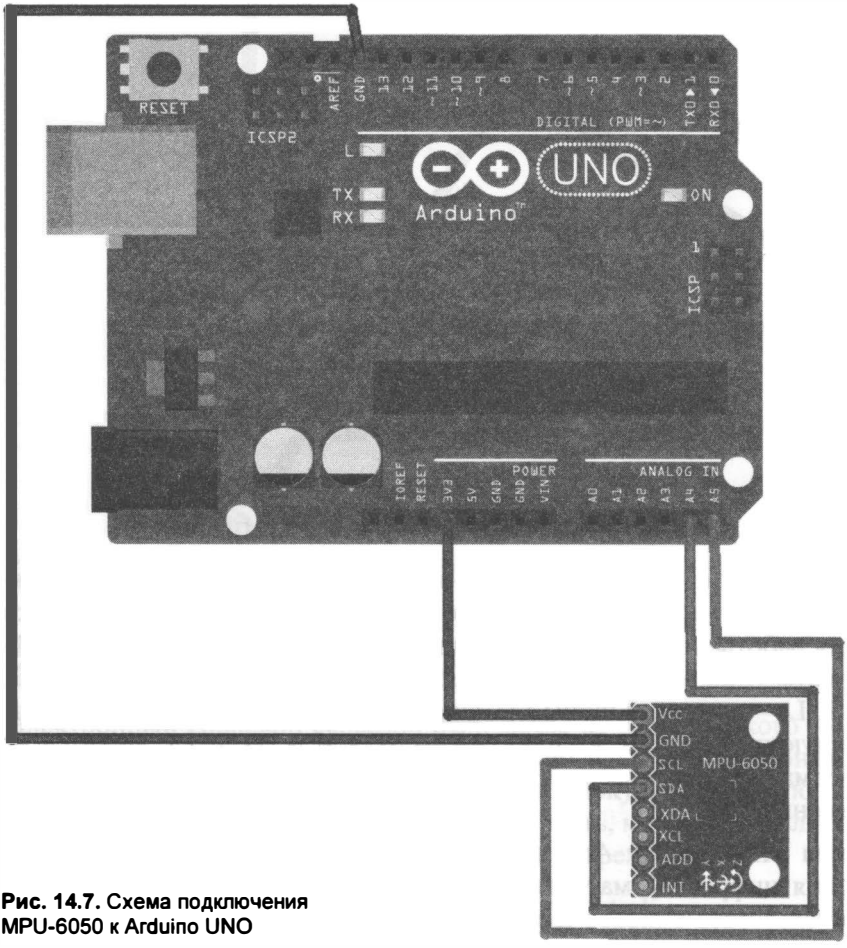


Рис. 14.7. Схема подключения MPU-6050 к Arduino UNO

одни и те же порты Arduino. В таком случае контроллер Arduino должен выступать в качестве ведущего шины и разрешать устройствам передачу информации поочередно.

Получение данных с MPU-6050

Листинг 14.1 реализует функцию получения и вывода данных от гироскопа-акселерометра MPU-6050. Его показания считываются с устройства и выводятся через порт на компьютер:

- на рис. 14.8 представлены показания устройства, когда его ось *Y* приблизительно совпадает с вертикалью и направлена вниз, при этом показания по этой оси наибольшие по модулю: –16340...–16220;

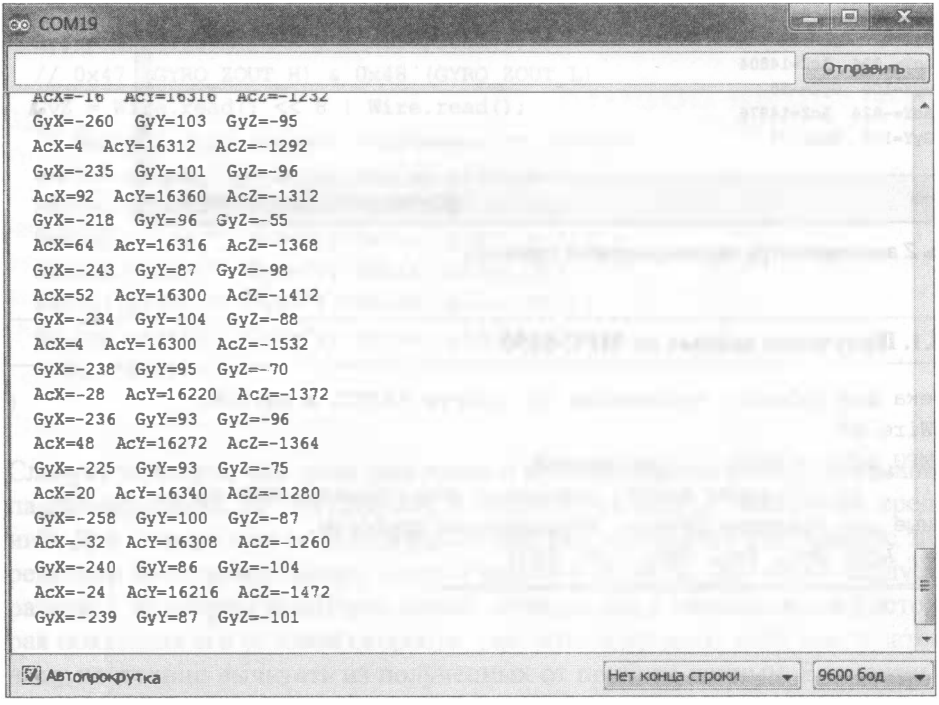


Рис. 14.8. Ось *Y* акселерометра перпендикулярна горизонту

- на рис. 14.9 представлены показания устройства, когда его ось *Z* вертикальна и направлена вверх, — в этом случае наибольшие показания акселерометра снимаются по оси *Z*: 14804...15148.

При вращении устройства изменяются проекции ускорения свободного падения на его оси. В первом случае максимальна проекция на ось *Y*, во втором — на ось *Z*. Показания устройства выводятся в относительных единицах, и их можно использовать для нахождения угла отклонения от вертикали по формуле (14.2).

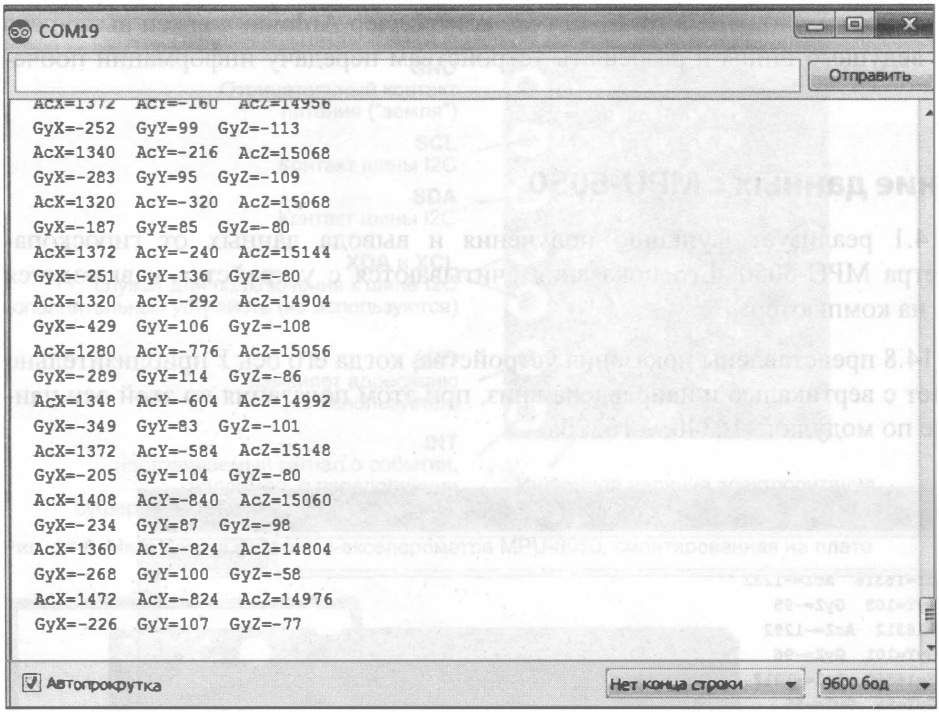


Рис. 14.9. Ось *Z* акселерометра перпендикулярна горизонту

Листинг 14.1. Получение данных от MPU-6050

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
const int MPU_addr = 0x68; // упрощенный
                // I2C-адрес нашего гироскопа-акселерометра MPU-6050.
// переменные для хранения данных, возвращаемых прибором.
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
//=====================================
void setup()
{
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  // Производим запись в регистр энергосбережения MPU-6050
  Wire.write(0x6B);
  Wire.write(0);      // устанавливаем его в ноль.
  Wire.endTransmission(true);
  Serial.begin(9600);
}
//=============== НАЧАЛО ==================
void loop()
{
  Wire.beginTransmission(MPU_addr);
```

```
  // Готовим для чтения регистры с адреса 0x3B.
  Wire.write(0x3B);
  Wire.endTransmission(false);
  // Запрос состояния 14 регистров.
  Wire.requestFrom(MPU_addr, 14, true);
  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcX = Wire.read() << 8 | Wire.read();
  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcY = Wire.read() << 8 | Wire.read();
  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  AcZ = Wire.read() << 8 | Wire.read();
  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  Tmp = Wire.read() << 8 | Wire.read();
  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyX = Wire.read() << 8 | Wire.read();
  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyY = Wire.read() << 8 | Wire.read();
  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  GyZ = Wire.read() << 8 | Wire.read();
  // Вывод в порт данных, полученных от прибора.
  Serial.print("  AcX="); Serial.print(AcX);
  Serial.print("  AcY="); Serial.print(AcY);
  Serial.print("  AcZ="); Serial.println(AcZ);
  Serial.print("  GyX="); Serial.print(GyX);
  Serial.print("  GyY="); Serial.print(GyY);
  Serial.print("  GyZ="); Serial.println(GyZ);
  delay(1000);
}
```

Следует заметить, что даже при полной неподвижности робота показания гироскопа-акселерометра не постоянные, а колеблются вокруг некоторых средних значений. Для повышения точности показаний рекомендуется эти средние значения определить экспериментально, а затем ввести в расчеты. Для этого следует до начала работы с прибором некоторое время держать его в неподвижном состоянии, собирая показания его угловой скорости, рассчитать среднее значение, а затем эту величину постоянно вычитать из полученных от прибора данных. Возможен также температурный дрейф данных — тогда погрешность будет зависеть и от температуры, что учесть можно, но сложнее.

Шкала значений MPU-6050

Как показания прибора из относительных единиц перевести в привычные величины? Для ответа на этот вопрос следует обратиться к документации. На сайте производителя InvenSense (**www.invensense.com**) представлена таблица основных характеристик MPU-6050 (рис. 14.10). В частности, чтобы перевести показания гироскопа в градус/сек, их требуется поделить на 131. Для перевода ускорения в

Gyro Full Scale Range	Gyro Sensitivity	Gyro Rate Noise	Accel Full Scale Range	Magnetometer Full Scale Range	Accel Sensitivity	Digital Output	Logic Supply Voltage	Operating Voltage Supply
(°/sec)	(LSB/°/sec)	(dps/√Hz)	(g)		(LSB/g)		(V)	(V +/-5%)
±250	131	0.005	±2	N/A	16384	I²C	1.8V±5% or VDD	2.375V–3.46V
±500	65.5	0.005	±4		8192			
±1000	32.8	0.005	±8		4096			
±2000	16.4	0.005	±16		2048			

Рис. 14.10. Основные характеристики MPU-6050

привычную для нас систему мер требуется поделить показания прибора на 16384, а затем умножить на 9,8 м/с².

Модернизация робота

Схема подключения

Оснастим гироскопом-акселерометром робота, уже собранного в предыдущей главе. Для этого заменим электронный компас на MPU-6050 (рис. 14.11), используя те же контакты и место на роботе. Впрочем, снимать компас не обязательно, если есть желание его оставить. На Arduino Sensor Shield v5.0 есть два выхода для SDA (A4) и SCL (A5), разъемов питания на 5 В также достаточно.

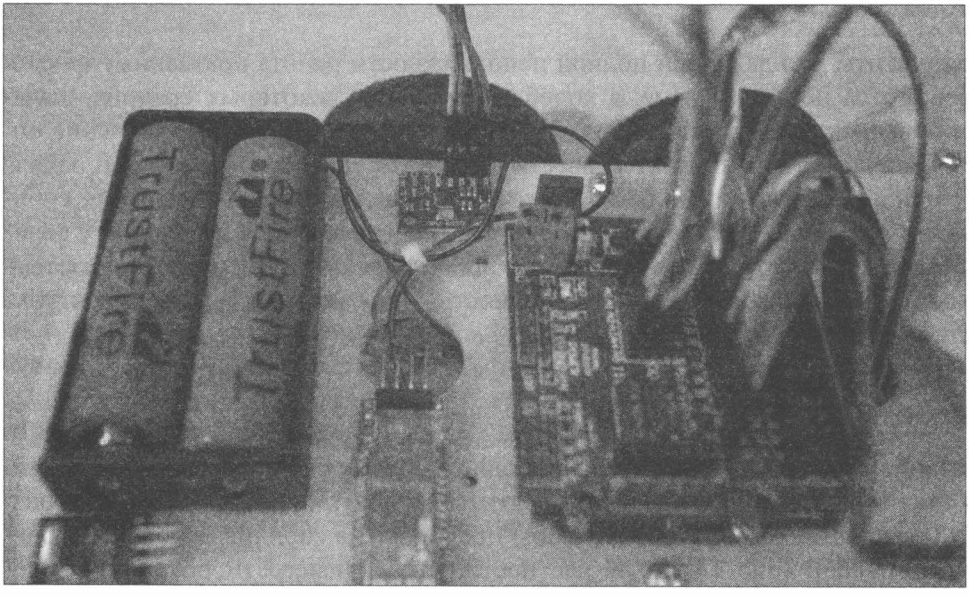


Рис. 14.11. MPU-6050 установлен на корпус робота

Программирование

Основные функции

Команды, запускающие гироскоп, вынесем в отдельную функцию `giroscop_setup()`, она будет запускаться единожды и давать команду гироскопу на запуск, т. к. после подключения питания MPU-6050 находится в состоянии ожидания и не будет отвечать на запросы.

Функцию, снимающую показания с гироскопа и помещающую их в переменные `AcX`, `AcY`, `AcZ`, `Tmp`, `GyX`, `GyY` и `GyZ`, назовем `Data_mpu6050()`. Глобальные переменные будут хранить следующие данные: `AcX`, `AcY`, `AcZ` — текущие показания акселерометра (проекция ускорения по осям), `GyX`, `GyY`, `GyZ` — угловая скорость в относительных единицах, `Tmp` — температура.

Заменим стандартную функцию `delay(time)`, которая создает задержку выполнения программы и используется везде, где нужно определить время на выполнение некоторого действия (например, на поворот или движение), функцией, которая создает задержку, во время которой опрашивает гироскоп и пересчитывает угол поворота. Назовем эту функцию `time_gyro(float mill_sec)` — входным параметром для нее будет время в миллисекундах (с возможностью вводить дробные значения меньше единицы). Для работы этой функции потребуется глобальная переменная `timer`, которая будет хранить время последнего снятия данных с MPU-6050. Это нужно для точного представления промежутков времени, для которых на основании полученной угловой скорости станет рассчитываться приращение поворота.

Упрощенный алгоритм, иллюстрирующий работу функции `time_gyro()`, приведен на рис. 14.12.

Важной характеристикой мобильного робота является возможность обеспечить прямолинейное движение. Создадим функцию, которая позволяет роботу двигаться прямо в течение определенного промежутка времени. Для этого потребуется, чтобы робот, двигаясь, постоянно опрашивал гироскоп и, в случае критичного поворота в сторону, выполнял компенсирующий поворот в обратную сторону. Назовем эту функцию `forward_t(long microsec)`. Упрощенный алгоритм ее работы приведен на рис. 14.13.

Создадим функцию, которая позволяет роботу поворачивать на определенный угол. Назовем ее `Angle(float ang)`. В качестве параметра она принимает угол поворота и в течении работы осуществляет микроповороты в указанную сторону, постоянно сличая результат с заданным значением. Завершается функция при достижении роботом заданного угла поворота. Упрощенный алгоритм ее работы приведен на рис. 14.14.

Рассмотренные основные функции и ряд второстепенных (вспомогательных) поместим в отдельный файл **gyro_acsel.h** — он будет подключаться к основной программе функцией `#include "gyro_acsel.h"`. Листинг 14.2 демонстрирует содержимое этого файла.

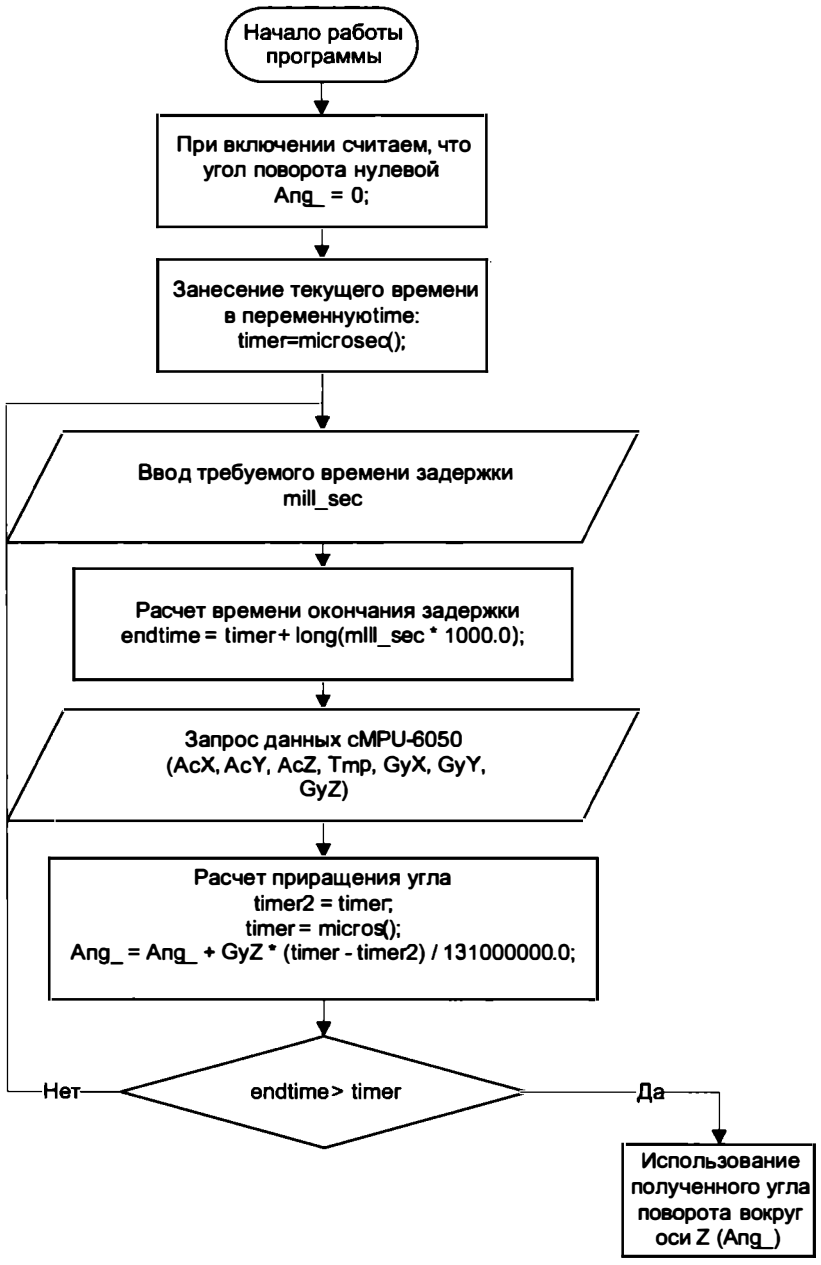


Рис. 14.12. Упрощенный алгоритм пересчета угла поворота робота

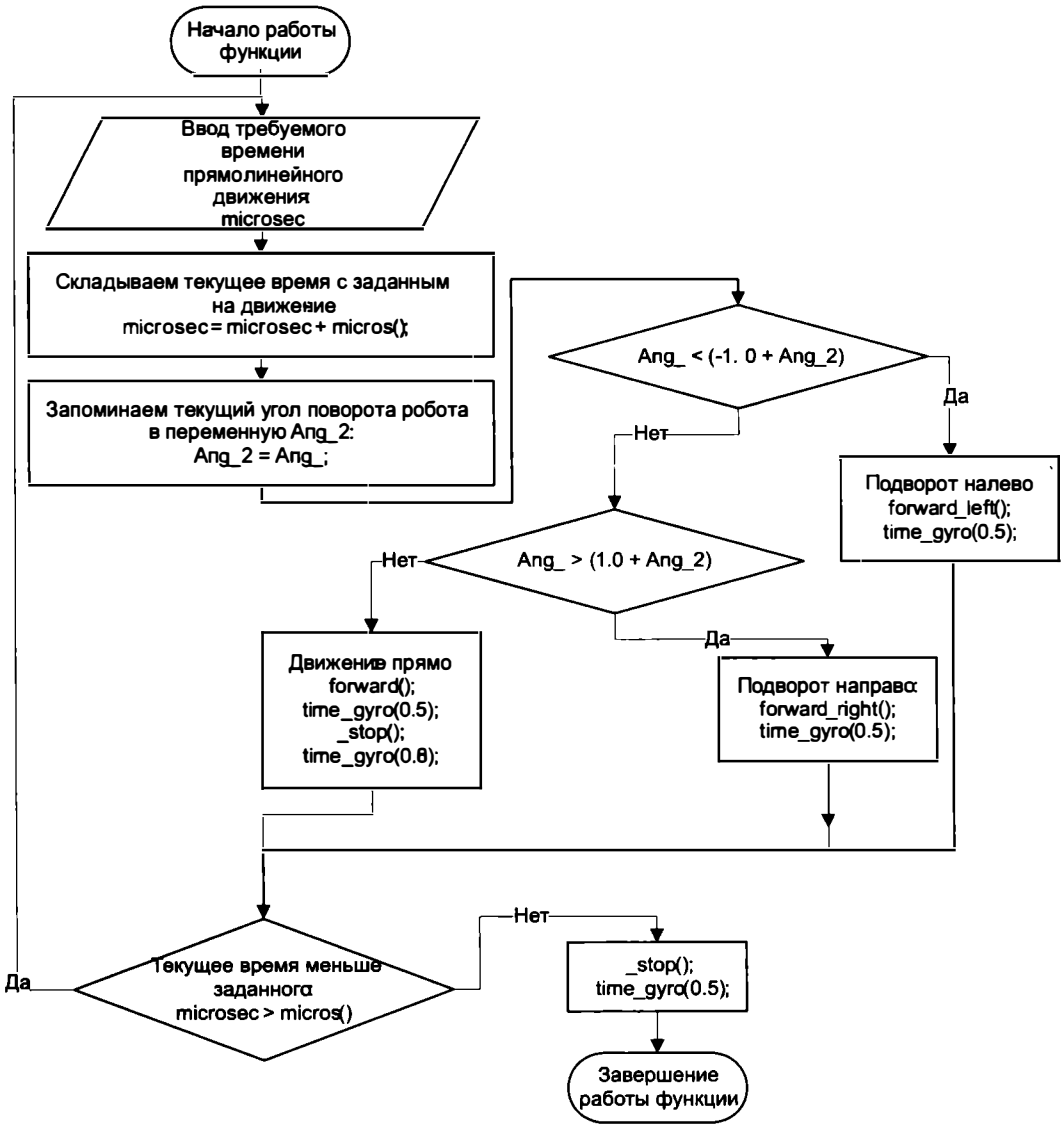


Рис. 14.13. Упрощенный алгоритм прямолинейного движения робота в течение заданного времени

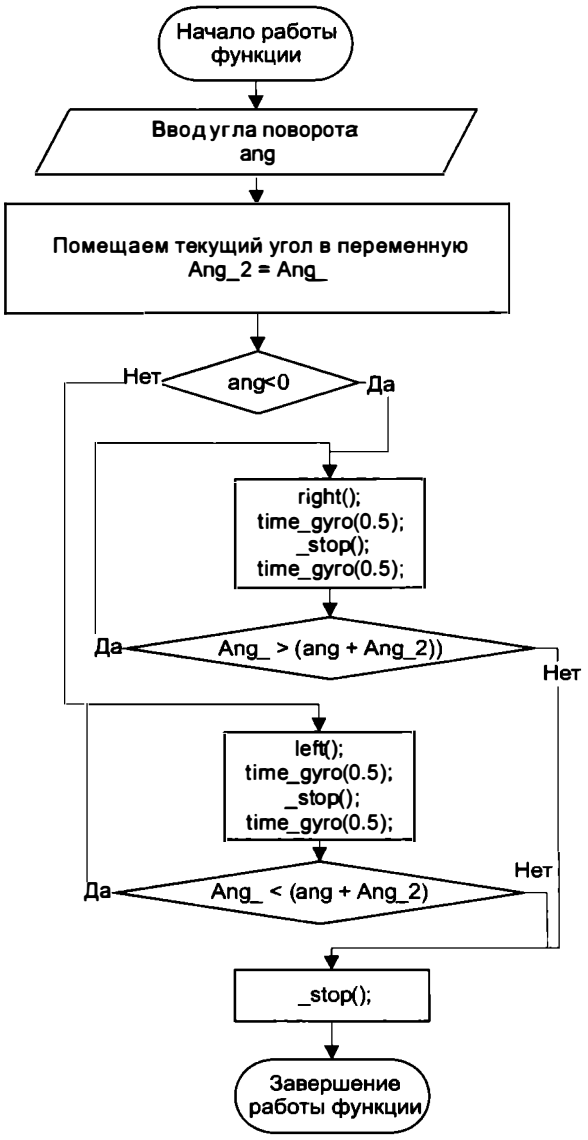


Рис. 14.14. Упрощенный алгоритм поворота робота на заданный угол

Листинг 14.2. Файл gyro_acsel.h

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
//#include <Wire.h>
// Коррекция склонения для акселерометра по Y
#define ACYSUMCORRECTION -0.95
// Коррекция склонения для акселерометра по X
#define ACXSUMCORRECTION -2.17
int PROG;
```

```
// упрощенный I2C адрес нашего гироскопа/акселерометра MPU-6050.
const int MPU_addr = 0x68;
// переменные для хранения данных возвращаемых прибором.
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
float AcYsum, AcXsum, GyXsum;
// Угол поворота вокруг оси Z.
float Ang_;
// Ускорения по осям  - для анализа наклона
float AcZtec, AcYtec, AcXtec;
float CompensatorZ, CompensatorX,CompensatorY,  CompensatorAcX;
unsigned long timer;
////////////////////////////////////////////////////////////////////////
//// Считывание данных с mpu6050
////////////////////////////////////////////////////////////////////////
void Data_mpu6050()
{
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B); // Готовим для чтения регистры с адреса 0x3B.
  Wire.endTransmission(false);
  // Запрос 14 регистров.
  Wire.requestFrom(MPU_addr, 14, true);
  // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
  AcX = Wire.read() << 8 | Wire.read();
  // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
  AcY = Wire.read() << 8 | Wire.read();
  // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
  AcZ = Wire.read() << 8 | Wire.read();
  // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
  Tmp = Wire.read() << 8 | Wire.read();
  // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
  GyX = Wire.read() << 8 | Wire.read();
  // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
  GyY = Wire.read() << 8 | Wire.read();
  // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
  GyZ = Wire.read() << 8 | Wire.read();
}
////////////////////////////////////////////////////////////////////////
/// Запуск гироскопа
////////////////////////////////////////////////////////////////////////
void giroscop_setup()
{
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);  // Производим запись в регистр энергосбережения MPU-6050
  Wire.write(0);     // устанавливаем его в ноль
  Wire.endTransmission(true);
}
```

```
//////////////////////////////////////////////////////////////
// задержка, но с расчетом изменения Z угла.
//////////////////////////////////////////////////////////////
void time_gyro(float mill_sec)
{
  unsigned long ms = mill_sec;
  unsigned long timer2;
  float m_t_d131;
  // unsigned long endtime = micros() + long(ms * 1000.0);
  unsigned long endtime = timer + long(ms * 1000.0);
  do {
    Data_mpu6050();
    timer2 = timer;
    timer = micros();
    m_t_d131 = (float)(timer - timer2) / 131000000.0;
    Ang_ = Ang_ + ((float)(GyZ) - CompensatorZ) * m_t_d131;
    AcYsum = (atan2(AcY, AcZ)) * RAD_TO_DEG+ACYSUMCORRECTION;
//balancing_zerro;
    // Измерение наклона по X.
    // Использование Комплементарного фильтра,
    // alfa - коэффициент фильтра.
    float alfa = 0.005;
    AcXtec = (1.0 - alfa) * (AcXtec + ((float)GyX - CompensatorX)* m_t_d131) +
alfa * AcYsum;
    AcXsum = - (atan2(AcX, AcZ)) *
RAD_TO_DEG+ACXSUMCORRECTION;//+balancing_zerro;
    // Измерение наклона по Y.
    // Использование Комплементарного фильтра,
    // alfa - коэффициент фильтра.
    AcYtec = (1.0 - alfa) * (AcYtec + ((float)GyY - CompensatorY) * m_t_d131) +
alfa * AcXsum;
    } while (endtime > timer);
}
//////////////////////////////////////////////////////////////////////////
// Компенсация нуля гироскопа
void Calc_CompensatorZ(float mill_sec)
{
  long ms = mill_sec;
  float i = 0;
  CompensatorZ = 0;
    CompensatorX = 0;
    CompensatorY = 0;
  timer = micros();
  unsigned long endtime = micros() + long(ms * 1000.0);
  while (endtime > timer) {
    CompensatorZ += (float)(GyZ);
    CompensatorX += (float)(GyX);
```

```
    CompensatorY += (float)(GyY);
    timer = micros();
    Data_mpu6050();
    delayMicroseconds(100);
    i++;
  }
  CompensatorZ /= i;
  CompensatorX /= i;
  CompensatorY /= i;
}
////////////////////////////////////////////////////////////////////////
// Ехать вперед заданное время. ////////
////////////////////////////////////////////////////////////////////////
void forward_t( long microsec)
{
  bool flag = false;
  microsec = microsec + micros();
  long microsec_begin_tormog = microsec;
  float Ang_2 = Ang_;
  do {
    if (Ang_ < (-1.0 + Ang_2)) {
      forward_left(); time_gyro(0.5);
    }
    else if (Ang_ > (1.0 + Ang_2)) {
      forward_right(); time_gyro(0.5);
    }
    else {
      forward(); time_gyro(0.5); _stop(); time_gyro(0.8);
    }
  } while (microsec > micros());
  _stop();
  time_gyro(0.5);
}
////////////////////////////////////////////////////////////////////////
// Ехать назад заданное время. ////////
////////////////////////////////////////////////////////////////////////
void backward_t(unsigned long microsec)
{
  bool flag = false;
  microsec = microsec + micros();
  long microsec_begin_tormog = microsec;
  float Ang_2 = Ang_;
  do {
    if (Ang_ < (-1.0 + Ang_2)) {
      backward_right(); time_gyro(0.5);
    }
```

```
    else if (Ang_ > (1.0 + Ang_2)) {
      backward_left(); time_gyro(0.5);
    }
    else {
      backward(); time_gyro(0.5);
    }
  } while (microsec > micros());
  _stop();
  time_gyro(0.5);
}
////////////////////////////////////////////////////////////////////////////
// Поворот на заданный угол. ////////
////////////////////////////////////////////////////////////////////////////
void Angle(float ang)
{
  bool flag = false;
  float Ang_2 = Ang_;
  // возможно стоит уменьшить скорость, если угол маленький.
  if (ang == 0) return;
  if (ang < 0)
  {
    do {
      right(); time_gyro(0.5); _stop(); time_gyro(0.5);
    } while (Ang_ > (ang + Ang_2));
  }
  else
  {
    do {
      left(); time_gyro(0.5); _stop(); time_gyro(0.5);
    } while (Ang_ < (ang + Ang_2));
  }
  _stop();
  Ang_2 = Ang_ - (ang + Ang_2);
  if (abs(Ang_2) > 1.0)
    Angle(-Ang_2);
}

////////////////////////////////////////////////////////////////////////////
// Поворот на заданный угол при движении вперед. ////////
////////////////////////////////////////////////////////////////////////////
void Angle_motion(float ang)
{
  bool flag = false;
  float Ang_2 = Ang_;
  // возможно стоит уменьшить скорость, если угол маленький.
  if (ang == 0) return;
  if (ang < 0)
```

```
  {
    do {
      forward_right(); time_gyro(0.5); //_stop(); time_gyro(0.5);
    } while (Ang_ > (ang + Ang_2));
  }
  else
  {
    do {
      forward_left(); time_gyro(0.5); //_stop(); time_gyro(0.5);
    } while (Ang_ < (ang + Ang_2));
  }
  _stop();
}
////////////////////////////////////////////////////////////////////////////
// Держать угол определенное время
////////////////////////////////////////////////////////////////////////////
void Angle_t( long microsec)
{
  //time_razgon
  bool flag = false;
  microsec = microsec + micros();
  long microsec_begin_tormog = microsec ;
  float Ang_t = Ang_;
  _stop();
  do {
    if (Ang_ < (Ang_t - 1.0))   {
      left();   time_gyro(1);   _stop();
    }
    else if (Ang_ > (Ang_t + 1.0)) {
      right();       time_gyro(1);       _stop();
    } else   _stop(); time_gyro(1);
  } while (microsec > micros());
  _stop(); time_gyro(10);
}
```

Программа

В листинге 14.3 приведен пример программы, реализующей повороты и прямолинейные движения робота на основании данных гироскопа. Для подачи команд был подключен инфракрасный пульт дистанционного управления. При нажатии кнопок от «1» до «6» робот поворачивает на угол от 10 до 60 градусов соответственно. При нажатии кнопки «вперед», робот едет вперед 10 секунд и компенсирует попытки себя повернуть, при нажатии кнопки «назад» то же происходит в обратном направлении. При нажатии кнопки «стоп» робот замирает на 10 секунд и, если его повернуть руками, он будет пытаться, как неваляшка, вернуться в прежнее положение.

Для работы программы потребуются файлы библиотеки `IRremote.h` из *главы 8*, уже неоднократно использованная нами библиотека управления моторами `motor.h`

и созданная в этой главе библиотека управления гироскопом `gyro_acsel.h`. Также потребуется подключить стандартную библиотеку `Wire.h`.

Для компенсации утечки нуля гироскопа робот в течение 3 секунд после включения анализирует утечку нуля. Компенсация утечки нуля в дальнейшем значительно уменьшает погрешности расчетов угла поворота робота.

При выполнении команд на роботе загорается информационный светодиод, он горит до окончания выполнения текущей команды.

Демонстрационный ролик

Демонстрацию работы гироскопа и программы из листинга 14.3 можно посмотреть в учебном ролике «Arduino робот/robot и MPU-6050 (гироскоп/gytoscope)» на канале автора в YouTube: **https://www.youtube.com/watch?v=76cEsG23BY4**.

Листинг 14.3. Программа управления движениями робота с использованием гироскопа

```
//Подключение библиотеки I2C шины.
#include <Wire.h>
//Подключение библиотеки удаленного управления.
#include "IRremote.h"
//Подключение библиотеки управления моторами.
#include "motor.h"
//Подключение библиотеки управления гироскопом.
#include "gyro_acsel.h"
// Порт для IR-приемника.
int RECV_PIN = 7;
// Создание IR-приемника.
IRrecv irrecv(RECV_PIN);
//Переменная для результатов IR-приемника.
decode_results results;
int diod_pin = 6;
//=======================================
void setup()
{
  //запуск гироскопа.
  giroscop_setup();
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
  // Запуск IR-приемника.
  //Расчет компенсации утечки нуля гироскопа.
  // Робот стоит на месте 3 секунды
  Calc_CompensatorZ(3000);
  //переводим пин светодиода в режим вывода.
  pinMode(diod_pin, OUTPUT);
```

```
  irrecv.enableIRIn();
  Serial.begin(9600);
}
// Основная программа.
void loop()
{
  if (irrecv.decode(&results))
  {
    Serial.println(results.value, HEX);
    switch (results.value) {
      // 1
      case 0xC101E57B:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(10);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // 2
      case 0x97483BFB:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(20);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // 3
      case 0xF0C41643:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(30);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // 4
      case 0x9716BE3F:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(40);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // 5
      case 0x3D9AE3F7:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(50);
```

```
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // 6
      case 0x6182021B:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle(60);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // Вперед 10 сек
      case 0x511DBB:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        forward_t(10000000);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      // Назад 10 сек
      case 0xA3C8EDDB:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        backward_t(10000000);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
      case 0xD7E84B1B:
        //Включаем светодиод.
        digitalWrite(diod_pin, 1);
        Angle_t(100000000);
        //Гасим светодиод.
        digitalWrite(diod_pin, 0);
        break;
    }
    irrecv.resume();
  }
  time_gyro(0.5);
}
```

Выводы

В этой главе мы изучили принципы работы гироскопа и акселерометра. Разобрались с различиями между электронным и классическим гироскопами. Подключили гироскоп-акселерометр MPU-6050 к Arduino, написали программу его опроса. Соз-

дали библиотеку, позволяющую роботу использовать гироскоп. Протестировали работу робота с гироскопом в реальной обстановке.

Таким образом, цель, поставленная нами перед собой в этой главе, достигнута.

Рекомендуем вам включить фантазию и самостоятельно модифицировать программу из листинга 14.3 — например, научить робота строить маршрут в виде сложных геометрических фигур, отъезжать и точно возвращаться назад в место старта. Знания, приобретенные в этой главе, будут востребованы при обучении робота кегельрингу в *главе 15*.

ГЛАВА 15

РОБОТ, ИГРАЮЩИЙ В КЕГЕЛЬРИНГ

Цель, которую мы ставим себе в этой главе, — сконструировать робота, выбивающего кегли из круга, очерченного черной линией. Робот должен уметь сортировать кегли по цвету и выбивать кегли только заданного (черного или белого) цвета, не выезжая при этом за ограничительную линию. Пример кегельринга представлен на рис. 15.1: диаметр круга — 1 метр, количество кеглей — до 8, возможный цвет кеглей — черный и белый, диаметр кегли — 70 мм, высота — 120 мм.

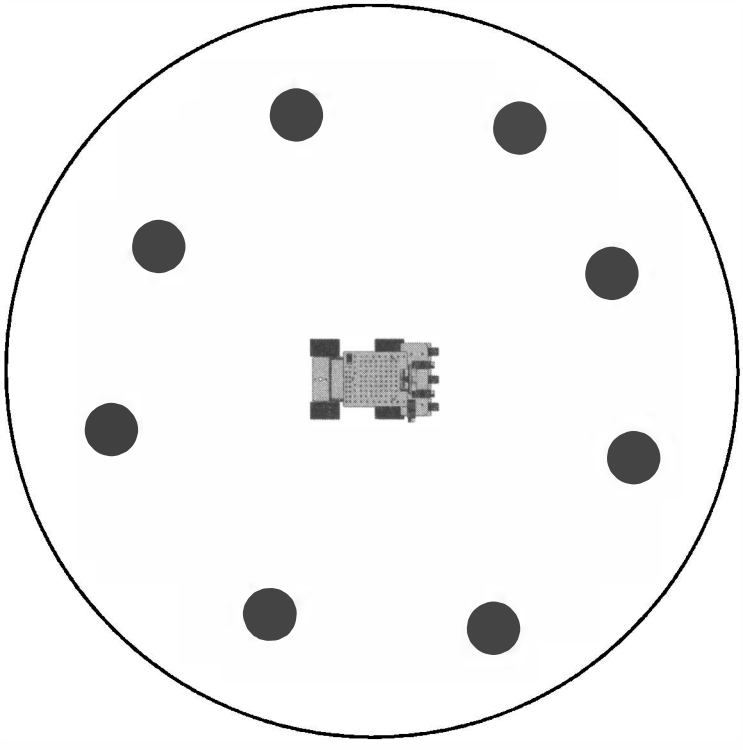


Рис. 15.1. Робот на ринге

Для достижения поставленной цели нам потребуется решить следующие задачи:

1. Разработать механизмы поиска кегли и определения ее цвета, модифицировать робота, если необходимо.
2. Определить, как ограничить движения робота черной линией, модифицировать робота, если необходимо.
3. С учетом используемых технических решений создать алгоритм поиска кегли, определения ее цвета и выталкивания за пределы круга.
4. На основании созданного алгоритма составить программу.
5. Реализовать разработанного робота, провести его тестирование и добиться стабильной работы.

Простой кегельринг

В простом варианте кегельринга роботу требуется вытолкнуть из белого круга все кегли. Эта задача может быть решена довольно просто, если заставить робота ехать вдоль черной линии на расстоянии около 15 см от нее и снабдить его косым бампером (рис. 15.2).

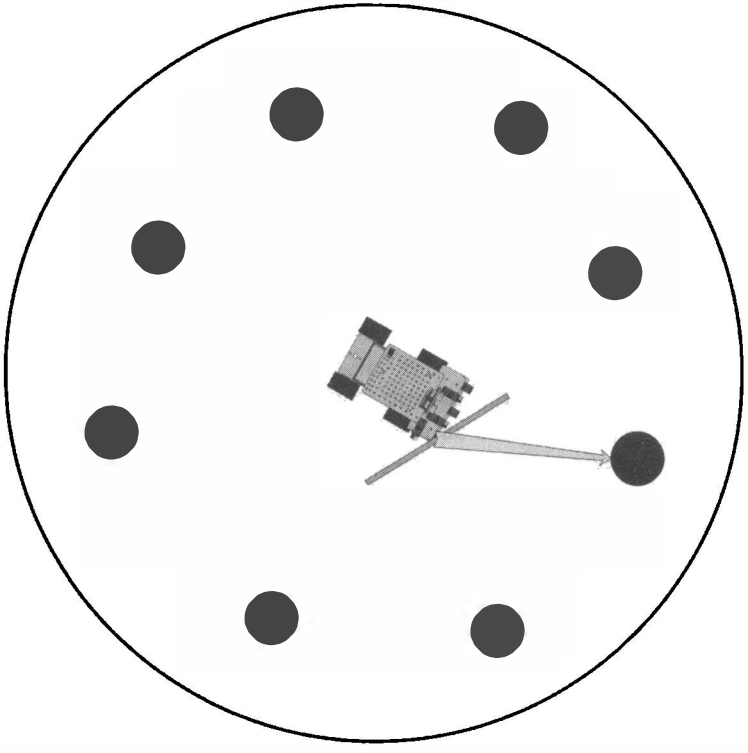


Рис. 15.2. Робот, выталкивающий все кегли (начало)

В этом случае робот должен выехать из центра круга, подъехать к ограничительной линии, не доезжая до нее примерно 20 см, и проехать вдоль нее по кругу. Сделав один полный круг, робот гарантированно вытолкнет все кегли (рис. 15.3).

Заставить робота двигаться по кругу можно, регулируя скорость вращения правых и левых колес, но если сделать так не получится, то просто чередовать краткие периоды движения по прямой линии и краткие повороты влево.

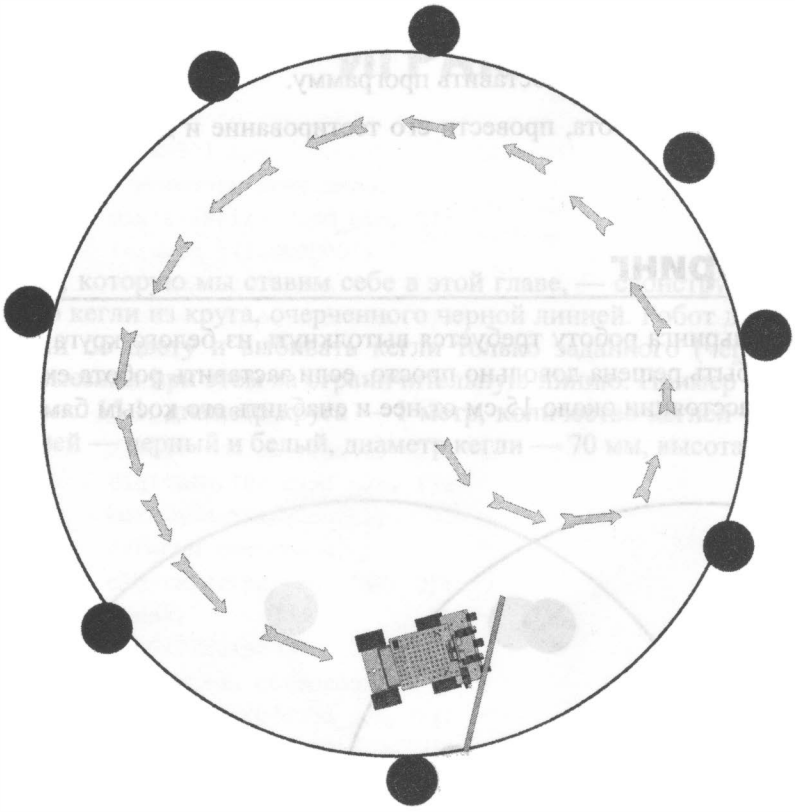


Рис. 15.3. Робот, выталкивающий все кегли (окончание)

Из-за неравномерности движения робота вследствие проскальзывания по поверхности может возникнуть проблема выхода его за пределы ограничительной линии. Для ее решения нужно заставить робота следить за тем, чтобы правый конец его бампера все время на этой линии находился. То есть, он каким-то образом должен отслеживать, что правый конец — только конец — бампера находится на черной линии, и подворачивать в соответствии с полученной информацией. Например, если бампер сильно углубился в черную линию, то сильнее подвернуть влево, а если бампер эту линию потерял, то подвернуть вправо.

Реализуем программу, выталкивающую все кегли с использованием возможностей гироскопа и функций, которые были созданы в предыдущей главе. При этом мы не станем устанавливать датчик линии на бампер, но реализуем круговое движение при помощи гироскопа.

В программе использован старт от IR-пульта — после включения и прохождения тестов робот ждет, когда на пульте будет нажата кнопка «вперед». Затем робот выходит на «орбиту» и начинает движение по кругу до тех пор, пока не будет нажата кнопка «стоп» на пульте.

При реализации программы следует на основании экспериментов скорректировать время движения вперед для выхода на «орбиту», а также скорректировать время прямолинейного движения в цикле, добившись требуемого радиуса круга.

Листинг 15.1. Робот выталкивает все кегли

```
//Подключение библиотеки I2C шины.
#include <Wire.h>
//Подключение библиотеки удаленного управления.
#include "IRremote.h"
//Подключение библиотеки управления моторами.
#include "motor.h"
//Подключение библиотеки управления гироскопом.
#include "gyro_acsel.h"
// Порт для IR-приемника.
int RECV_PIN = 7;
// Создание IR-приемника.
IRrecv irrecv(RECV_PIN);
//Переменная для результатов IR-приемника.
decode_results results;
int diod_pin = 6;
//=========================================
void setup()
{
  //запуск гироскопа.
  giroscop_setup();
  // Переменные - номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
  // Запуск IR-приемника.
  //Расчет компенсации утечки нуля гироскопа.
  Calc_Compensator2(3000);
  //переводим пин светодиода в режим вывода.
  pinMode(diod_pin, OUTPUT);
  //Serial.begin(9600);
  digitalWrite(diod_pin, 1);
  delay(500);
  digitalWrite(diod_pin, 0);
  irrecv.enableIRIn();
}
```

```
// Основная программа.
void loop()
{
  bool start = false;
  while (!start)
  {
    if (irrecv.decode(&results))
    {
      //Если нажата кнопка "вперед" - стартуем
      if (results.value == 0x511DBB)
      {
        start = true;
        digitalWrite(diod_pin, 1);
      }
      irrecv.resume();
    }
  }
  //Выход на нужный радиус.
  // Время указано в микросекундах
  // и его нужно настроить экспериментально.
  forward_t(200000);
  // Поворот влево в движении
  // на 90 градусов.
  Angle_motion(90);

  while (start)
  {
    //Едет прямо 1000 микросекунд.
    forward_t(1000);
    //Поворачивает на 5 градусов
    Angle_motion(5);
    if (irrecv.decode(&results))
    {
      //Если нажата кнопка "стоп" - останавливаемся
      if (results.value == 0xD7E84B1B)
      {
        start = false;
        digitalWrite(diod_pin, 0);
      }
      irrecv.resume();
    }
  }
}
```

Двухцветный кегельринг

Усложним задачу и научим робота выбивать кегли с учетом их цвета: только черные или только белые. Сортировка по цветам — задача решаемая, поскольку существуют датчики определения цвета (например, TCS230).

Порядок обхода

Поскольку перед роботом поставлена задача выбивать кегли только определенного цвета, логично принять за основу порядок обхода им кегельринга, изображенный на рис. 15.4.

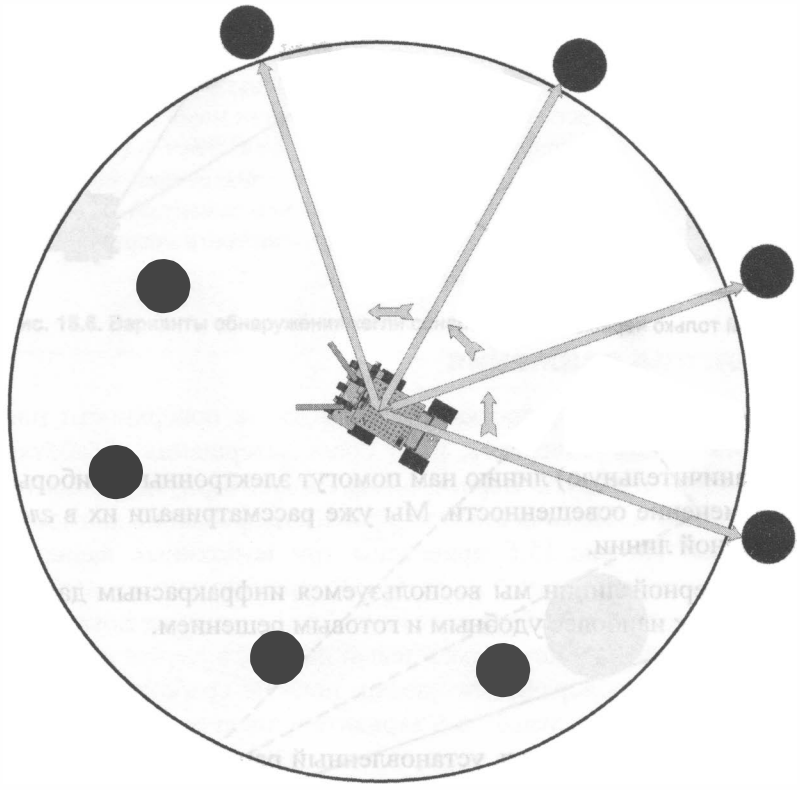


Рис. 15.4. Порядок поочередной обработки кеглей роботом

Как можно видеть, робот при этом будет поочередно подъезжать к каждой кегле и определять ее цвет, после чего принимать решение, удалять ее или нет. Порядок обхода кегельринга при выбивании, например, только черных кеглей представлен на рис. 15.5: робот находит кеглю, подъезжает к ней, определяет ее цвет и, в зависимости от цвета кегли, либо выталкивает ее, либо возвращается на середину и продолжает обход.

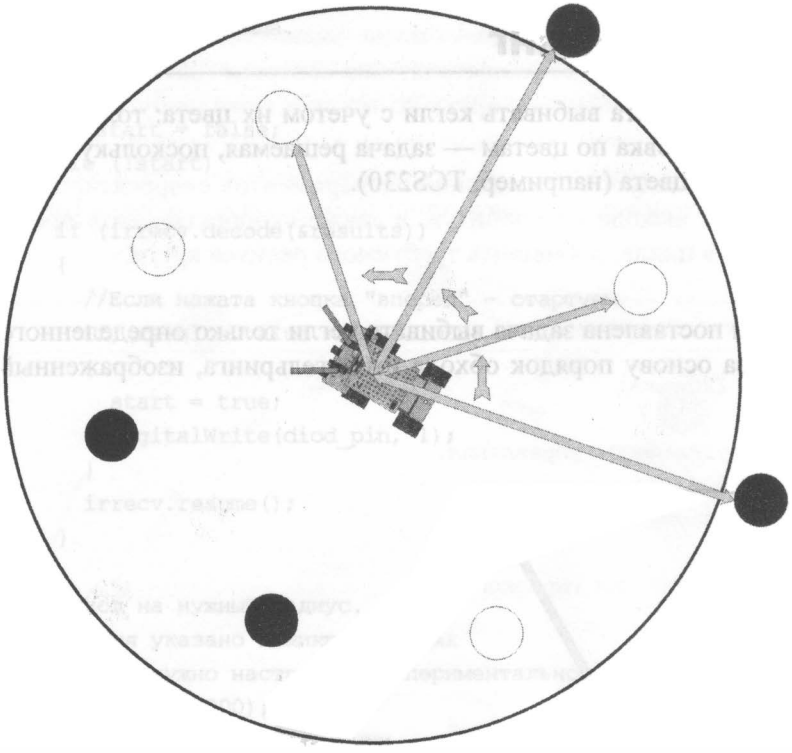


Рис. 15.5. Робот, выталкивающий только черные кегли

Обнаружение черной линии

Обнаружить черную (ограничительную) линию нам помогут электронные приборы, которые реагируют на изменение освещенности. Мы уже рассматривали их в *главе 9* при следовании по черной линии.

Для определения наличия черной линии мы воспользуемся инфракрасным датчиком отражения TCRT 5000, как наиболее удобным и готовым решением.

Обнаружение кегли

Для обнаружения кегли можно использовать установленный ранее ультразвуковой датчик расстояния на сервомоторе. Предлагается определять только кегли, находящиеся слева от робота, считая, что правая сторона уже исследована или будет исследована после прохождения роботом полного круга. Алгоритм поиска кегли следующий: робот вращает головой влево, постепенно увеличивая угол. После получения данных о препятствии, робот анализирует угол, на который повернута голова, и поворачивает себя на такой же угол, наводясь на кеглю.

К сожалению, ультразвуковой сонар имеет широкую диаграмму направленности (около 30 градусов), и его область обнаружения расширяется с увеличением расстояния до цели, вследствие чего угол поворота головы не всегда точно укажет

на расположение кегли. Возможные варианты обнаружения кегли приведены на рис. 15.6, и большинство из них показывают, что объект обнаружен, но, повернув на измеренный угол, робот ошибется и проедет мимо (кроме вариантов *а* и *в*).

Чтобы этого не произошло, потребуется фактически ощупывать кеглю ультразвуковым лучом, находить ее края и по краям определять центр цели (рис. 15.7):

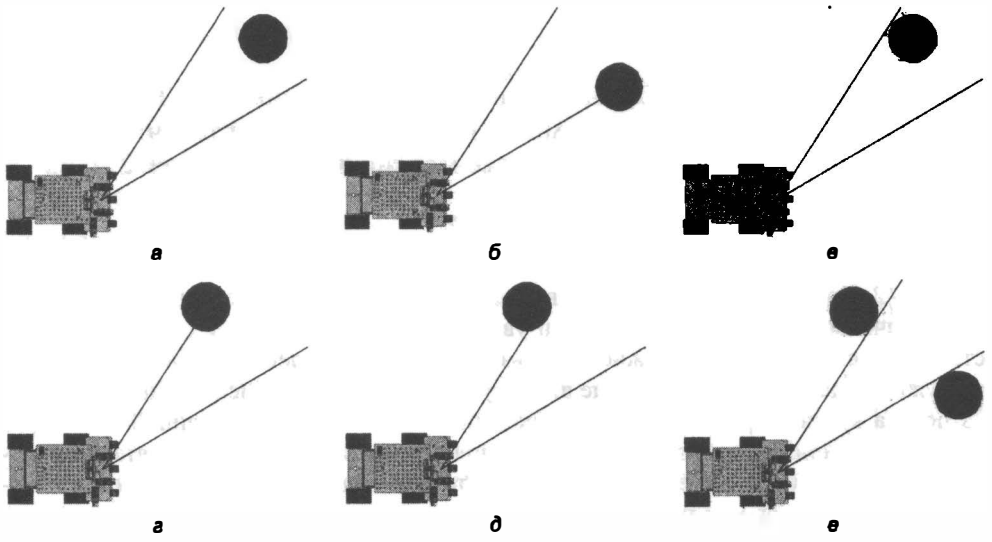


Рис. 15.6. Варианты обнаружения кегли сонаром

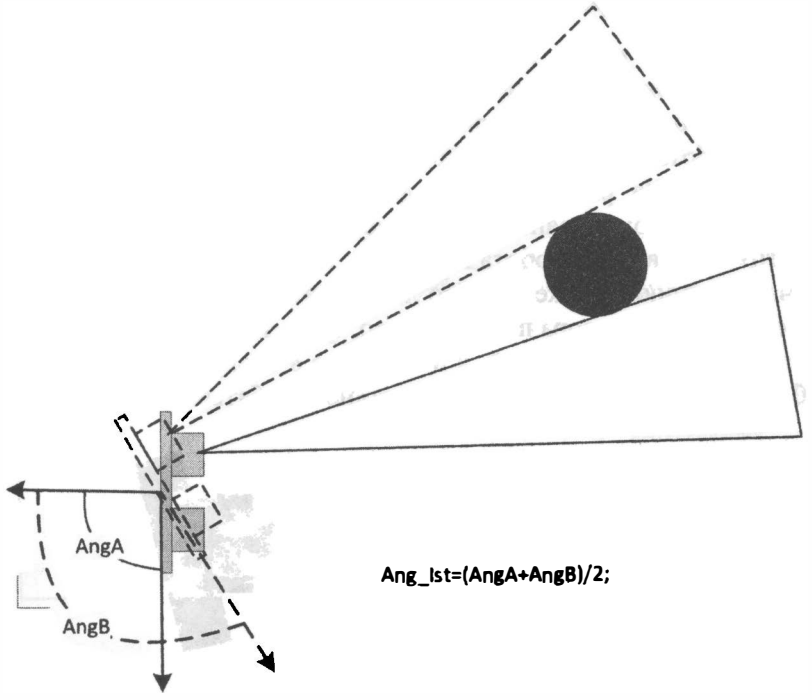


Рис. 15.7. Нахождение центра кегли

робот определяет правый край кегли, затем левый, а потом по несложной формуле находит середину. Впрочем, зная расстояние до кегли и ее ширину, можно ограничиться нахождением одного края цели, — например, правого.

Более подробно порядок поиска центра кегли рассмотрен в приведенном далее алгоритме и реализован в созданной по нему программе (см. *разд. «Программа»*).

Определение цвета кегли

Для определения цвета кегли удобно воспользоваться датчиком TCRT 5000. Его нужно установить в передней части бампера и направить вперед так, чтобы, подъезжая к кегле, робот имел расстояние между сенсорами датчика и кеглей 2–4 см.

Применим детектор препятствия?

Определять наличие кегли можно было бы детектором препятствия, использованным в *главе 12*. Принцип его действия позволяет ему реагировать на черные и белые кегли на различном расстоянии — белую он увидит значительно раньше (с большего расстояния), чем черную, что можно использовать при определении цвета кегли: роботу не нужно будет подъезжать к кегле вплотную, и он сможет на расстоянии (возможно, из центра круга) определять, какая перед ним кегля. Это в приведенной здесь программе не реализовано, но может быть использовано вами при самостоятельной доработке. В подобном случае детектор препятствия должен смотреть вперед и находиться по центру фронтальной плоскости робота.

Коррекция направления движения

Из-за неравномерного вращения колес, проскальзывания их на поверхности или торможения кеглей, робота может развернуть, и он после завершения обработки кегли не сможет вернуться в центр круга. Вернее, он не сможет вернуться в центр, если эту ситуацию не проработать. Помочь стабилизировать направление движения робота может черная линия. На рис. 15.8 приведены три возможных варианта наезда роботом на черную линию: вариант *б* не требует корректировки, но варианты *а* и *в* показывают, что направление робота искажено. Эти искажения покажут и датчики линии: в случае *а* линию покажет только левый датчик, в случае *в* — правый. Тогда в варианте *а* следует поворачивать налево, пока не сработает правый датчик или не перестанет срабатывать левый, а в варианте *в* следует поворачивать направо, пока не сработает левый датчик или не перестанет срабатывать правый.

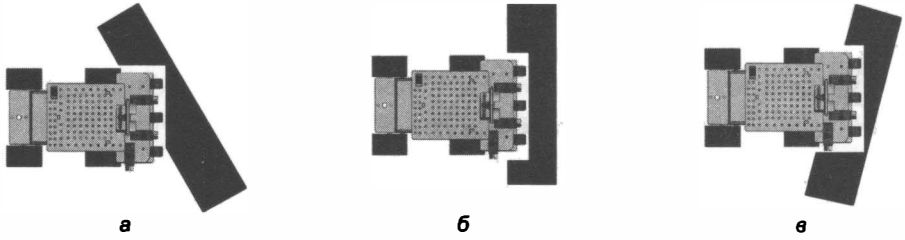


Рис. 15.8. Варианты наезда робота на черную линию

Модернизация робота с использованием гироскопа

Вернемся к началу главы и обратим внимание на следующие условия игры в кегельринг:

- кегли стоят по кругу в специальных местах;
- расстояние от кегли до центра окружности всегда одинаковое;
- угловое расстояние между кеглями одинаковое и составляет 45 градусов;
- кроме того, мы настроим датчик определения цвета кегли так, чтобы робот мог определять цвет кегли непосредственно из центра круга, не приближаясь к ней.

Эти условия позволяют нам использовать гироскоп на все 100 процентов. Применим описанные в предыдущей главе функции — для поворота на указанный угол: `Angle(float ang)` и для прямолинейного движения при выталкивании кегли: `forward_t(long microsec)`.

Уже было упомянуто, что инфракрасный детектор препятствий по-разному реагирует на белые и черные предметы (он лучше видит белые предметы), а если точно известно, что кегля на определенном расстоянии есть, то можно настроить реакцию датчика только на белые кегли, чтобы, находясь в центре круга, робот белые кегли

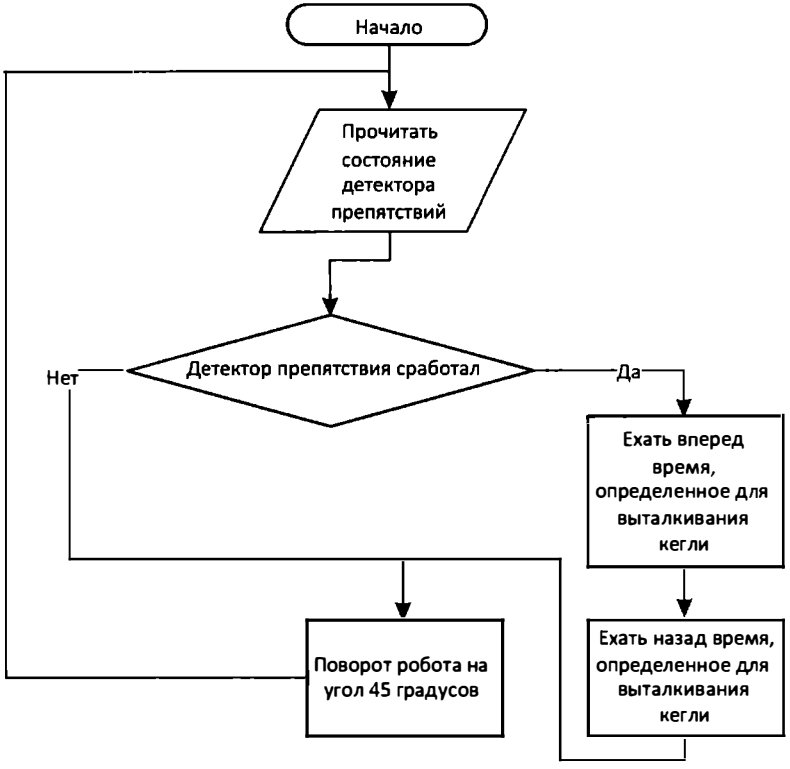


Рис. 15.9. Алгоритм выбивания белых кеглей: робот повернут точно на кеглю

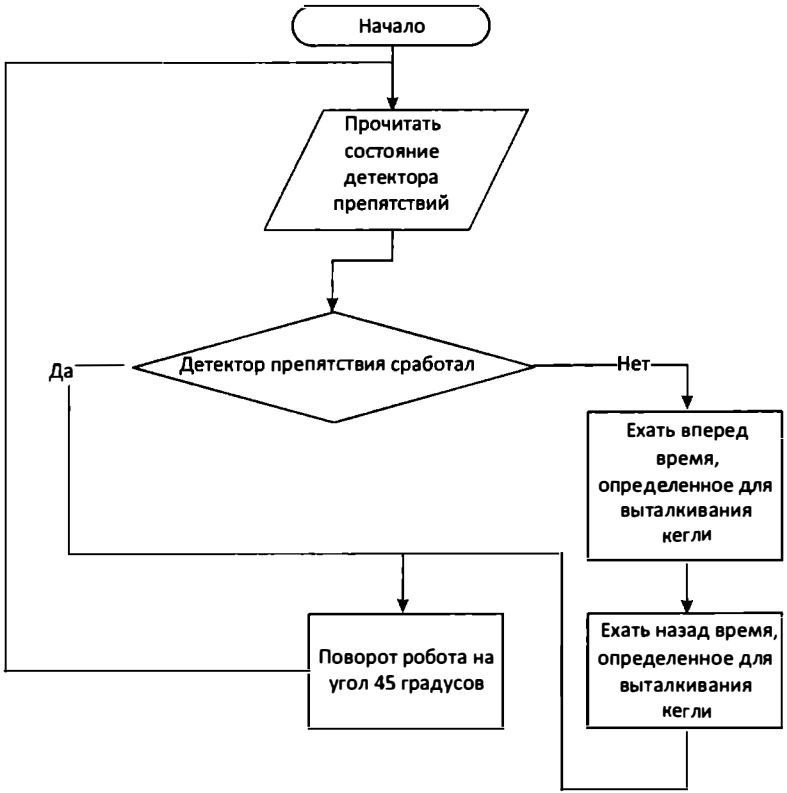


Рис. 15.10. Алгоритм выбивания черных кеглей: робот повернут точно на кеглю

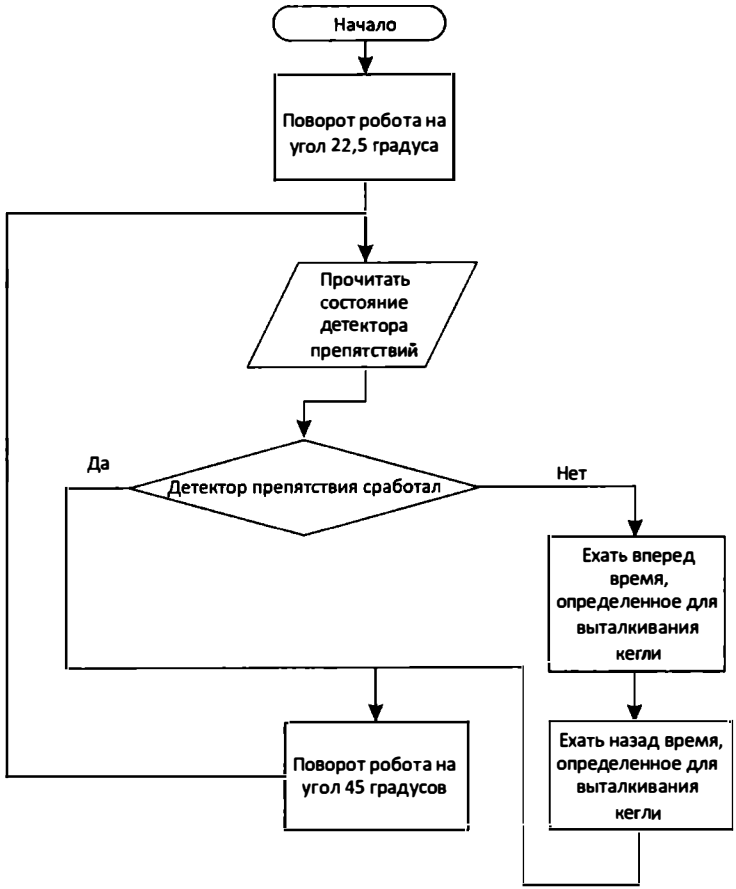


Рис. 15.11. Алгоритм выбивания черных кеглей: робот направлен между кеглями

уже видел, а черные еще нет. Этого достаточно, чтобы определить, какая перед роботом кегля — белая или черная. Теперь выясним, а нужно ли роботу иметь датчик черной линии? Если имеется гироскоп, то робота и так можно заставить двигаться прямо, а время движения по выбиванию кегли и возврату отрегулировать временными константами.

Теперь алгоритм стал максимально простым, мы учли условия игры и настроили датчик цвета кегли.

При проведении соревнований робота ставят в центр круга и направляют либо на кеглю, либо между кеглями, — оба варианта следует учесть при работе с гироскопом.

Установка датчиков

Для реализации подготовленных алгоритмов потребуется по центру бампера робота установить только один датчик — инфракрасный детектор препятствия. Для того чтобы он надежнее работал и не реагировал на помехи, можно фототранзистор (он черного цвета) поместить в черную трубочку — например, колпачок от фломастера или шариковой ручки (рис. 15.12).

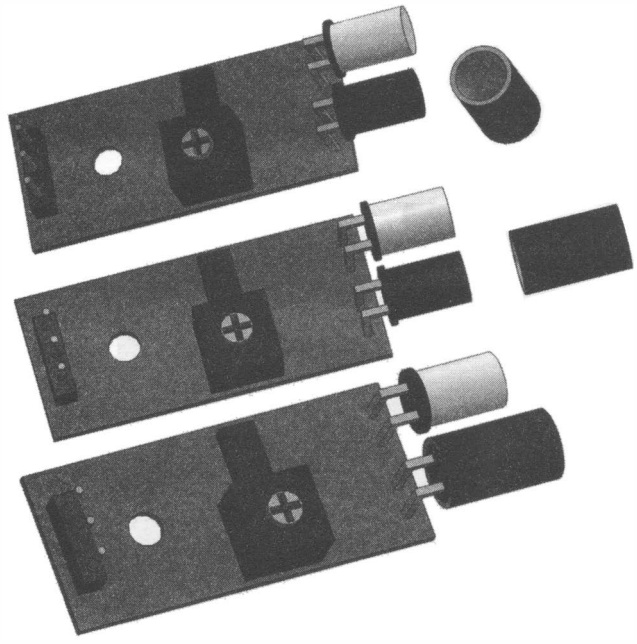


Рис. 15.12. Модернизация детектора препятствия

В *главе 12* датчики использовались для прохождения лабиринта. Сейчас нам потребуется только один из них (рис. 15.13) — пусть это будет датчик, установленный в 8-й порт, тем более, что порт 7 уже используется IR-приемником (см. *главу 8*). Закрепим датчик по центру, как показано на рис. 15.14. Используйте для этого двусторонний скотч или термоклей.

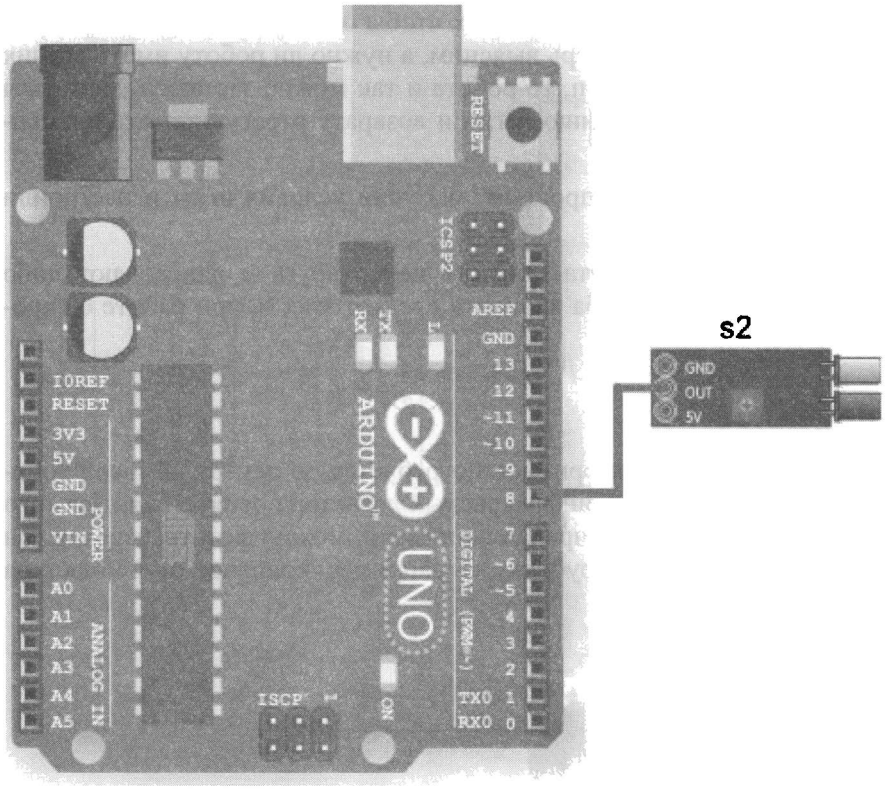


Рис. 15.13. Детектор препятствия в роли датчика определения цвета кегли

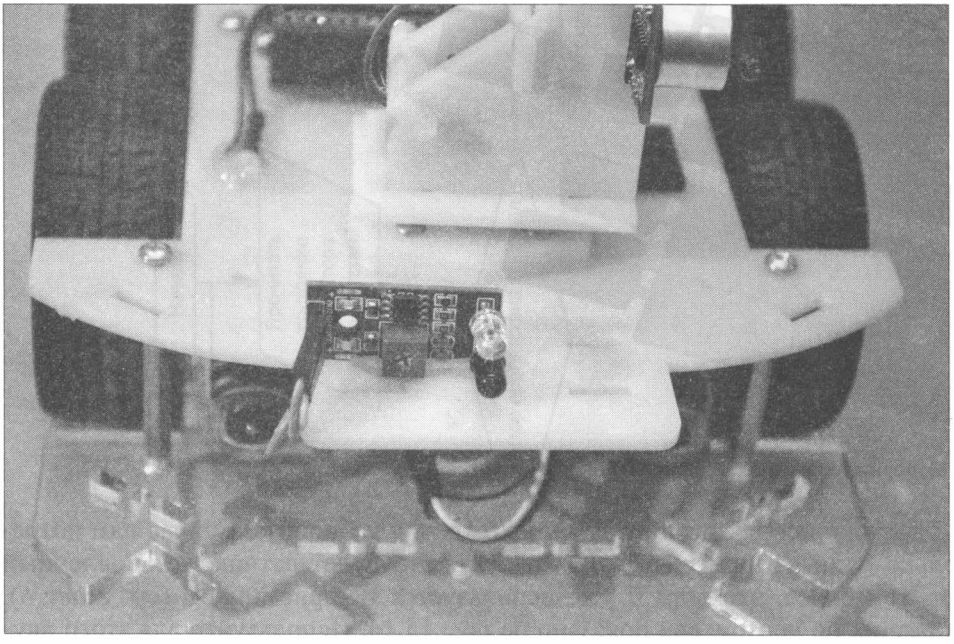


Рис. 15.14. Робот с установленным детектором препятствий для определения цвета кегли

Программа

Создадим программу, реализующую алгоритм, изображенный на рис. 15.9 (робот повернут точно на кеглю). Для запуска робота используется IR-пульт (кнопка «вперед»), а остановить программу можно кнопкой «стоп». Датчик цвета кегли установлен на 8-й порт/пин Arduino. Следует учесть, что, срабатывая, детектор препятствия сбрасывается в ноль, иначе на его выходе будет высокое напряжение. Вам экспериментально потребуется определить и поменять время прохода на выбивание кегли (в программе это 300 миллисекунд). Все остальное довольно просто — робот поворачивает на 45 градусов и проверяет состояние детектора препятствия, если детектор сработал, считает, что кегля белая, и выталкивает ее за ринг, затем возвращается в центр.

Также потребуется настроить детектор препятствия (используя переменный резистор на корпусе) так, чтобы он на заданном расстоянии 10–15 см срабатывал на белые кегли, а черные еще не видел. Это процесс, требующий точности и терпения, не торопитесь!

> ***Солнечное освещение сбивает робота с толку***
>
> При наличии солнечного освещения робот не сможет должным образом идентифицировать кегли, т. к. датчик станет реагировать на стороннее инфракрасное излучение.

Листинг 15.2. Программа выбивания белых кеглей

```
//Подключение библиотеки I2C шины.
#include <Wire.h>
//Подключение библиотеки удаленного управления.
#include "IRremote.h"
//Подключение библиотеки управления моторами.
#include "motor.h"
//Подключение библиотеки управления гироскопом.
#include "gyro_acsel.h"
// Порт для IR-приемника.
int RECV_PIN = 7;
// Создание IR-приемника.
IRrecv irrecv(RECV_PIN);
//Переменная для результатов IR-приемника.
decode_results results;
int diod_pin = 6;
int keg_colour = 8;
//==========================================
void setup()
{
  //запуск гироскопа.
  giroscop_setup();
  // Переменные - номера контактов (пинов) Arduino.
```

```
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  _stop(); //Двигатели остановлены.
  // Запуск IR-приемника.
  //Расчет компенсации утечки нуля гироскопа.
  Calc_CompensatorZ(3000);
  //переводим пин светодиода в режим вывода.
  pinMode(diod_pin, OUTPUT);
  // Пин датчика цвета кегли в режим ввода.
  pinMode(keg_colour, INPUT);
  //Serial.begin(9600);
  digitalWrite(diod_pin, 1);
  delay(500);
  digitalWrite(diod_pin, 0);
  irrecv.enableIRIn();
}
// Основная программа.
void loop()
{
  bool start = false;
  while (!start)
  {
    if (irrecv.decode(&results))
    {
      //Если нажата кнопка "вперед" - стартуем
      if (results.value == 0x511DBB)
      {
        start = true;
        digitalWrite(diod_pin, 1);
      }
      irrecv.resume();
    }
  }
  while (start)
  {
    //Если сработал датчик
    // Датчик, когда срабатывает - сбрасывается в ноль,
    // иначе он установлен в единицу.
    if (!digitalRead(keg_colour)) //есть белая кегля
    {
      //Время движения к кегле и
      //от кегли определяется экспериментом.
      forward_t(300000); // в мик.сек.
      _stop();
      backward_t(300000);
      _stop();
    }
    Angle(45);
    if (irrecv.decode(&results))
```

```
    {
      //Если нажата кнопка "стоп" - останавливаемся
      if (results.value == 0xD7E84B1B)
      {
        start = false;
        digitalWrite(diod_pin, 0);
      }
      irrecv.resume();
    }
  }
}
```

Значительно улучшить работу датчика цвета можно, собрав его самостоятельно по описанной далее схеме (рис. 15.15). Датчик компонуется из двух частей: управляе-

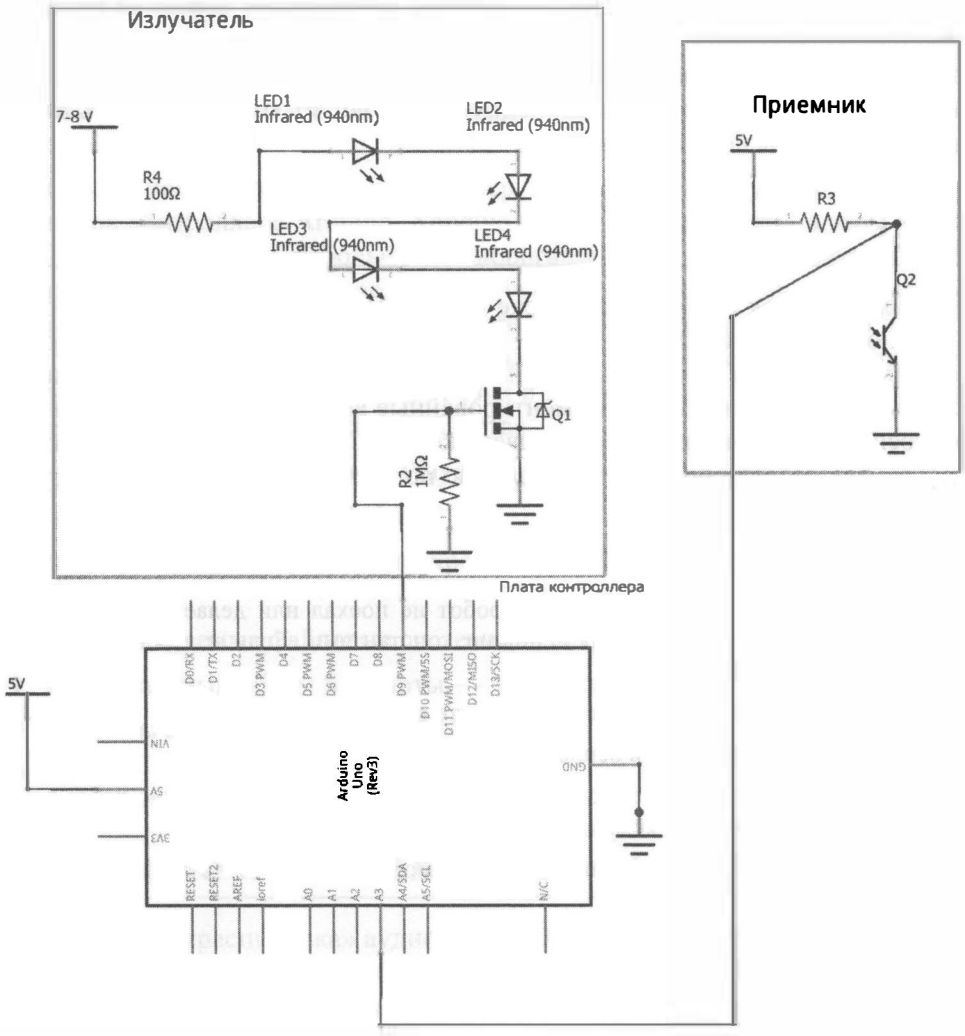


Рис. 15.15. Схема подключения самостоятельно изготовленного датчика цвета к Arduino

мого излучателя, собранного из нескольких (в данном случае — четырех) инфракрасных светодиодов, и приемника, состоящего из инфракрасного фототранзистора в паре с резистором 10 кОм. Подача сигнала на 9-й контакт регулирует поток инфракрасного излучения, а приемник регистрирует его отражение от объекта. Фактически подобный датчик может определить три состояния:

1. **Объекта нет** — отражение минимальное, на аналоговом входе A3 максимальное значение напряжения (фототранзистор Q2 закрыт, и на нем падает все напряжение).
2. **Объект темный** — отражение среднее, на аналоговом входе A3 значение напряжения ниже максимального (фототранзистор Q2 приоткрыт, и часть напряжения падает на резисторе R3).
3. **Объект светлый** — отражение хорошее, на аналоговом входе A3 значение напряжения низкое (фототранзистор Q2 открыт, напряжения падает на резисторе R3).

За счет наличия ключа (полевой транзистор Q1) возможно измерять два состояния: при включенных и выключенных светодиодах, что позволит компенсировать влияние внешнего инфракрасного излучения.

Выводы

Мы определили порядок обхода при поиске кеглей. Подобрали датчики для обнаружения черной ограничительной линии, использовали гироскоп для поворота к кегле на заданный угол и для сохранения направления прямолинейного движения. Также были использованы недокументированные возможности инфракрасного детектора препятствий при определении цвета кегли. Создали алгоритмы и разработали программу, которая позволяет выборочно по цвету выбивать кегли из круга. Цель главы достигнута.

Поиск ошибок

Если после сборки и программирования робот не поехал или делает все шиворот-навыворот, стоит «поиграть» с временны́ми константами. Если и это не помогло, следует искать ошибки пошагово:

1. Подключите робота к компьютеру и снимите колеса.
2. Закомментируйте весь код в основной программе.
3. Подключая блоки кода по одному, выводите служебную информацию через порт на компьютер, используя команды `Serial.print()` и `Serial.println()`.
4. Сверяйте полученную информацию с ожидаемой.

В следующем проекте мы научим робота говорить.

ГЛАВА 16
ГОВОРЯЩИЙ РОБОТ

Попытаемся «разговорить» нашего робота. Конечно, он не сможет слушать аудиокоманды, но сообщать о наступлении определенного события при помощи разговорной речи сможет.

Для этого нам потребуется аудиоплеер с возможностью управления при помощи сигналов контроллера Arduino. Хорошо зарекомендовала себя в этом качестве плата DFPlayer Mini — небольшая недорогая плата, которую можно подключить к большинству контроллеров Arduino через программно-эмулируемый последовательный интерфейс, подобный тому, что был задействован в *главе 8* для подключения Bluetooth-модуля удаленного управления. В качестве аудиосообщений используются аудиофайлы формата MP3, звук в которых сжат при помощи специального алгоритма с небольшой потерей качества.

Подробные характеристики DFPlayer Mini:

- напряжение питания 3,3–5 В;
- поддерживаемые частоты дискретизации: 8, 11,025, 12, 16, 22,05, 24, 32, 44,1, 48 кГц;
- поддерживаемые файловые системы: FAT16, FAT32;
- максимальный объем SD-карты — 32 Гбайт;
- форматы аудиофайлов: MP3, WAV, WMA;
- количество уровней громкости — 30.

Создание и монтаж аудиосистемы робота

Общий вид и «распиновка» аудиоплеера DFPlayer Mini приведены на рис. 16.1. Для подключения к роботу потребуются два контакта: TX и RX (последовательный интерфейс). Большинство незадействованных контактов служат для управления плеером при помощи других интерфейсов (табл. 16.1).

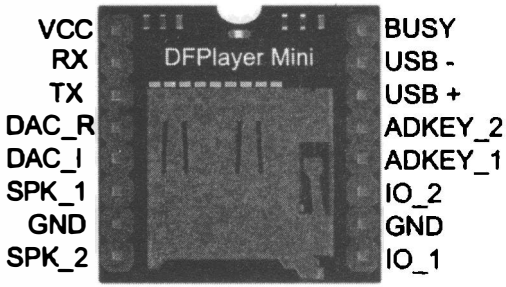


Рис. 16.1. Аудиоплеер DFPlayer Mini

Таблица 16.1. *Описание контактов DFPlayer Mini*

Вывод	**Описание**
VCC	Питание «+»
GND	Питание «–»
RX	UART-прием
TX	UART-передача
SPK1	Громкоговоритель «+»
SPK2	Громкоговоритель «–»
BUSY	Индикатор состояния («0» — простой, «1» — проигрывание)
DAC_R	Выход на наушник или усилитель (канал «R»)
DAC_L	Выход на наушник или усилитель (канал «L»)
IO1	Вход управления: короткое нажатие — «назад», длинное — уменьшить громкость
IO2	Вход управления: короткое нажатие — «вперед», длинное — увеличить громкость
ADKEY1	Порт для подключения резистивной клавиатуры, вход 1
ADKEY2	Порт для подключения резистивной клавиатуры, вход 2
USB+	USB-порт, вывод «+»
USB–	USB-порт, вывод «–»

В пользу DFPlayer Mini говорит также и наличие у него встроенного аудиоусилителя, что дает возможность подключить к модулю динамик мощностью до 3 Вт. Динамик подключается на контакты SPK_1 и SPK_2.

Электропитание подается на контакты VCC и GND — от положительного и отрицательного контактов источника питания соответственно. Модуль может работать при напряжении питания от 3,3 до 5 В. Для устранения просадок напряжения питания рекомендуется использовать маломощный динамик на 0,25 Вт или, если вы со-

бираетесь использовать мощный динамик, запитывать модуль от отдельного стабилизатора электрического питания на 3,3–5 В.

Для подключения DFPlayer Mini к Arduino могут потребоваться дополнительные компоненты в связи с большим током, потребляемым платой в режиме разговора, и необходимостью защиты информационных выходов от перегрузок (логика платы рассчитана на 3,3 вольта).

Примерный перечень необходимых компонентов приведен на рис. 16.2: провода с клеммами «мама-мама» (1), провода с клеммами «мама-папа» (2), электролити-

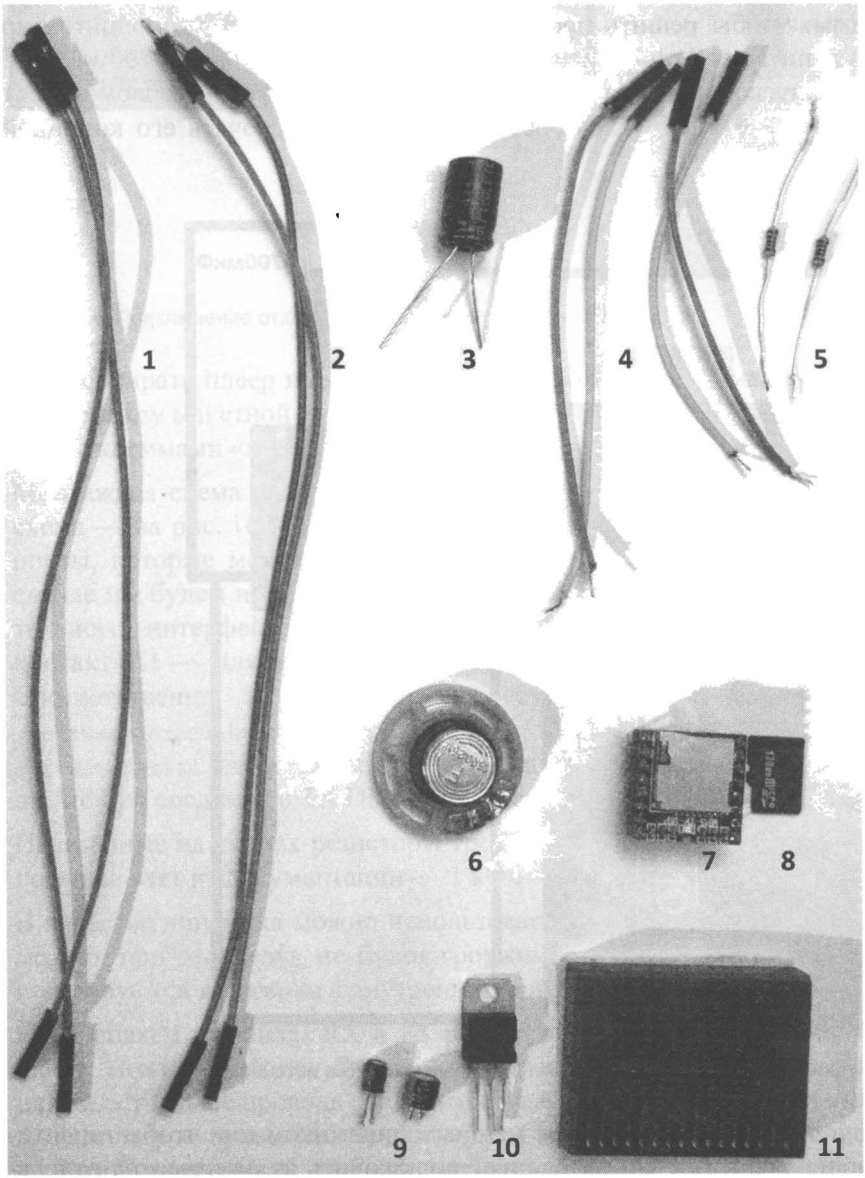


Рис. 16.2. Набор компонентов для подключения DFPlayer Mini к Arduino

ческий конденсатор 4700 мкФ/6В (3), два разрезанных провода с клеммами «мама-мама» (4), два резистора на 1 кОм (5), динамик 16 Ом, 0,25 Вт (6), сама плата плеера (7), карта microSD (8), конденсаторы внешнего стабилизатора питания (9), дополнительный стабилизатор питания на 5 вольт (10), макетная плата (11).

Для электропитания аудиоплеера можно применить стабилизатор питания на 5 вольт от драйвера моторов, но при одновременном использовании сервомотора sg90 и аудиоплеера стабилизатор перестает справляться с нагрузкой, и напряжение на нем падает. Если при этом от стабилизатора запитана еще и плата Arduino Uno (через Arduino Sensor Shield v5.0), то начинают происходить сбои в работе загруженной программы. Чтобы решить проблему, можно установить (дополнительно поджать) в клеммы питания Arduino Sensor Shield v5.0 конденсатор на 4700 мкФ/6V (рис. 16.3), перед запуском речи на плеере программно отключать сервомотор от управления командой `dettach`, а после остановки речи подключать его командой `attach`.

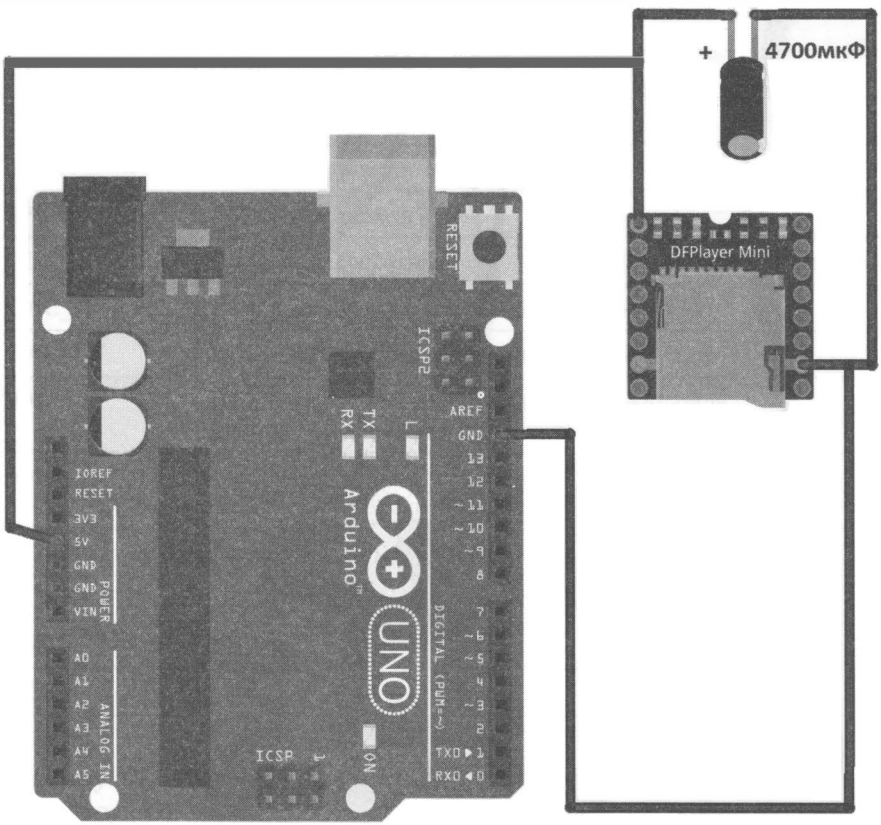


Рис. 16.3. Электропитание аудиоплеера DFPlayer Mini и Arduino от одного источника

Если же требуется воспользоваться более мощным динамиком или чтобы гарантированно обезопасить робота от сбоев, следует подключить аудиоплеер на отдельный стабилизатор питания — например, на микросхеме L7805CV (рис. 16.4).

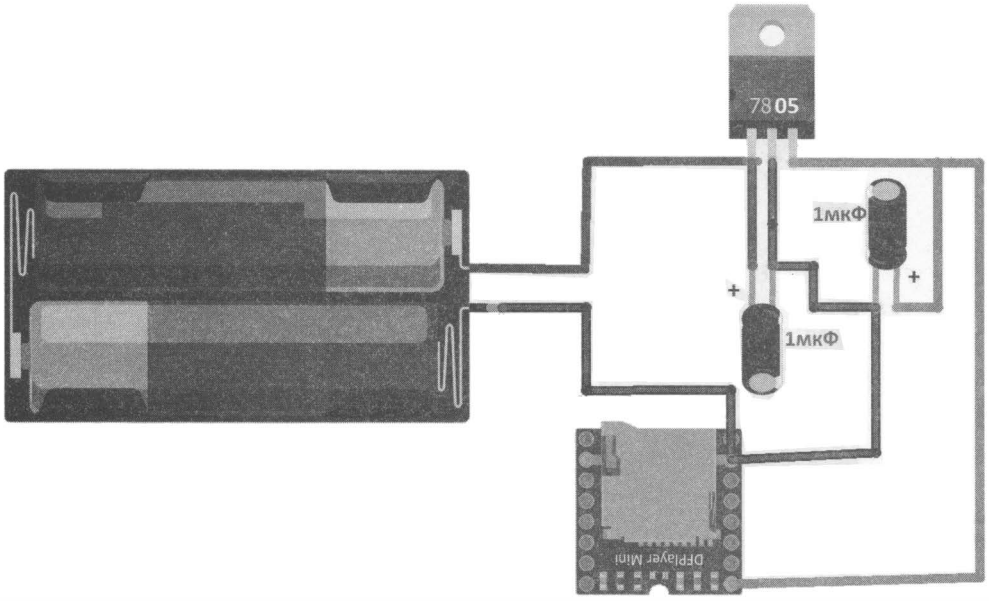


Рис. 16.4. Подключение отдельного стабилизатора для DFPlayer Mini

Если собирать плеер на макетной плате, провода типа «мама-мама» не потребуются — между макетной платой и Arduino Sensor Shield v5.0 будут использованы провода с клеммами «мама-папа».

Монтажная схема подключения плеера приведена на рис. 16.5, а принципиальная схема — на рис. 16.6. Мы задействуем контакты Arduino A1 (15) и A2 (16) — это порты, которые могут работать в качестве и аналоговых, и цифровых. В нашем случае мы будем использовать их в качестве цифровых для организации последовательного интерфейса обмена информацией между Arduino и DFPlayer Mini: контакт A1 — для ввода информации (RX), а контакт A2 — для вывода (TX). Соответственно, в программе потребуется подключить библиотеку `#include <SoftwareSerial.h>`, а при создании последовательного соединения указать: `SoftwareSerial DFPSerial(15, 16)`. Соединение должно быть перекрестным — RX от Arduino соединен с TX DFPlayer, и наоборот.

Показанные на схемах резисторы требуются для согласования уровня сигналов, их номинал взят из документации — 1 кОм.

В качестве динамика можно использовать динамик от старых наушников. Но, возможно, при этом звук не будет громким, потому что в наушниках, как правило, используются динамики с внутренним сопротивлением 16 Ом.

Итак, впаяем в провода RX и TX резисторы на 1 кОм (если использовать макетную плату, можно обойтись без пайки), для соединения динамика также потребуется пайка, остальные провода соединим согласно рассмотренным и выбранным схемам. Аудиоплеер с проводами и динамиком для подключения к Arduino Sensor Shield v5.0 изображен на рис. 16.7. Желательно установить аудиоплеер на робота так, чтобы ничего не мешало вынимать карту памяти, — для него есть место в задней части

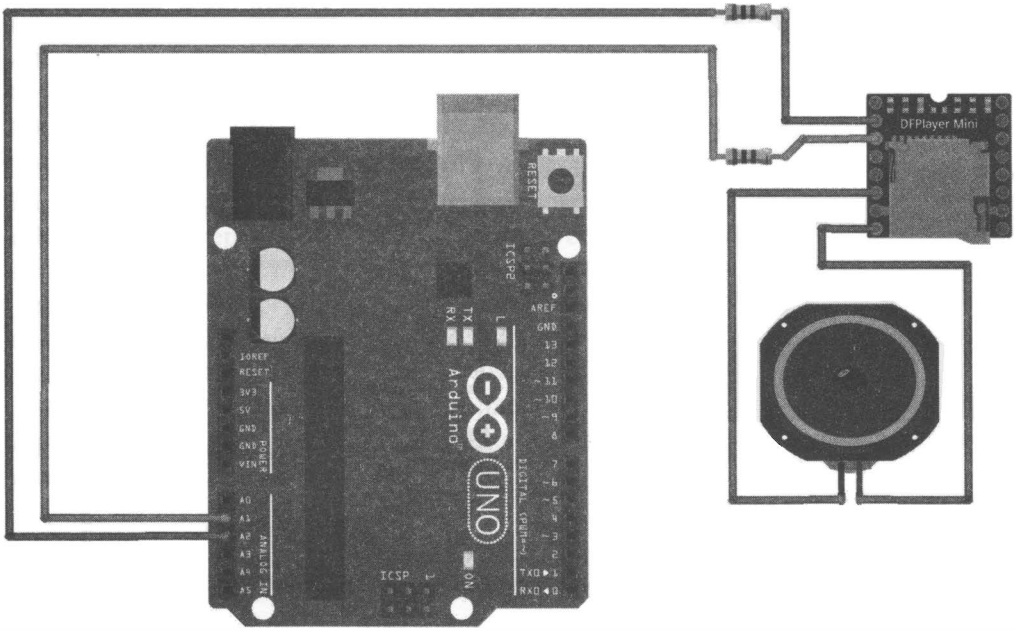


Рис. 16.5. Порядок соединения DFPlayer Mini и Arduino

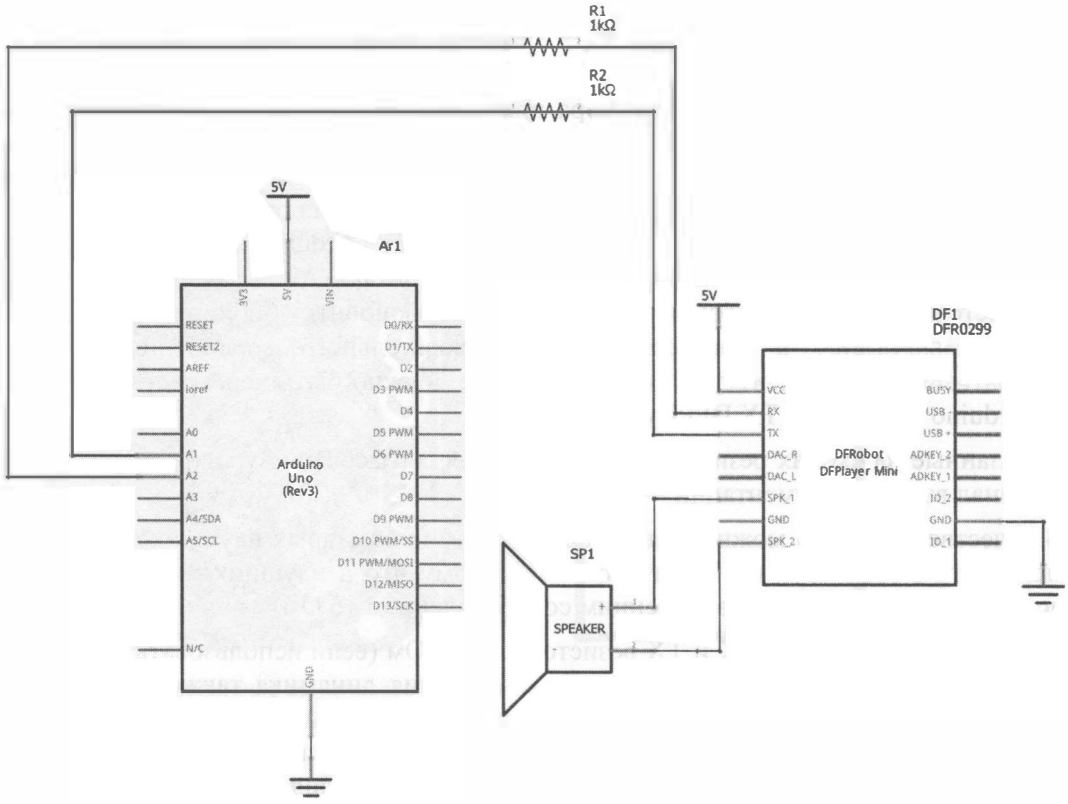


Рис. 16.6. Принципиальная схема соединения DFPlayer Mini и Arduino UNO

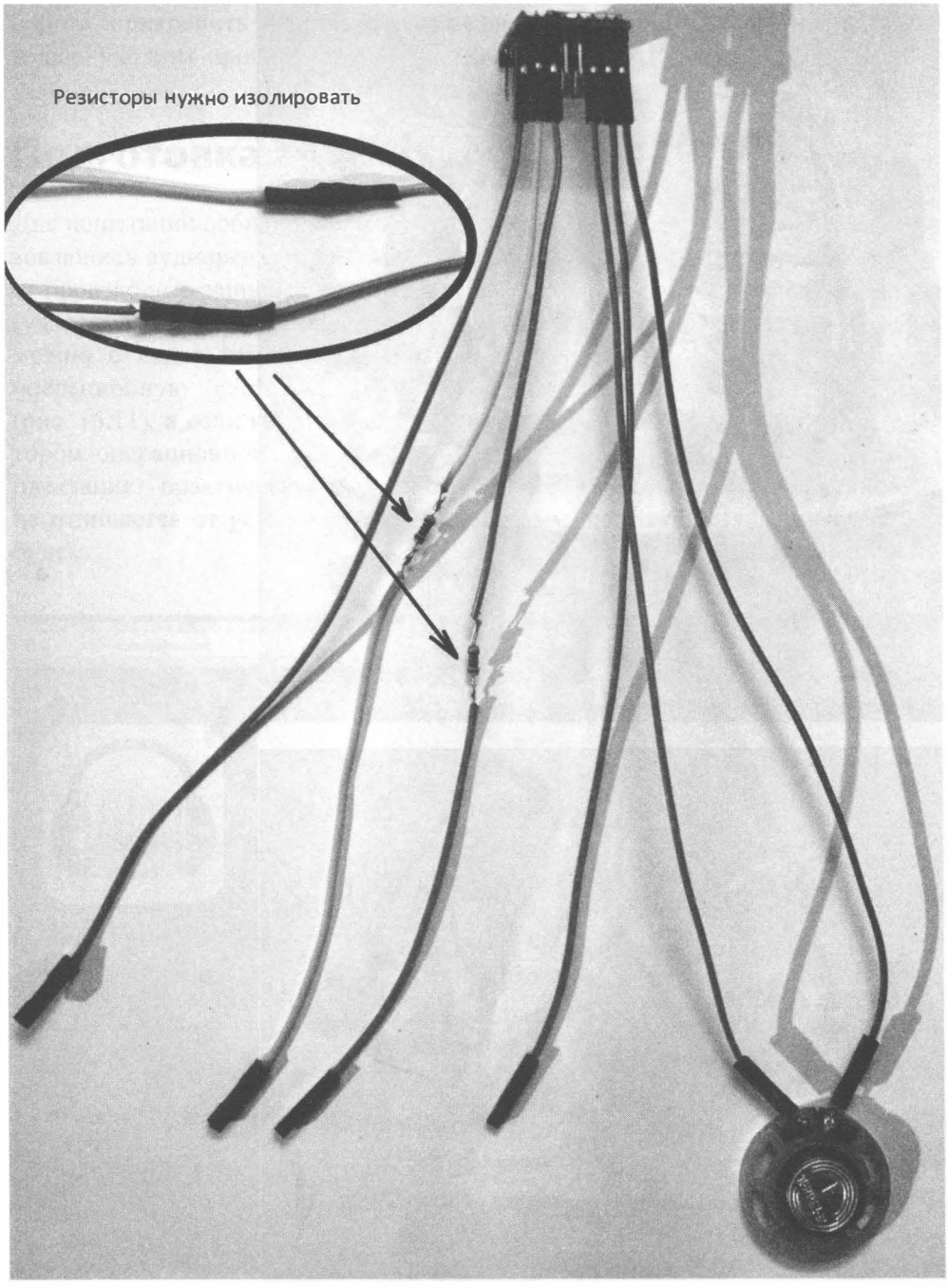


Рис. 16.7. DFPlayer Mini с проводами и динамиком для подключения к Arduino Sensor Shield v5.0

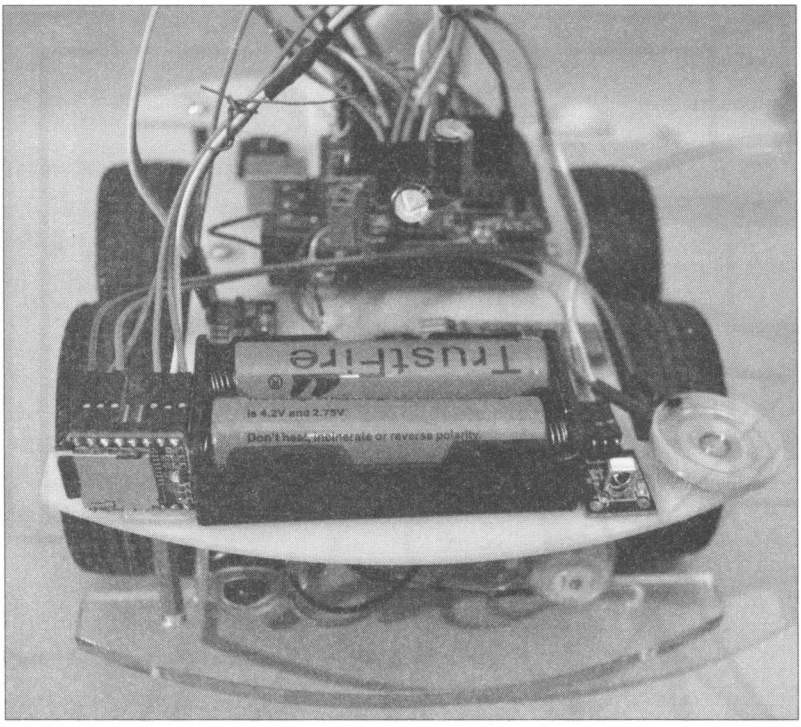


Рис. 16.8. Робот с установленным DFPlayer Mini

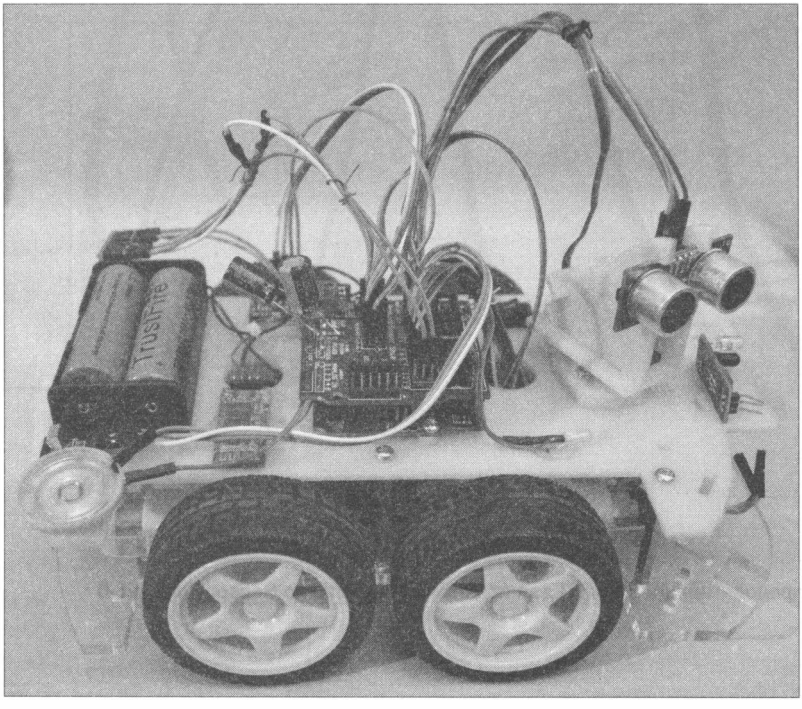


Рис. 16.9. Робот в сборе с установленным DFPlayer Mini и динамиком 16 Ом, 0,25 Вт

робота, прикрепить его допускается на двухсторонний скотч (рис. 16.8). Осталось только уложить провода (рис. 16.9), и можно приступать к испытаниям.

Подготовка аудиосообщений

Для испытаний собранной схемы подготовим несколько аудиосообщений, воспользовавшись аудиоредактором Audacity. Это бесплатный редактор, который позволяет производить запись аудио с различных устройств и последующую его обработку (удаление шума, обрезка, изменение тональности). Установить аудиоредактор можно с сайта **http://www.audacityteam.org** (рис. 16.10). Если вы используете операционную систему Windows, то скопируйте соответствующую версию (рис. 16.11), а если вы работаете в Linux, то воспользуйтесь внутренним инсталлятором операционной системы (как правило, Audacity входит в стандартную комплектацию практически всех дистрибутивов Linux). Процесс установки ничем не отличается от установки других программ, поэтому останавливаться на нем не будем.

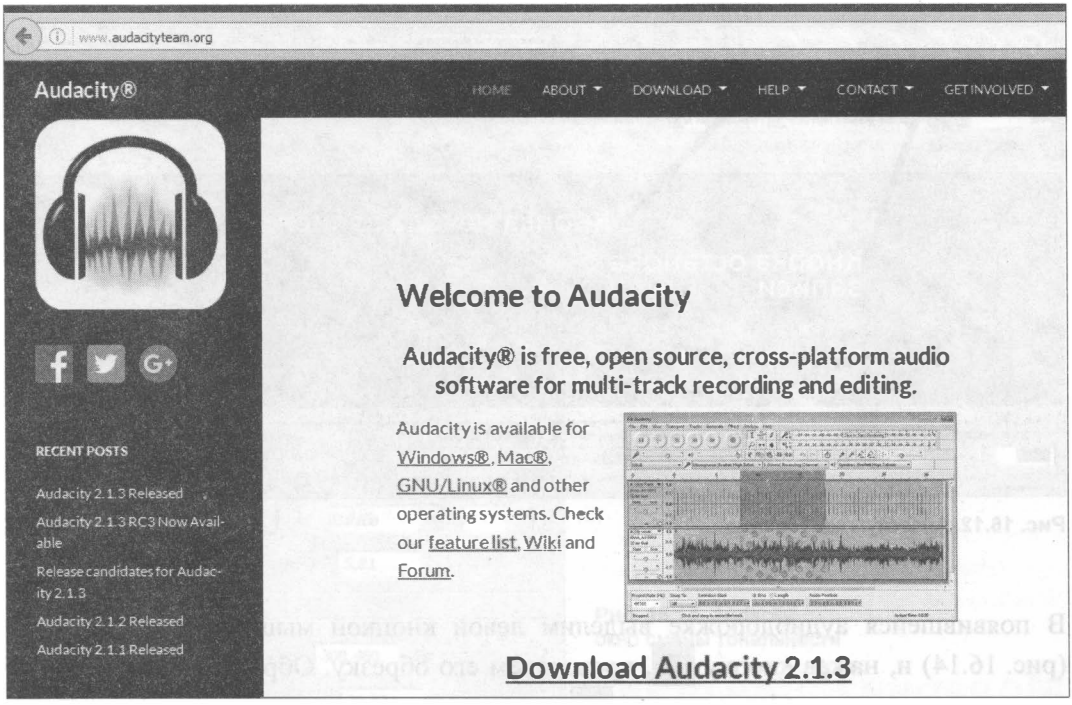


Рис. 16.10. Официальный сайт Audacity

Подключим к компьютеру микрофон и немного поработаем в Audacity: запустим редактор, нажмем кнопку записи (рис. 16.12) и продиктуем в микрофон подготовленную для робота фразу (рис. 16.13), затем нажмем кнопку останова записи.

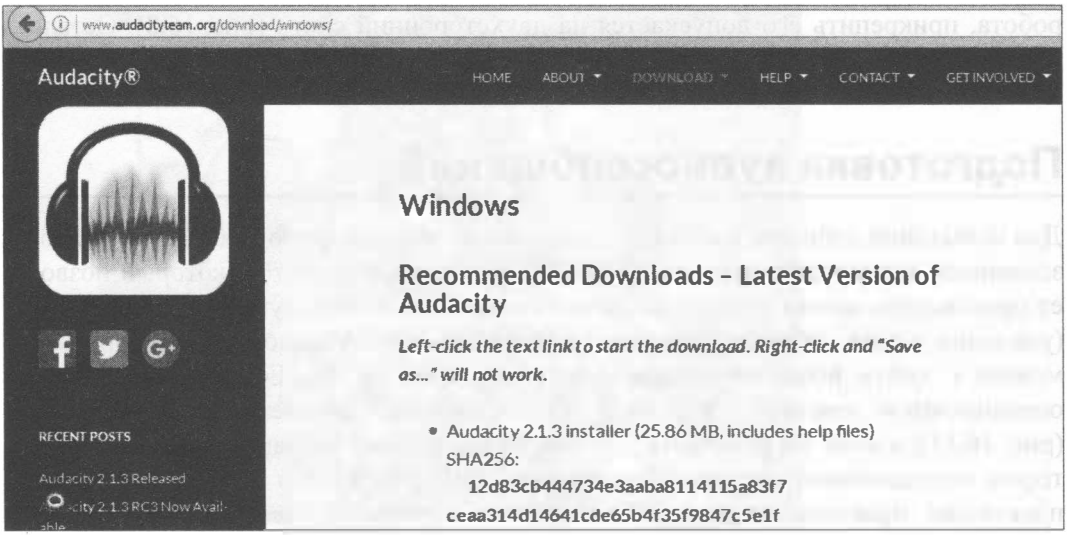


Рис. 16.11. Скачивание Audacity для Windows

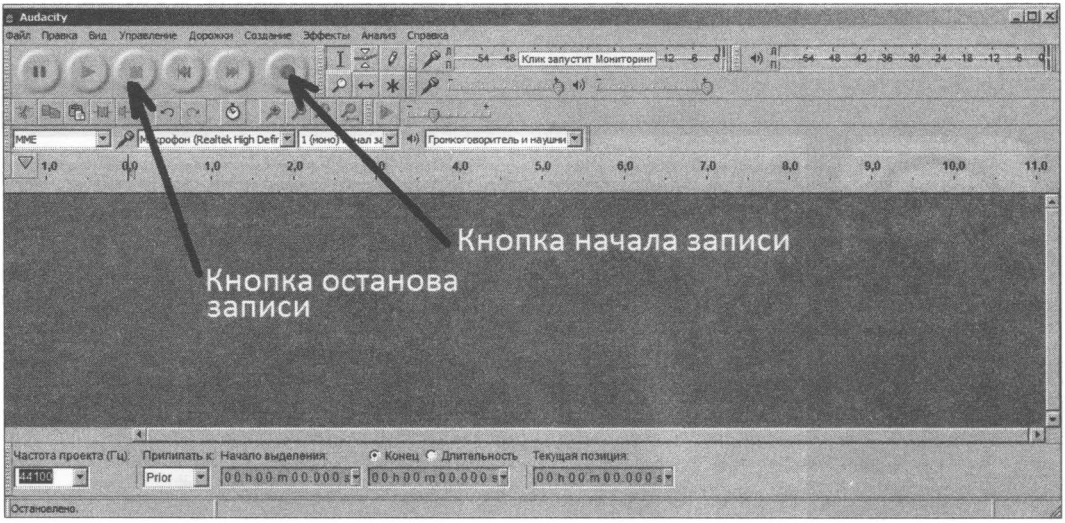


Рис. 16.12. Audacity: главное окно

В появившейся аудиодорожке выделим левой кнопкой мыши нужный участок (рис. 16.14) и, нажав кнопку  , произведем его обрезку. Образовавшиеся пустые участки можно удалять ножницами 

Для придания аудиосообщениям оригинального звучания можно изменить высоту тона. Для этого выделяем нужный участок дорожки, переходим из главного меню в меню **Эффекты** и выбираем пункт **Смена высоты тона**. Увеличим высоту тона — на рис. 16.15 увеличение высоты тона составляет 39,891 процент.

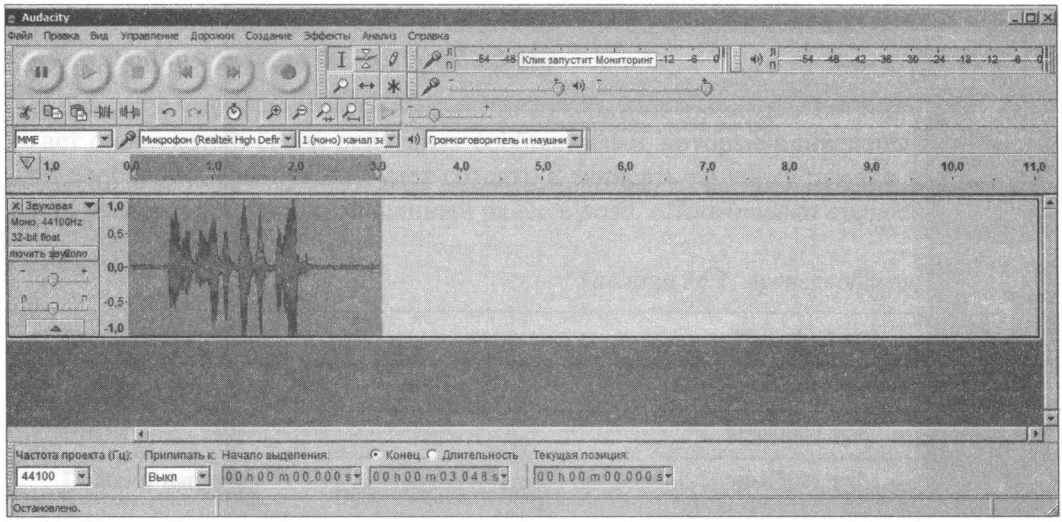


Рис. 16.13. Audacity: записанная аудиодорожка

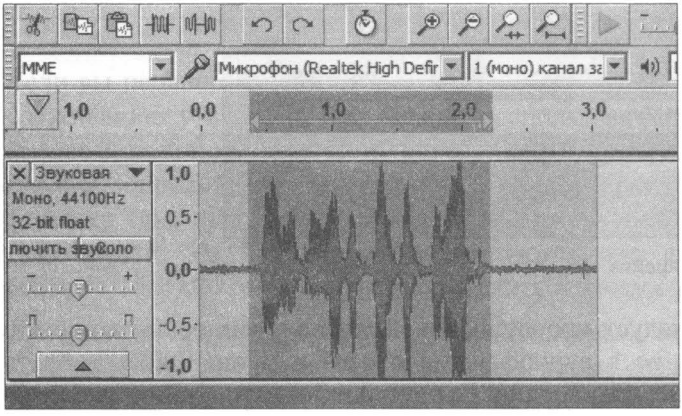


Рис. 16.14. Audacity: выделение участка аудиодорожки для последующей обрезки

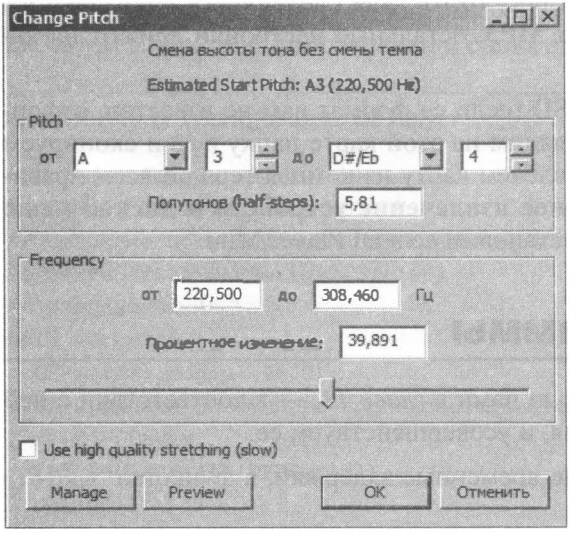


Рис. 16.15. Audacity: окно смены тональности

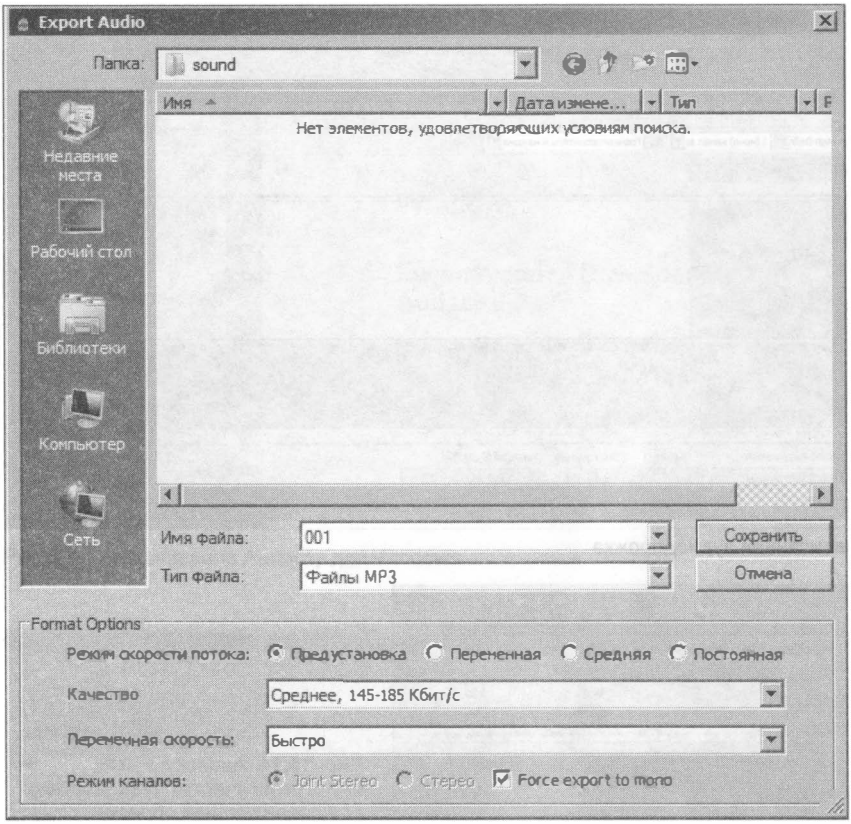


Рис. 16.16. Audacity: сохранение аудиофайла

Полученную аудиодорожку следует прослушать и, если полученное звучание вас удовлетворит, экспортировать ее в аудиофайл, для чего в меню **Файл** выбрать пункт **Экспорт аудио**. В окне сохранения **Export Audio** дать файлу числовое имя — например, 001 (как показано на рис. 16.16), для следующего сохраняемого файла — 002 и т. д. Тип файла **Файлы MP3**, остальные настройки влияют на качество записи.

Подключим к компьютеру карту microSD (если ее формат вам не известен, отформатируйте ее в FAT16 или FAT32), создадим на этой карте папку mp3 и скопируем в нее созданные ранее аудиофайлы. Извлечем карту из компьютера по всем правилам: сначала используя пункт **Безопасное извлечение устройств и дисков** меню Windows, а только затем физически, и установим ее в DFPlayer Mini.

Модернизация программы

Возьмем за основу программу, созданную нами в *главе 11*, — в соответствии с ней робот путешествует, обходя препятствия, и усовершенствуем ее:

- для точных поворотов используем не временные задержки, а функцию `Angle()` из библиотеки `gyro_acsel.h`.;

- пусть робот оповещает о своих действиях голосом при помощи только что рассмотренного модуля DFPlayer Mini.

Алгоритм, по которому будет работать программа, приведен на рис. 16.17. Робот будет оповещать о своих действиях сообщениями, которые приведены в табл. 16.2. Эти сообщения вам нужно будет создать и записать на карту памяти для DFPlayer Mini, используя порядок, описанный ранее в *разд. «Подготовка аудиосообщений»*.

Таблица 16.2. *Аудиосообщения для робота*

Сообщение	Имя файла
Анализ обстановки	mp3/0001.mp3
Вперед	mp3/0002.mp3
Поворот направо	mp3/0003.mp3
Поворот налево	mp3/0004.mp3
Разворот	mp3/0005.mp3

Если вы недавно начали работать с DFPlayer Mini, и своей библиотеки по его обслуживанию у вас еще нет, можно воспользоваться библиотекой по управлению плеером, взяв по следующему адресу: **https://github.com/DFRobot/DFPlayer-Mini-mp3**. В сопровождающем книгу электронном архиве, размещенном на сайте издательства (см. *приложение 2*), эта библиотека находится в папке с создаваемой программой. Из архива нам потребуются два файла: DFPlayer_Mini_Mp3.cpp и DFPlayer_Mini_Mp3.h — их нужно поместить в папку с рабочей программой.

Для начала протестируем работу аудиоплеера с помощью тестовой программы, приведенной в листинге 16.1. В этой программе создается последовательное соединение на контактах A1(15) и A2(16), инициализируется DFPlayer, устанавливается средняя громкость, а потом проигрываются пять файлов (треков) сначала по очереди, затем в обратном направлении и снова вперед до пятого трека.

Листинг 16.1. Тестирование DFPlayer

```
#include <SoftwareSerial.h>
#include "DFPlayer_Mini_Mp3.h"
// Создание последовательного соединения.
SoftwareSerial DFPSerial(15, 16);
// Инициализация.
void setup () {
  DFPSerial.begin (9600);
  // Привязка созданного последовательного соединения
  // к DFPayer.
  mp3_set_serial (DFPSerial);
  delay(1);  //
```

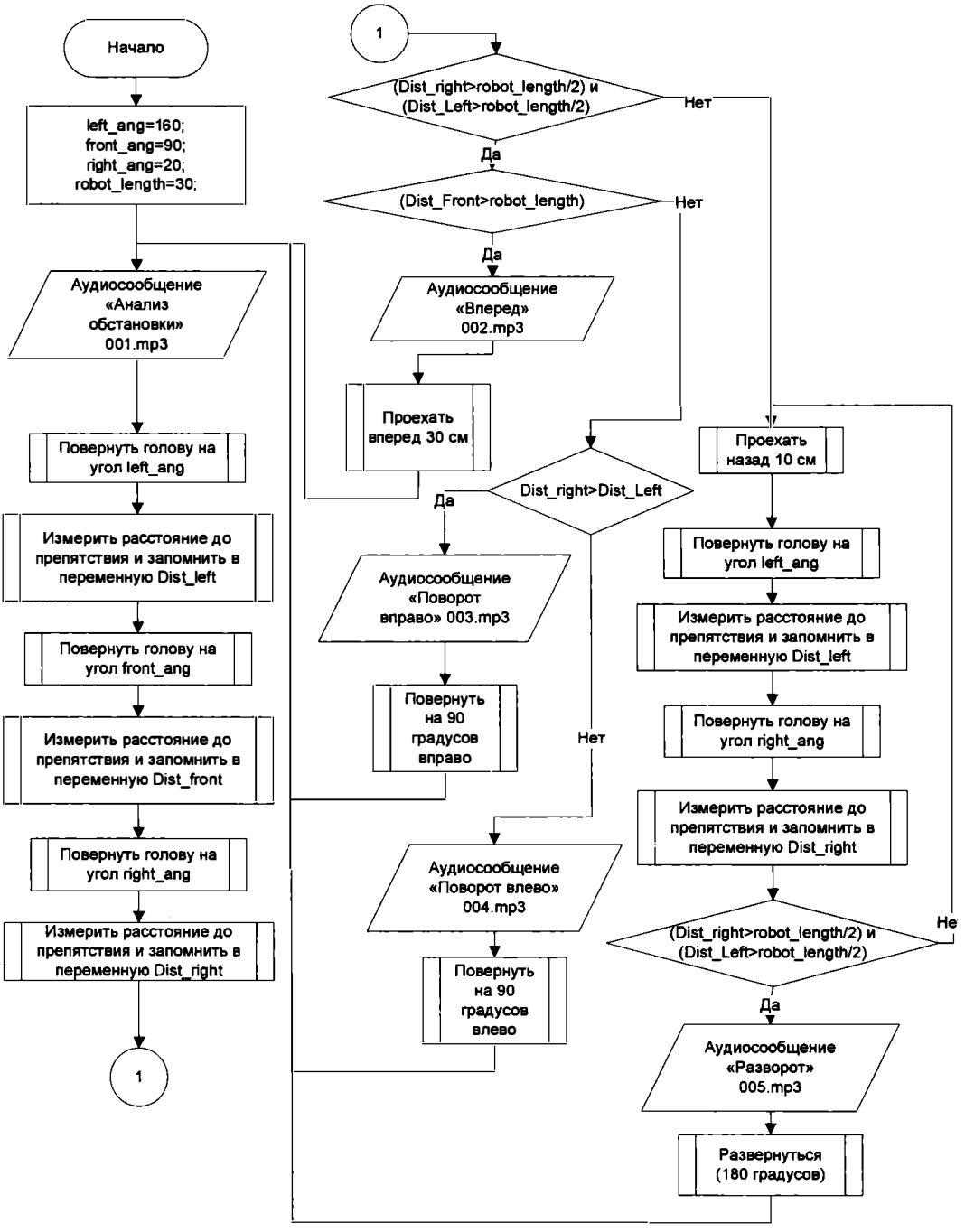


Рис. 16.17. Алгоритм программы обхода препятствий с аудиосообщениями

```
  mp3_set_volume (15);
}
// Основная программа.
void loop ()
{
//6 секунд играть 001.mp3 .
  mp3_play (1);
  delay (6000);
//6 секунд играть 002.mp3 .
  mp3_play (2);
  delay (6000);
//6 секунд играть 003.mp3 .
  mp3_play (3);
  delay (6000);
//6 секунд играть 004.mp3 .
  mp3_play (4);
  delay (6000);
//6 секунд играть 005.mp3 .
  mp3_play (5);
  delay (6000);
  for (int i = 0; i < 4; i++)
  {
    // играть предыдущий файл.
    mp3_prev ();
    delay (6000);
  }
  for (int i = 0; i < 4; i++)
  {
    // играть следующий файл.
    mp3_next ();
    delay (6000);
  }
}
```

Если все получилось, и файлы воспроизводятся, можно приступать к написанию программы для «говорящего» робота (листинг 16.2).

Создадим программу с именем listing_16_2.ino и поместим ее в каталог с аналогичным именем: listing_16_2. Поместим в него также библиотеку управления аудиоплеером — файлы DFPlayer_Mini_Mp3.h и DFPlayer_Mini_Mp3.cpp, библиотеку управления моторами — файл motor.h, библиотеку измерения расстояния ультразвуковым сонаром — файл sonar.h, библиотеку работы с гироскопом/акселерометром MPU-6050 — файл gyro_acsel.h.

Подключим три стандартные библиотеки: для создания программного последовательного соединения: `#include <SoftwareSerial.h>`, библиотеку для работы с протоколом I^2C: `#include <Wire.h>`, библиотеку управления сервомоторами: `#include <Servo.h>`.

По аналогии с предыдущими проектами создадим последовательное соединение: `SoftwareSerial DFPSerial(15, 16)` — для связи аудиоплеера и Arduino. Создадим объект `сервомотор` для поворота шеи и инициализируем его на 14-й порт (A0).

Этапы осмотра роботом окружающей остановки поместим в отдельную функцию `look()`. Во время ее выполнения значения расстояния до препятствия справа, слева и прямо заносятся в переменные: `Dist_left`, `Dist_front` и `Dist_right`.

Несложный анализ с использованием операторов `if...else` и `do...while` позволяет роботу с успехом преодолевать препятствия.

Константа/макроподстановка `robot_length` содержит длину робота и служит для анализа — может ли робот развернуться, или ему будут мешать препятствия.

Листинг 16.2. Модифицированная программа объезда препятствий

```
#include <SoftwareSerial.h>
#include "DFPlayer_Mini_Mp3.h"
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
// Подключаем библиотеку управления сервомоторами.
#include <Servo.h>
// Подключаем библиотеку, управляющую моторами.
#include "motor.h"
// Подключаем библиотеку, управляющую дальномером.
#include "sonar.h"
// Подключение библиотеки управления гироскопом.
#include "gyro_acsel.h"
// Создание последовательного соединения.
SoftwareSerial DFPSerial(15, 16);
// Создаем сервомотор поворота головы.
Servo neck;
// Константы - постоянные значения для уточнения углов.
#define left_ang 180
#define front_ang 135
#define right_ang 70
#define robot_length 30
#define servo_pin 14
// Создаем переменные для хранения трех дистанций - слева, впереди, справа.
int Dist_left, Dist_front, Dist_right;
// Функция осмотра препятствий роботом,
// ее тело ниже основной программы.
void look();
// Инициализация.
void setup () {
  //запуск гироскопа.
  giroscop_setup();
  // Инициализируем дальномер Trig = 13, Echo = 12.
  Sonar_init(13, 12);
```

```
  // Переменные – номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 5, 4);
  _stop(); //Двигатели остановлены.
  DFPSerial.begin (9600);
  // Привязка созданного последовательного соединения
  // к DFPlayer.
  mp3_set_serial (DFPSerial);
  // Расчет компенсации утечки нуля гироскопа.
  Calc_CompensatorZ(3000);
  mp3_set_volume (10);
}
// Основная программа.
void loop ()
{
  _stop();
  mp3_play (1);
  delay(2000);
  look();
  // Если проход достаточно широк.
  if((Dist_right>robot_length/2)&&(Dist_left>robot_length/2))
  {
    if (Dist_front > robot_length)
    {
      mp3_play (2);
      delay(2000);
      forward_t(650000); //в микросекундах.
    }
    else
    {
      if (Dist_right > Dist_left)
      {
        mp3_play (3);
        delay(2000);
        Angle(-90);
      }
      else
      {
        mp3_play (4);
        delay(2000);
        Angle(90);
      }
    }
  }
  else //проход узок
  {
    do
    {
      backward_t(550000); //в микросекундах
```

```
      look();
    }
    while ((Dist_right<robot_length/2)||(Dist_left<robot_length/2));
    mp3_play (5);
    delay(2000);
    Angle(180);
  }
}
//==============================================================
void look()
{
  // Инициализируем сервомотор, управление портом servo_pin.
  neck.attach(servo_pin);
  // Поворачиваем голову налево.
  neck.write(left_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(550);
  // Записываем расстояние до препятствия слева.
  Dist_left = Sonar(50);
  // Поворачиваем голову прямо вперед.
  neck.write(front_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(550);
  // Записываем расстояние до препятствия впереди.
  Dist_front =  Sonar(50);
  // Поворачиваем голову направо.
  neck.write(right_ang);
  // Ждем, т. к. поворот занимает небольшое время.
  delay(550);
  // Записываем расстояние до препятствия впереди.
  Dist_right = Sonar(50);
  neck.detach();
}
```

Выводы

Мы научили робота информировать о своем состоянии (сообщение «Анализ обстановки») и принимаемых решениях (сообщения «Вперед», «Разворот», «Поворот направо» и «Поворот налево»). Заметим, что DFPlayer Mini имеет множество функций, которые не были рассмотрены, но и изученного достаточно, чтобы сделать робота забавным и оригинальным. Конечно, весьма интересно было бы научить робота вести диалог, но мощностей нашего контроллера Arduino для этого недостаточно, хотя хоронить эту идею рано, и в дополнительной, 18-й, главе мы рассмотрим, как это реализовать, установив вместо головы робота смартфон со специальной программой.

Следующий же наш проект уже про другого — двухколесного — робота, которого мы научим держать равновесие.

ГЛАВА 17
БАЛАНСИРУЮЩИЙ РОБОТ

Балансирующий робот должен уметь ездить всего на двух колесах и держать при этом равновесие. Сконструировать его можно разными способами. Например, использовать большие колеса и сместить центр тяжести ниже их осей — при этом робот будет держать равновесие, практически, как игрушка-неваляшка. Можно сделать робота с датчиками препятствия спереди и сзади и заставить его балансировать в зависимости от наклона и соответствующей реакции одного из датчиков. Можно и еще что-нибудь придумать.

Мы же оснастим балансирующего робота электронными гироскопом и акселерометром, принцип действия которых уже был рассмотрен нами в *главе 14*.

Для достижения поставленной цели нам потребуется решить следующие задачи:

1. Сконструировать балансирующего робота, в основу балансировки которого положена информация от гироскопа-акселерометра MPU-6050.
2. Написать для робота программу, которая, используя данные, полученные с MPU-6050, заставит двухколесного робота сохранять равновесие.

Сборка балансирующего робота

Схема подключения

Схема подключения робота приведена на рис. 17.1. Мы возьмем два мотора с редукторами и плату Arduino из предыдущих проектов, а в качестве драйвера двигателей — модуль L298 или микросхему L293D (см. рис. 5.11) с радиатором (выводы EN1 и EN2 этой микросхемы можно «посадить» на питание 5 вольт, подав на них тем самым постоянную единицу, или подключить их к портам D10 и D11 платы Arduino). Фактически, электрическая схема балансирующего робота мало чем отличается от схемы робота из *главы 14*, за исключением меньшего количества моторов. Оснастим робота красным светодиодом LED1 — он будет информировать

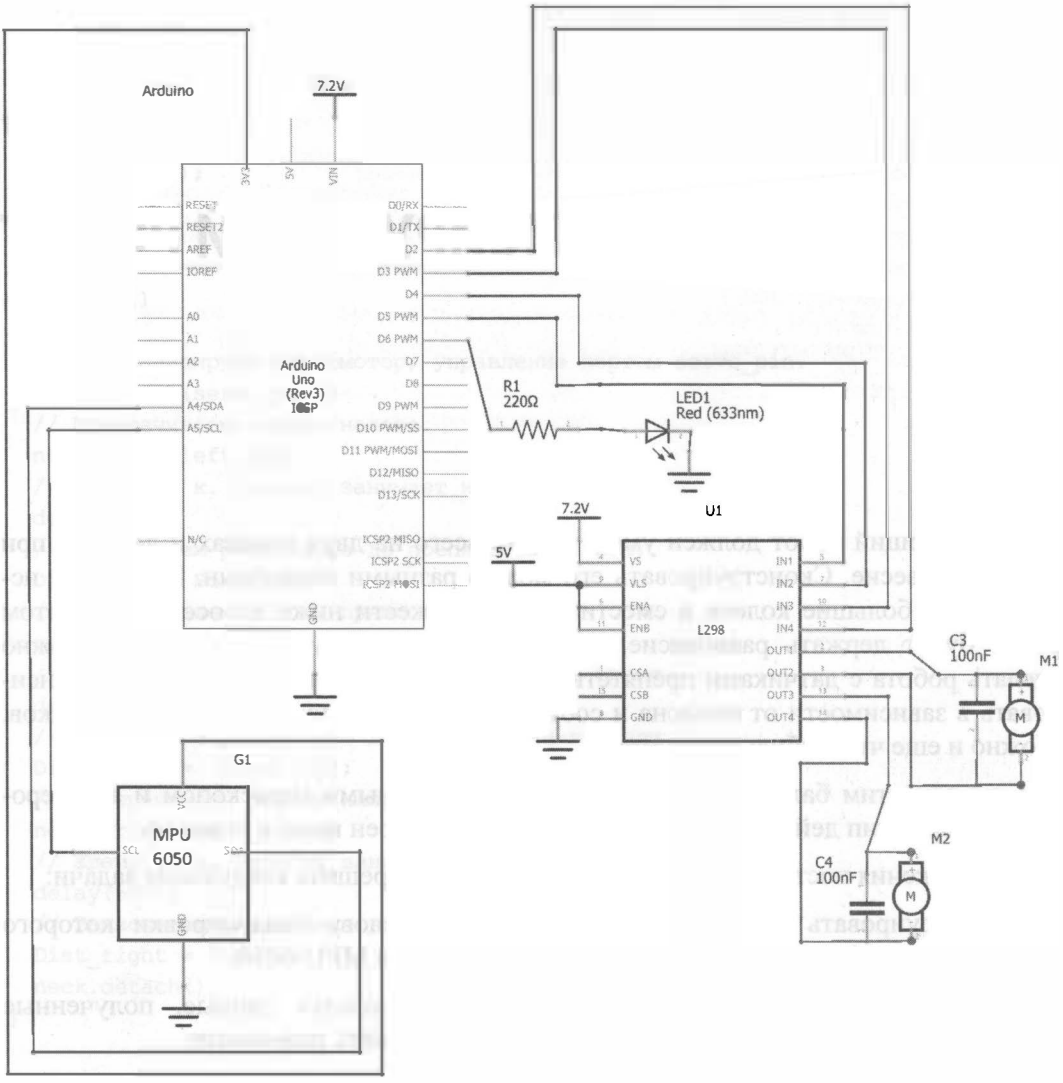


Рис. 17.1. Полная схема балансирующего робота

о готовности робота к работе. На клеммы моторов припаяем керамические конденсаторы — без них электромагнитные помехи от работающих двигателей будут мешать работе электроники робота.

Конструкция

На рис. 17.2 показан сам робот, участвующий в испытаниях. Следует заметить, что чем выше центр тяжести такого робота, тем он устойчивее в сравнении со всем известной игрушкой «ванька-встанька», что на первый взгляд парадоксально. Дело в том, что устойчивость робота растет за счет увеличения момента инерции —

он падает медленнее, а, значит, момент начала падения легче отследить и компенсировать. Поэтому тяжелые аккумуляторы следует перенести в его верхнюю часть.

Для сборки робота мы возьмем три деревянные (фанерные) линейки (рис. 17.3) и стандартные элементы, неоднократно примененные нами в предыдущих главах: Arduino UNO, Arduino Sensor Shield v5.0, драйвер двигателей, бокс для аккумуляторов формата 18650, литиевые аккумуляторы формата 18650, выключатель электропитания, два мотора с редукторами, гироскоп-акселерометр MPU-6050.

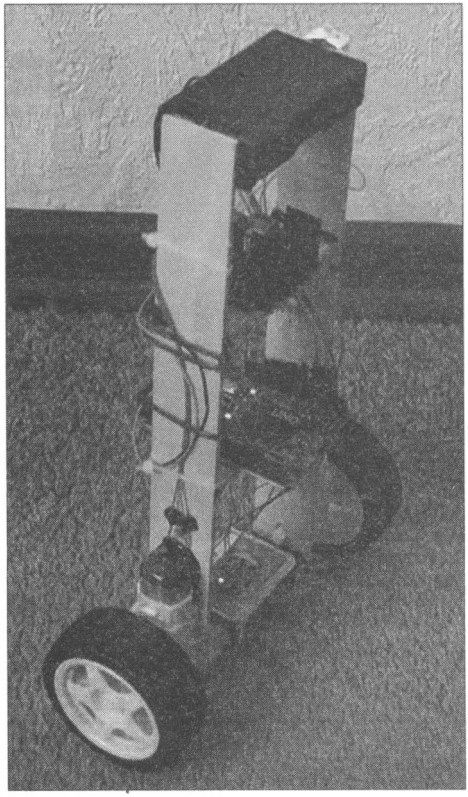


Рис. 17.2. Балансирующий робот с высоким центром тяжести

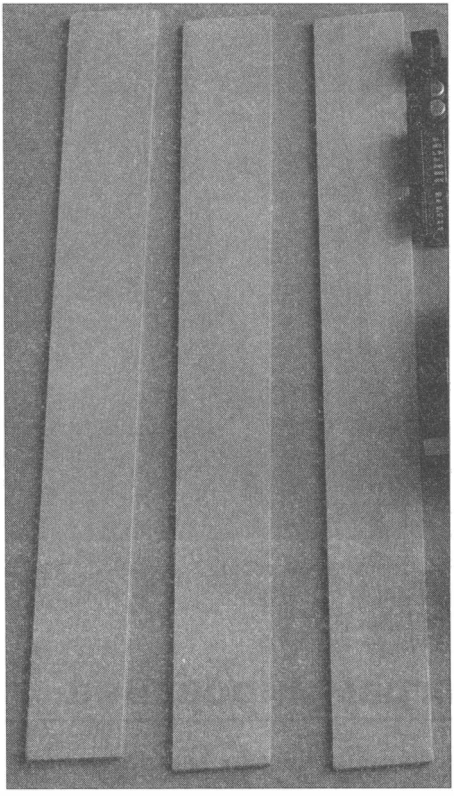


Рис. 17.3. Фанерные линейки

Используя клеевой пистолет (термоклей), из линеек и бокса для аккумуляторов следует собрать раму, подобную изображенной на рис. 17.4, закрепить на раме двигатели и прочие компоненты робота и соединить их согласно схеме, приведенной на рис. 17.1.

Демонстрационный ролик

Процесс сборки робота представлен в ролике «Как сделать балансирующего робота»: **https://youtu.be/giElk_ay1u8**.

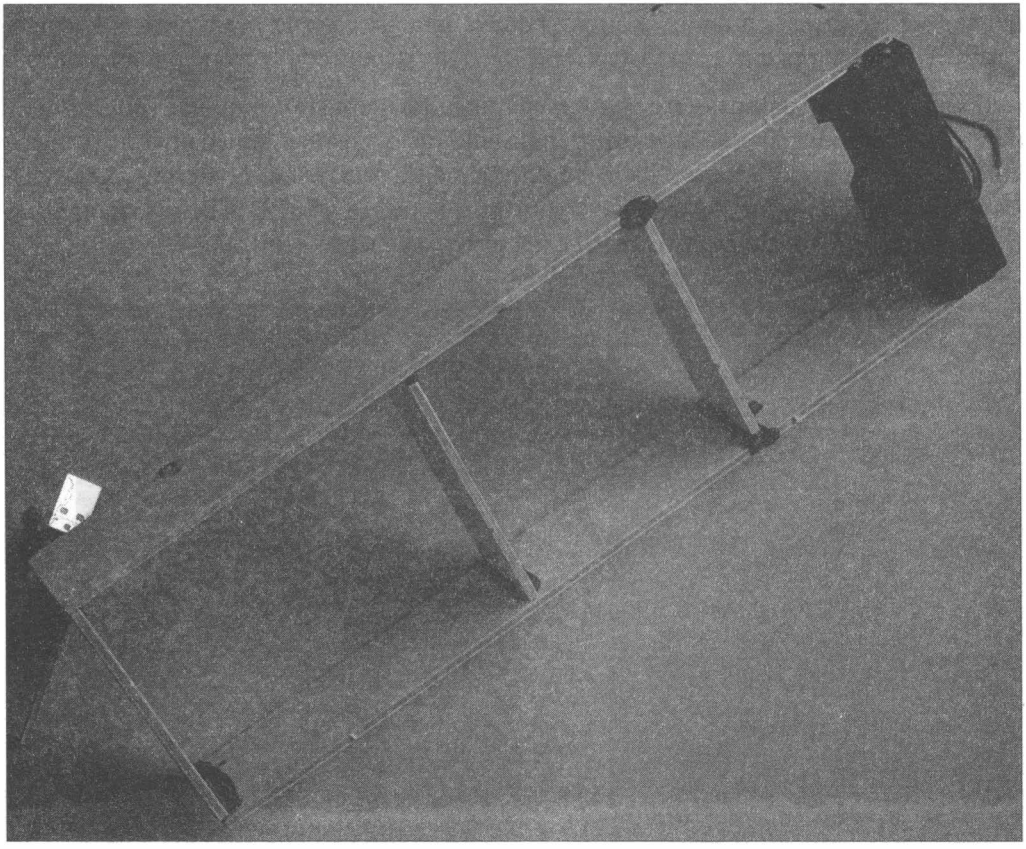


Рис. 17.4. Сборка рамы робота

Программирование

Рассмотрим несколько вариантов программ, управляющих балансирующим роботом.

Программа на показаниях гироскопа

Воспользуемся здесь подготовленными ранее библиотеками `motor.h` (команды моторам) и `gyro_acsel.h` (работа с гироскопом-акселерометром MPU-6050).

Первая программа (листинг 17.1) анализирует угловую скорость вращения робота вокруг оси X и стремится снизить ее до нуля. При этом, когда робот начинает падать, ему дается команда двигаться в направлении падения, компенсируя наклон. Когда же робот встает прямо, его моторы выключаются (рис. 17.5). На практике робот не может стоять неподвижно и вынужден непрерывно балансировать, как клоун в цирке на одноколесном велосипеде.

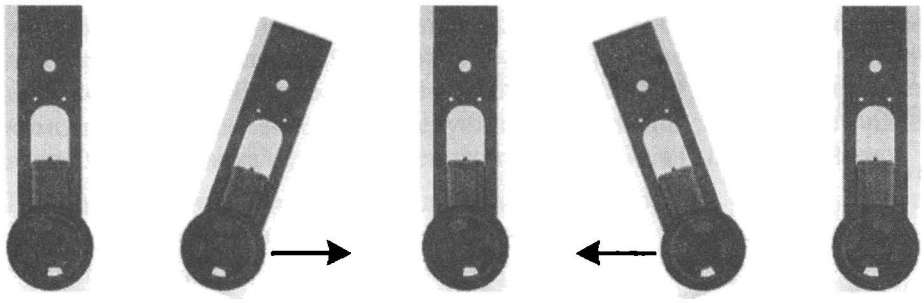


Рис. 17.5. Демонстрация поведения робота: *а* — исходное положение, робот начинает падать вправо; *б* — роботу дана команда двигаться вправо; *в* — робот скомпенсировал наклон и выпрямился; *г* и *д* — обратная ситуация (робот начал падать влево)

Листинг 17.1. Программа балансировки робота на основе анализа показаний гироскопа

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
#include "motor.h"
#include "gyro_acsel.h"
// Пин информационного светодиода.
int  s_diod = 6;
//=====================================================================
void setup()
{
  // Заносим в переменные номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Двигатели остановлены.
  _stop();
  giroscop_setup();
  Serial.begin(9600);
  pinMode(s_diod, OUTPUT);
  _stop();
  for (int i = 0; i < 5; i++)
  {
    delay(100); digitalWrite(s_diod, HIGH);
    delay(100); digitalWrite(s_diod, LOW);
  }
  digitalWrite(s_diod, HIGH);
  GyXsum = 0;
}
//=====================================================================
// Основная программа.
void loop()
{
  const long compensatorX = -257;
```

```
  //Запрос данных от гироскопа MPU-6050
  Data_mpu6050();
  GyXsum = GyXsum + (long)GyX - compensatorX;
  if (GyXsum > 10)
  {
    forward();
  }
  else
  {
    if (GyXsum < -10) backward(); else _stop();
  }
}
```

Алгоритм работы программы из листинга 17.1 довольно прост (рис. 17.6). Считая, что временны́е интервалы между опросами гироскопа одинаковые, будем опериро-

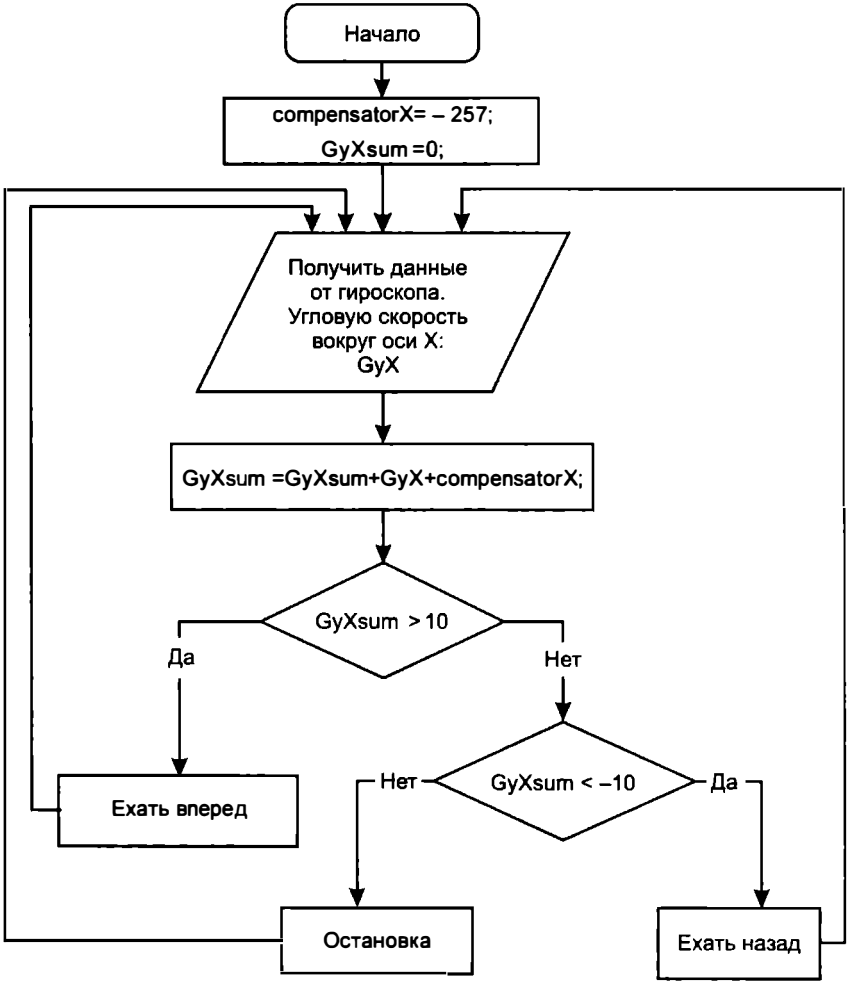


Рис. 17.6. Алгоритм программы балансировки на основе анализа показаний гироскопа

вать напрямую данными, поступающими от MPU-6050. Здесь надо иметь в виду, что даже при неподвижном роботе гироскоп показывает ненулевые значения угловой скорости. Чтобы учесть дрейф нуля показаний гироскопа, введем специальный компенсатор (`compensatorX`). Следует подобрать для него такое значение, чтобы вычисляемая угловая скорость неподвижного робота была равна нулю.

Практически это реализовать довольно сложно, но реакция робота в целом верная — при падении он пытается вернуть себя в начальное положение. При этом устойчивость робота оставляет желать лучшего.

Программа с фильтром Калмана

Для написания второй программы применим немного математики. Чаще всего для решения задач обеспечения устойчивости используют *фильтр Калмана*.

Фильтр Калмана — это рекурсивный фильтр, который оценивает текущее состояние динамической системы, используя данные, содержащие помехи. Фильтр Калмана успешно сглаживает скачки, преобразуя показания состояния системы в предсказуемые величины. При этом он учитывает предыдущие показания и генерирует сглаженное текущее значение.

На сайте **https://github.com/TKJElectronics/KalmanFilter** разработчик Kristian Lauszus выложил свою реализацию библиотеки фильтра Калмана (рис. 17.7). Следует скачать файлы, обозначенные стрелками, и положить их в папку с программой из листинга 17.2.

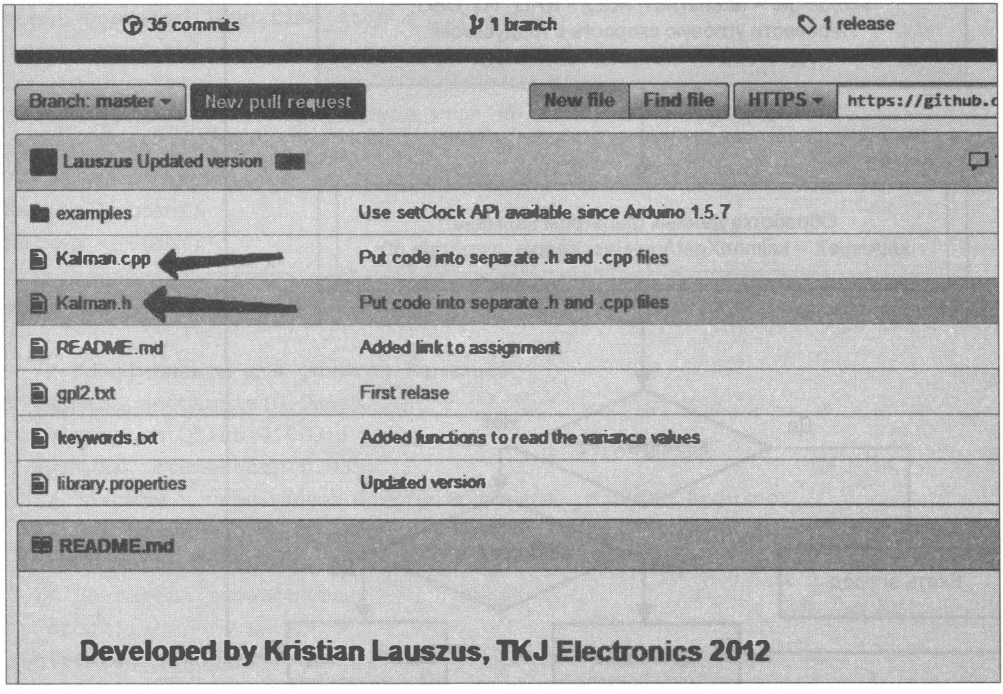


Рис. 17.7. Библиотека фильтра Калмана

Электронный архив

Файл kalman.h находится в папке listing_17_2 электронного архива, сопровождающего книгу (см. *приложение 2*). При его использовании файл kalman.cpp вам не понадобится.

Алгоритм (рис. 17.8) разбит на два этапа. Первый этап — прогнозирование, на этом этапе фильтр экстраполирует значения переменных состояния и их неопределенности. Во время второго этапа результат уточняется на основании полученных измерений.

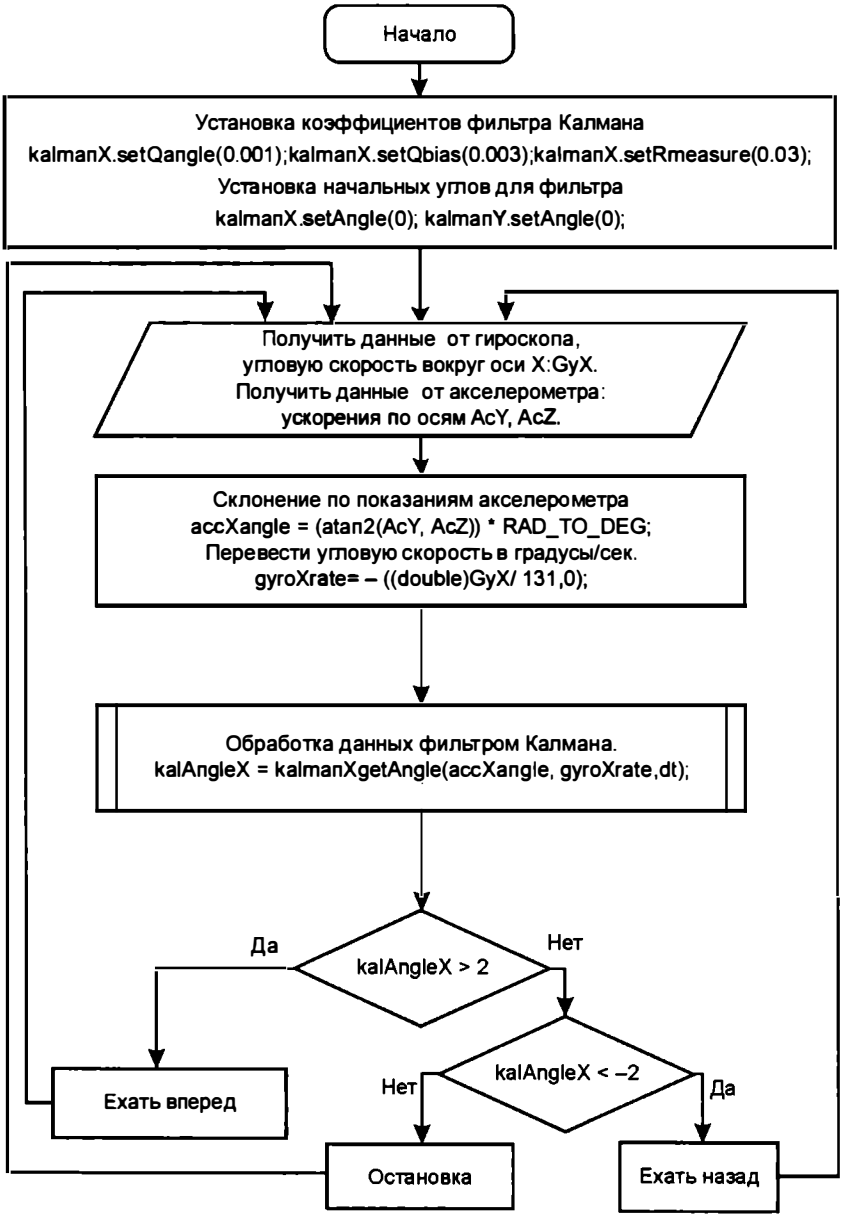


Рис. 17.8. Алгоритм программы балансировки с использованием фильтра Калмана

Пусть MPU-6050 расположен на роботе так, что его ось *Y* направлена вперед, а ось *X*— вправо по отношению к ходу робота (оси эти изображены на плате прибора). Тогда для балансировки робота с использованием фильтра Калмана потребуется снимать с MPU-6050 три значения: ускорения по осям *Y* и *Z* и угловую скорость по оси *X*. Пользуясь формулой (14.2) из *главы 14*, находим угол склонения по данным акселерометра (угол нужно будет перевести из радиан в градусы).

Затем вызывается фильтр Калмана с входными параметрами: угол склонения по данным акселерометра, отрицательная угловая скорость по оси *X* (приведенная к градусам в секунду) и приращение времени.

Листинг 17.2. Программа балансировки с использованием фильтра Калмана

```
#include "Kalman.h"
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
#include "motor.h"
#include "gyro_acsel.h"
// Пин информационного светодиода.
int  s_diod = 6;

// Переменные для хранения данных времени
// работы контроллера Arduino.
unsigned long time_data_request, time_in_system;
// Время, через которое производится опрос гироскопа-акселерометра.
const unsigned long time_step = 10;
double accXangle; // Угол, посчитанный по акселерометру.
double gyroXangle ; // Угол, посчитанный по гирокомпасу.
double gyroXrate; // Приращение угла по показаниям гироскопа.
double kalAngleX ;
// Результирующий угол после обработки фильтром Калмана.
Kalman kalmanX;
Kalman kalmanY;
//================================================================
void setup()
{
  // Коэффициенты для фильтра Калмана.
  kalmanX.setQangle(0.001);
  kalmanX.setQbias(0.003);
  kalmanX.setRmeasure(0.03);
  // Заносим в переменные номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Двигатели остановлены.
  _stop();
  giroscop_setup();
  Serial.begin(9600);
  pinMode(s_diod, OUTPUT);
```

```
  digitalWrite(s_diod, LOW);
  delay(200);
  digitalWrite(s_diod, HIGH);
  _stop();
  kalmanX.setAngle(0); // Set starting angle
  kalmanY.setAngle(0);
  timer = micros();
}
//=============================================================
// Основная программа.
void loop()
{
  int i = 0;
  while (true)
  {
    // Запрос данных от гироскопа MPU-6050
    Data_mpu6050();

    // Расчет угла по показаниям акселерометра.
    accXangle = (atan2(AcY, AcZ)) * RAD_TO_DEG;
    // Расчет приращения угла по гироскопу.
    gyroXrate = -((float)GyX / 131.0);
    // Расчет угла по показаниям гироскопа.
    // Обработка данных фильтром Калмана.
    kalAngleX = kalmanX.getAngle(accXangle, gyroXrate,
                                 (float)( micros() - timer) / 1000000.0);
    timer = micros();
    // Компенсируем наклон
    if (kalAngleX > 2.0)
    {
      forward();
    }
    else if (kalAngleX < -2.0)
    {
      backward();
    }
    else {
      _stop();
    }
   /* if (i == 100)
    {
      Serial.print("kalAngleX=");
      Serial.println( kalAngleX);
      i = 0;
    }
    i++; */
  }
}
```

Если раскомментировать строки, выделенные полужирным курсивом, выключить основное питание (или снять колеса) и подключить робота к компьютеру, можно будет следить за изменением угла наклона робота через окно порта (рис. 17.9). Угол, рассчитанный программой, визуально совпадает с заданным углом (углом наклона робота).

В результате испытаний робота с фильтром Калмана было замечено, что фильтр работает, показания плавные, но они отстают от реальности по времени, что приводит к тому, что реакция моторов становится неверной.

Более детальное исследование работы фильтра подтвердило, что контроллеру Arduino при математических расчетах не хватает скорости.

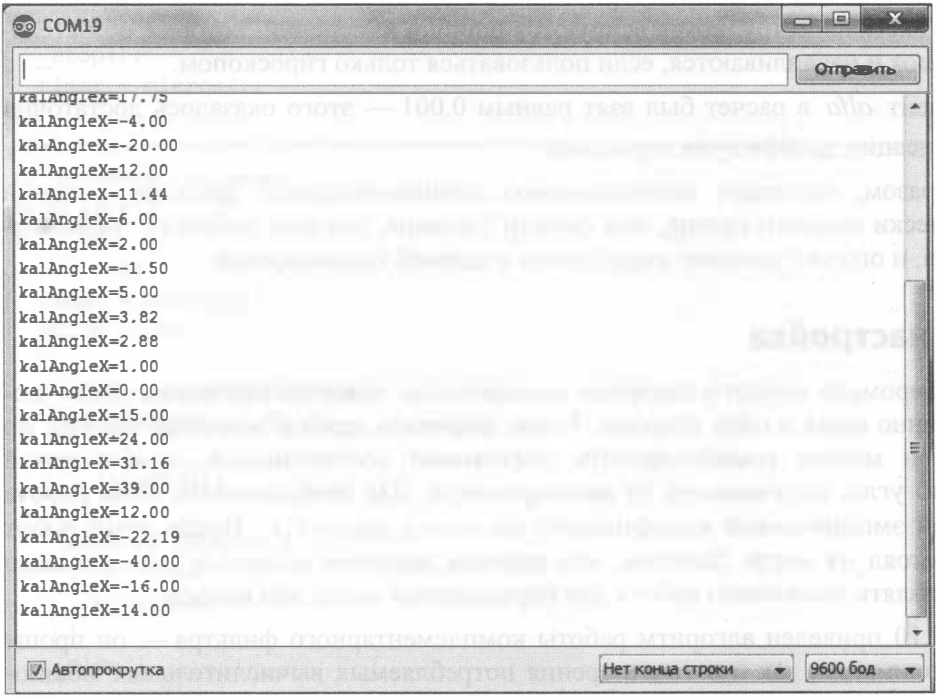


Рис. 17.9. Демонстрация работы программы из листинга 17.2 (данные выводятся в порт)

Программа с комплементарным фильтром

Комплементарный фильтр

Существуют другие, менее инерционные и сравнительно быстрые фильтры, позволяющие, смешивая данные с различных приборов, получать эффективные выходные значения. К таким фильтрам относится *комплементарный фильтр*.

При объединении показаний акселерометра и гироскопа на основе комплементарного фильтра значение угла наклона рассчитывается по формуле:

$$GyXsum_1 = (1 - alfa) \times (GyXsum_0 + GyX \times \Delta t) + alfa \times AcYsum, \qquad (17.1)$$

где $GyXsum_1$ — расчетное значение угла наклона, $GyXsum_0$ — значение угла наклона из предыдущего расчета, *alfa* — коэффициент комплементарного фильтра (подбирается экспериментально от 0,001 до 0,05), *GyX* — угловая скорость, Δt — время итерации, *AcYsum* — угол склонения, рассчитанный по текущим показаниям акселерометра.

Итоговая величина является долевой суммой интегрированного значения показаний гироскопа и мгновенного значения показаний акселерометра. Доля показаний акселерометра мала, поскольку они умножаются на малую величину *alfa*, равную 0,001–0,05, и основной вклад вносят показания гироскопа — они умножаются на величину (1 – *alfa*), равную 0,999–0,95 (настраивается экспериментально). Однако, благодаря акселерометру, компенсируются ошибки измерений, которые обязательно возникают и накапливаются, если пользоваться только гироскопом.

Коэффициент *alfa* в расчет был взят равным 0,001 — этого оказалось достаточно для компенсации дрейфа нуля гироскопа.

Таким образом, благодаря использованию комплементарного фильтра, который математически намного проще, чем фильтр Калмана, удалось добиться стабилизации робота, и он стал успешно справляться с задачей балансировки.

Точная настройка

Если акселерометр немного наклонен относительно поверхности земли, робот может постоянно ехать в одну сторону. Точно закрепить прибор довольно сложно, но этот наклон можно компенсировать постоянной составляющей, прибавляемой к значению угла, получаемому от акселерометра. Для описываемого нами робота был введен эмпирический коэффициент: `balancing_zerro=2.4`. После этого робот уверенно стоял на месте. Заметим, что изменяя значение `balancing_zerro`, можно также управлять движением робота для перемещения назад или вперед.

На рис. 17.10 приведен алгоритм работы комплементарного фильтра — он проще как для понимания, так и с точки зрения потребляемых вычислительных мощностей. Созданная по этому алгоритму программа (листинг 17.3) снабжена понятными комментариями. Для достижения положительного эффекта следует скорректировать программу под своего робота, поэкспериментировав с коэффициентами.

Листинг 17.3. Программа балансировки с использованием комплементарного фильтра

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
#include "motor.h"
#include "gyro_acsel.h"
// Пин информационного светодиода.
int  s_diod = 6;
// начальное смещение нуля баланса.
double balancing_zerro = 2.4;
```

```
//=====================================================================
void setup()
{
   // Заносим в переменные номера контактов (пинов) Arduino.
  // Для левых и правых моторов машинки.
  setup_motor_system(2, 3, 4, 5);
  // Двигатели остановлены.
  _stop();
  giroscop_setup();
  Serial.begin(9600);
  pinMode(s_diod, OUTPUT);
  digitalWrite(s_diod, LOW);
  delay(200);
  digitalWrite(s_diod, HIGH);
  _stop();
  timer = micros();
}
//=====================================================================
// Основная программа.
void loop()
{
  int i=0;
  timer = micros();
  while (true)
  {
    // Компенсируем наклон.
    if (GyXsum > 1.0)
    {
      forward();
    }
    else if (GyXsum < -1.0)
    {
      backward();
    }
    else {
      _stop();
    }
    // Запрос данных от гироскопа MPU-6050
    Data_mpu6050();

    // Расчет угла по показаниям акселерометра
    // с учетом корректировки точки равновесия balancing_zerro.
    AcYsum = (atan2(AcY, AcZ)) * RAD_TO_DEG+balancing_zerro;
    // Измерение наклона по X.
    // Использование Комплементарного фильтра,
    // alfa - коэффициент фильтра.
    double alfa = 0.001;
    GyXsum = (1 - alfa) * (GyXsum + ((double)GyX * (double)(micros() - timer)) /
                                                    131000000.0) + alfa * AcYsum;
    // Для перевода угловой скорости в угол нужно знать время!
    timer = micros();
```

```
    /* if (i == 100)
    {
      Serial.print("GyXsum =");
      Serial.println(GyXsum);
      i = 0;
    }
    i++;
    */
  }
}
```

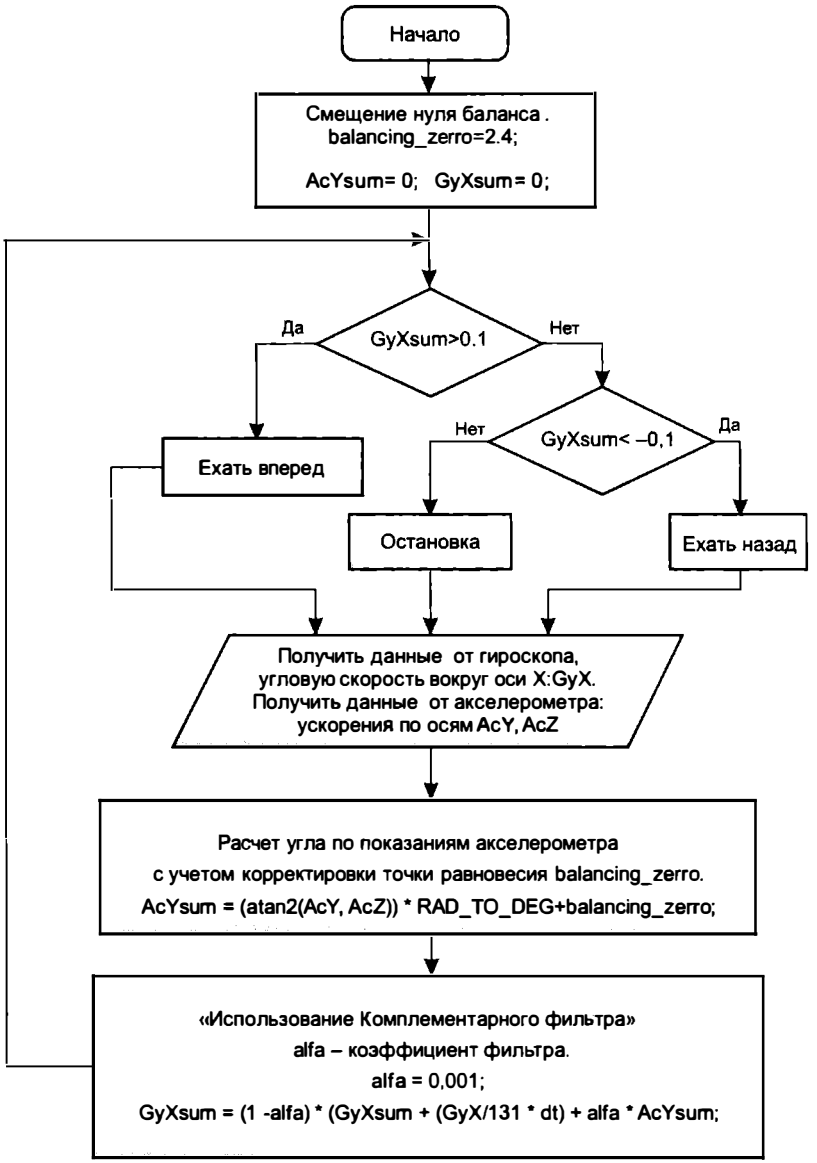


Рис. 17.10. Алгоритм программы балансировки с использованием комплементарного фильтра

Выводы

В этой главе мы создали несложную схему и конструкцию балансирующего робота. Поэтапно рассмотрели три версии программы, предназначенной для организации балансировки робота, и выбрали наиболее удачный алгоритм — с комплементарным фильтром.

Таким образом, цель, поставленная в этой главе, достигнута.

Рекомендуем вам и дальше развивать полученные знания и умения, не бояться экспериментировать, пробовать свои версии роботов и алгоритмов.

ГЛАВА 18

НЕКОТОРЫЕ УЛУЧШЕНИЯ И ПРОЧАЯ ПОЛЕЗНАЯ ИНФОРМАЦИЯ

Если не хватает портов ввода/вывода

Ситуация, когда некуда подключать новые устройства и датчики, может возникнуть на этапе повышения сложности проекта. Расширить соответствующие возможности контроллера Arduino можно, используя ряд приемов, рассмотренных далее.

Сдвиговые регистры: подключаем 8 светодиодов, электронное табло и управляем 18-ю выходами

Вам требуется подключить несколько дополнительных двигателей постоянного тока или светодиоды, мигание которых управляется контроллером?

Задачу можно решить при помощи микросхемы 74HC595 (рис. 18.1), представляющей собой *сдвиговый регистр*. Микросхема позволяет, используя только три порта Arduino, получить 8 выходных сигналов. Если выходных сигналов требуется больше, то сдвиговые регистры могут быть подключены последовательно, но на Arduino все равно будут задействованы только три порта.

Схема подключения сдвигового регистра 74HC595 к Arduino UNO при управлении восемью светодиодами приведена на рис. 18.2: светодиоды LED1–LED8 через токоограничивающие резисторы подключены к выходам Q0–Q7 микросхемы 74HC595. Контакты SH_CP, ST_CP и DS микросхемы 74HC595 подключены к контактам D4–D6 Arduino (рис. 18.3).

Листинг 18.1 демонстрирует работу со сдвиговым регистром. В регистр последовательно записываются числа от 0 до 255, что соответствует всем вероятным состояниям восьми выходов, к которым подключены светодиоды. Практически светодиоды будет подсвечивать двоичный код текущего значения переменной `j`.

Потребуется определить три константы с номерами портов: `latchPin`, `clockPin` и `dataPin` для `ST_CP`, `SH_CP` и `DS`. В функции `setup()` переводим эти порты в состояние вывода данных. Далее в цикле заносим значения от 0 до 255 в сдвиговый регистр,

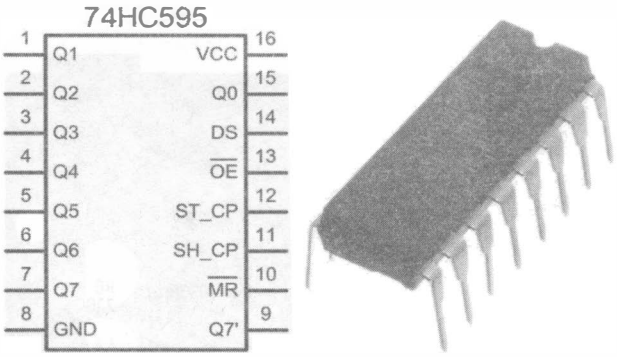


Рис. 18.1. Сдвиговый регистр 74HC595: контакты 15 (Q0) и 1–7 (Q1–Q7) — параллельные выходы; контакт 8 (GND) — «земля»; контакт 9 (Q7') — выход для последовательного соединения микросхем сдвиговых регистров; контакт 10 (MR) — сброс значений регистра (сброс происходит при «0» на входе); контакт 11 (SH_CP) — вход для тактовых импульсов; контакт 12 (ST_CP) — синхронизация («защелкивание») выходов; контакт 13 (OE) — вход для переключения состояния выходов из высокоомного в рабочее; контакт 14 (DS) — вход для последовательных данных; контакт 16 (VCC) — питание

используя функцию `shiftOut(dataPin, clockPin, LSBFIRST, j)`, где `LSBFIRST` — константа, указывающая, что заполнение регистра будет происходить справа.

Листинг 18.1. Пример управления сдвиговым регистром

```
//Порт, подключенный к ST_CP 74HC595
#define latchPin 5
//Порт, подключенный к SH_CP  74HC595
#define clockPin 4
//Порт, подключенный к DS 74HC595
#define dataPin 6
//=================================
 void setup() {
  //устанавливаем режим порта выхода
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
//=================================
void loop() {
  for (int j = 0; j <= 255; j++) {
    //устанавливаем LOW на latchPin
    //пока не окончена передача байта.
    digitalWrite(latchPin, LOW);
     //MSBFIRST (Most Significant Bit First) - слева.
     //LSBFIRST (Least Significant Bit First) – справа.
    shiftOut(dataPin, clockPin, LSBFIRST, j);
    //устанавливаем HIGH на latchPin,
    //чтобы проинформировать регистр, что передача окончена.
```

```
    digitalWrite(latchPin, HIGH);
    delay(1000);
  }
}
```

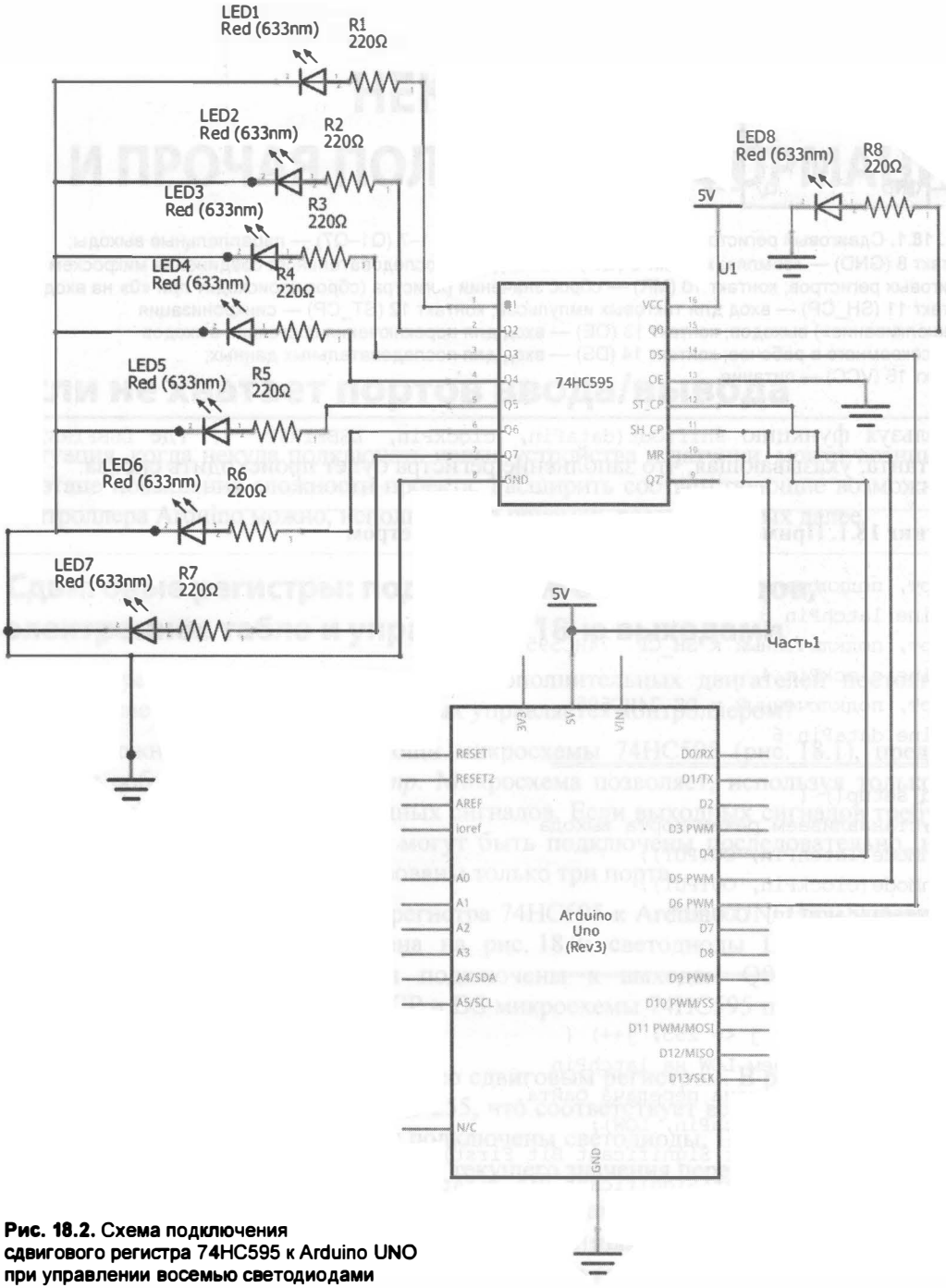


Рис. 18.2. Схема подключения сдвигового регистра 74HC595 к Arduino UNO при управлении восемью светодиодами

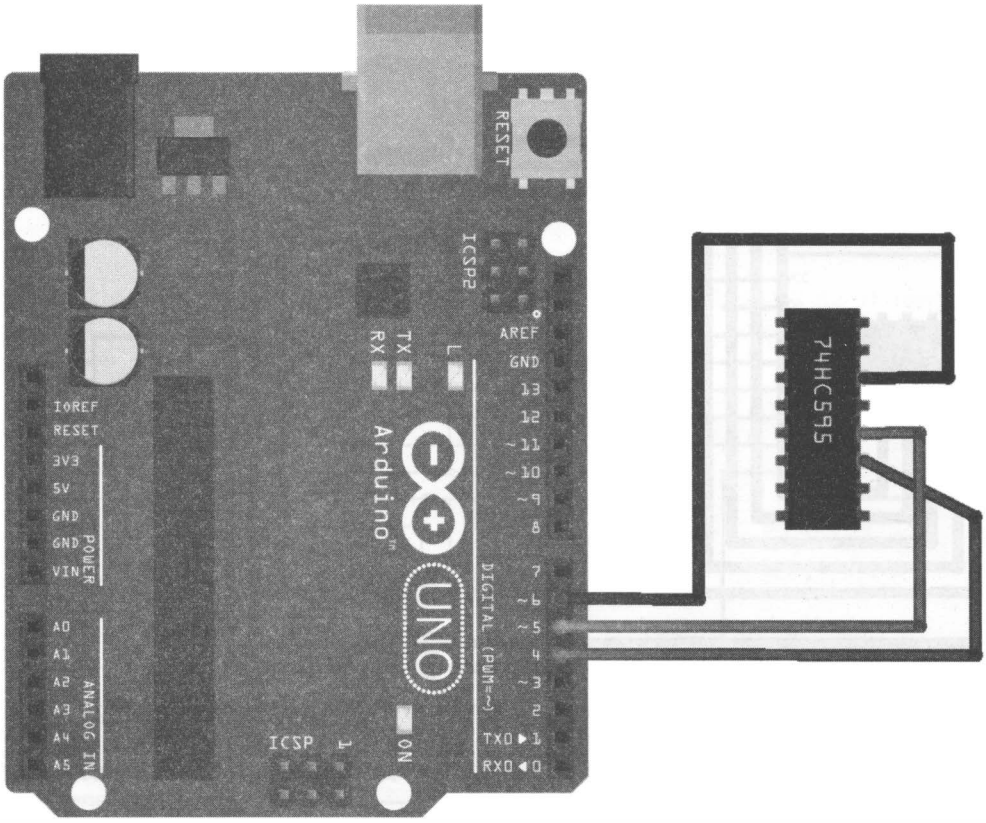


Рис. 18.3. Монтажная схема подключения сдвигового регистра 74HC595 к Arduino UNO

Возможная монтажная схема подключения к сдвиговому регистру 74HC595 цифрового табло приведена на рис. 18.4.

Существенно упростить работу со сдвиговыми регистрами можно при помощи, например, библиотеки `ShiftRegister595.h`. Пример использования этой библиотеки приведен в листинге 18.2. Листинг показывает работу со структурой из трех сдвиговых регистров, представленной на схеме из рис. 18.5. Сдвиговые регистры соединены последовательно, и для того чтобы записать данные в регистр `U3`, они должны быть сдвинуты через предыдущие регистры: `U1` и `U2`. Таким образом, в этой программе, как и в листинге 18.1, определены три константы с номерами портов управления регистрами, затем создан объект `shReg`, который представляет собой совокупность функций и данных по работе со сдвиговым регистром. Далее в программе в пару старших регистров заносятся случайные числа, а в регистр `U1` — номер итерации: от 0 до 255. Все это реализовано с использованием сдвигов данных, а именно так: первая команда `shReg.write(random(0,255))` приводит к заполнению младшего регистра, вторая `shReg.write(random(0,255))` записывает в младший регистр `U1` новые данные, но предыдущие данные не стираются, а попадают в регистр `U2`, третья команда `shReg.write(j)` опять совершает сдвиг и запись номера итерации в младший регистр, а данные от первой команды `shReg.write(random(0,255))` попадают (сдвигаются) в регистр `U3`.

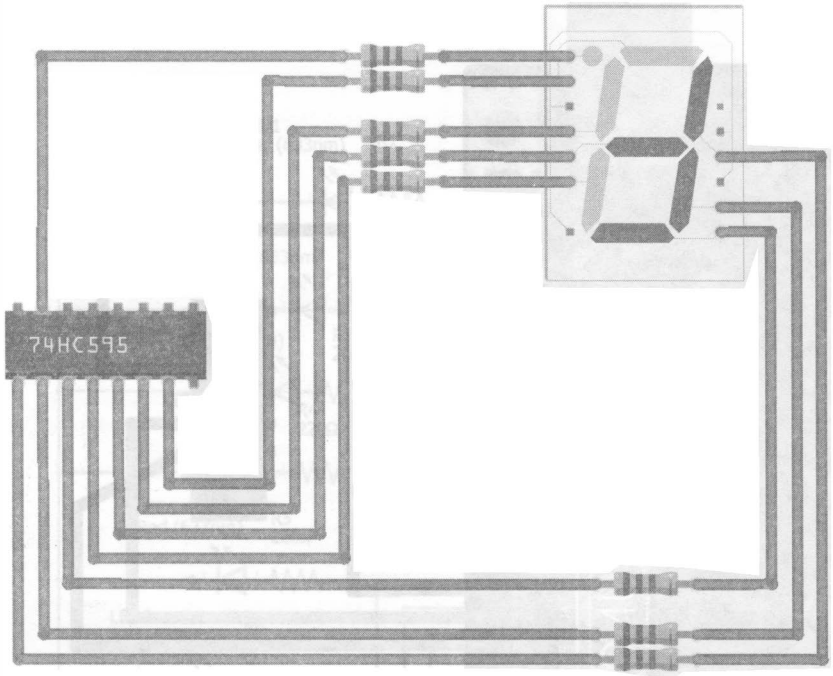


Рис. 18.4. Монтажная схема подключения сдвигового регистра 74HC595 к электронному табло

Если к выходам этих регистров подключить светодиоды, можно будет наблюдать интересную динамичную картину мигания.

Электронный архив

Библиотека ShiftRegister595.h находится в составе электронного архива, сопровождающего книгу (см. *приложение 2*).

Листинг 18.2. Пример управления тремя сдвиговыми регистрами

```
#include "ShiftRegister595.h"
//Порт, подключенный к ST_CP 74HC595
#define latchPin 5
//Порт, подключенный к SH_CP  74HC595
#define clockPin 4
//Порт, подключенный к DS 74HC595
#define dataPin 6
ShiftRegister595 shReg( latchPin , clockPin , dataPin );
void setup() {
  randomSeed(analogRead(0));
}
void loop() {
  for (int j = 0; j < 256; j++)
  {
    shReg.write(random(0,255));
```

```
    shReg.write(random(0,255));
    shReg.write(j);
    delay(1000);
  }
}
```

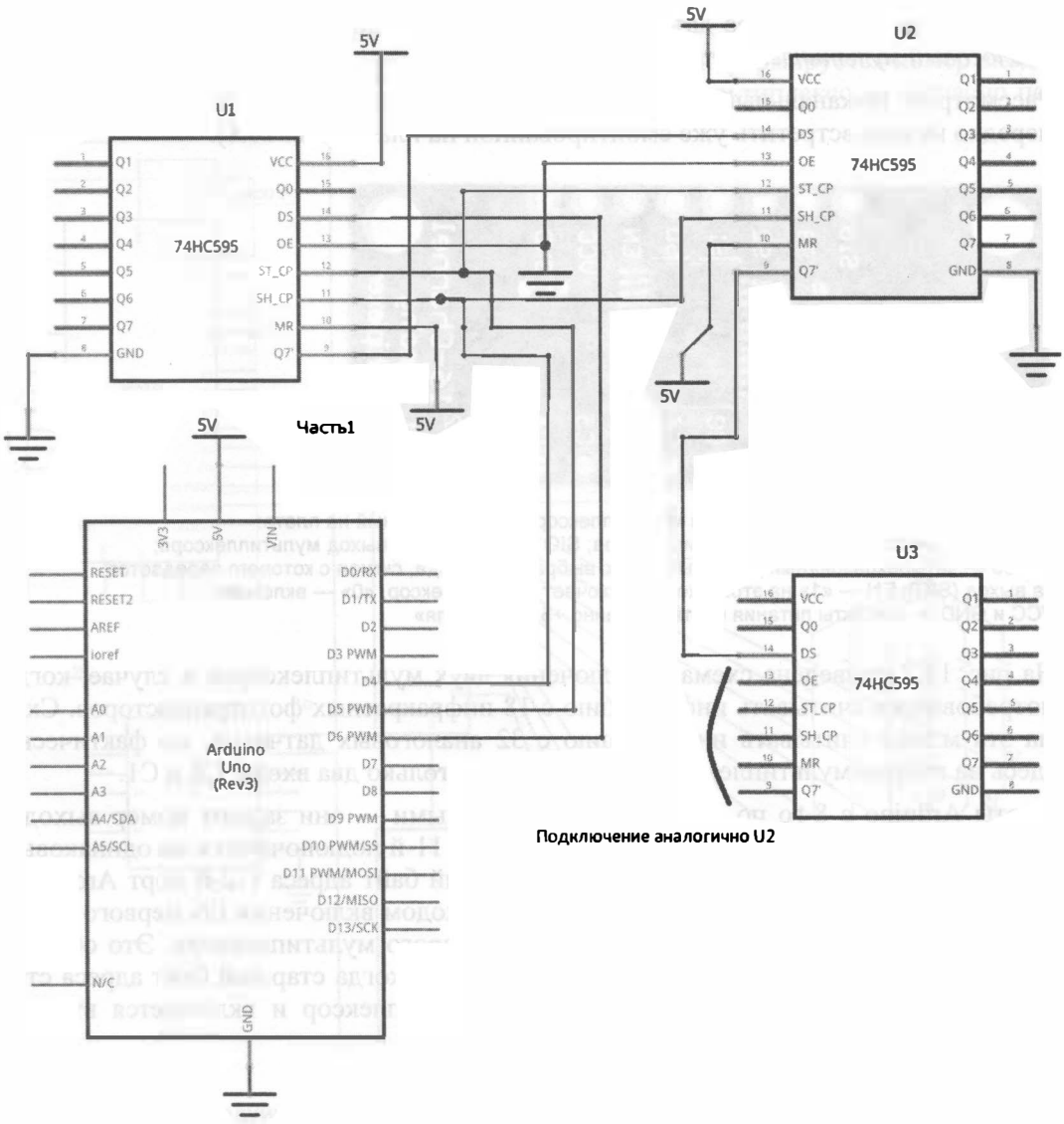


Рис. 18.5. Схема подключения сдвиговых регистров 74HC595 к Arduino UNO при управлении 24-мя выходами

Аналоговый мультиплексор: подключаем 16 и более аналоговых датчиков

Что делать, если нужно подключить к роботу большое количество аналоговых датчиков? Конечно, можно попытаться использовать более мощный контроллер — например, Arduino Mega, но и там всего 12 аналоговых входов. А если требуется подключить 16 или более датчиков? Выходом из подобной ситуации может стать *аналоговый мультиплексор*.

Рассмотрим 16-канальный аналоговый мультиплексор 74HC4067. Эту микросхему нередко можно встретить уже смонтированной на плате (рис. 18.6).

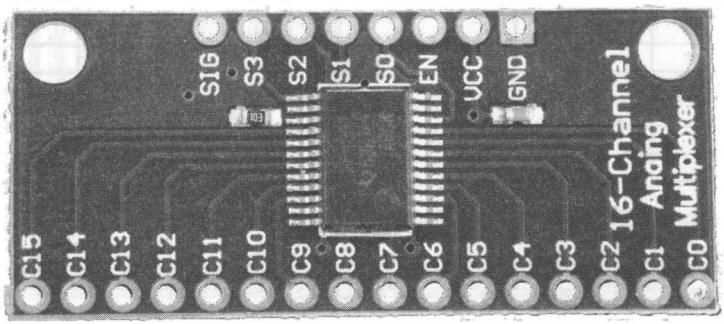


Рис. 18.6. 16-канальный аналоговый мультиплексор, смонтированный на плате: C0–C15 — аналоговые входы мультиплексора; SIG — аналоговый выход мультиплексора; S0–S3 — четырехразрядный двоичный адрес выбранного входа, сигнал с которого передается на выход (SIG); EN — «1» на этом входе отключает мультиплексор, «0» — включает; VCC и GND — контакты питания соответственно +5 В и «земля»

На рис. 18.7 приведена схема подключения двух мультиплексоров в случае, когда потребовалось считывать информацию с 18 инфракрасных фототранзисторов. Схема эта может считывать информацию с 32 аналоговых датчиков, но фактически здесь на втором мультиплексоре задействованы только два входа: C0 и C1.

Порты Arduino с 8-го по 12-й являются адресными — они задают номер выхода, с которого считывается сигнал. Порты с 8-го по 11-й подключаются на одинаковые входы обоих мультиплексоров (S0–S3). Старший байт адреса (12-й порт Arduino), помимо того, что управляет непосредственно входом включения EN первого мультиплексора, через инвертор подается на EN второго мультиплексора. Это обеспечивает переключение между мультиплексорами — когда старший байт адреса становится «единицей», отключается первый мультиплексор и включается второй, в противном случае работает первый.

Инвертор

Инвертор инвертирует логическое значение: «ноль» превращает в «единицу» и наоборот. На рис. 18.7 это микросхема SN74LS14N.

Причина использования для адресации портов с 8-го по 12-й

Подключение портов Arduino именно в такой последовательности не случайно, поскольку они представляют собой один внутренний регистр контроллера ATMega (на

которых основаны большинство плат Arduino). Это порт PORTB (в него входят с 8-го по 13-й порты Arduino UNO, Nano, ProMini), что позволяет довольно просто формировать адрес для желаемого входа мультиплексоров (листинг 18.3).

Порты с 8-го по 12-й переводятся в режим вывода, порт A0 переводится в режим ввода. Для анализа полученной информации на ПК задействуем стандартный порт связи.

В цикле основной программы значения `j` изменяются от 0 до 18 (цикл по `j`), их получает регистр PORTB, на портах Arduino с 8-го по 12-й формируется номер входа, с которого планируется считать аналоговый сигнал. Мультиплексоры согласно по-

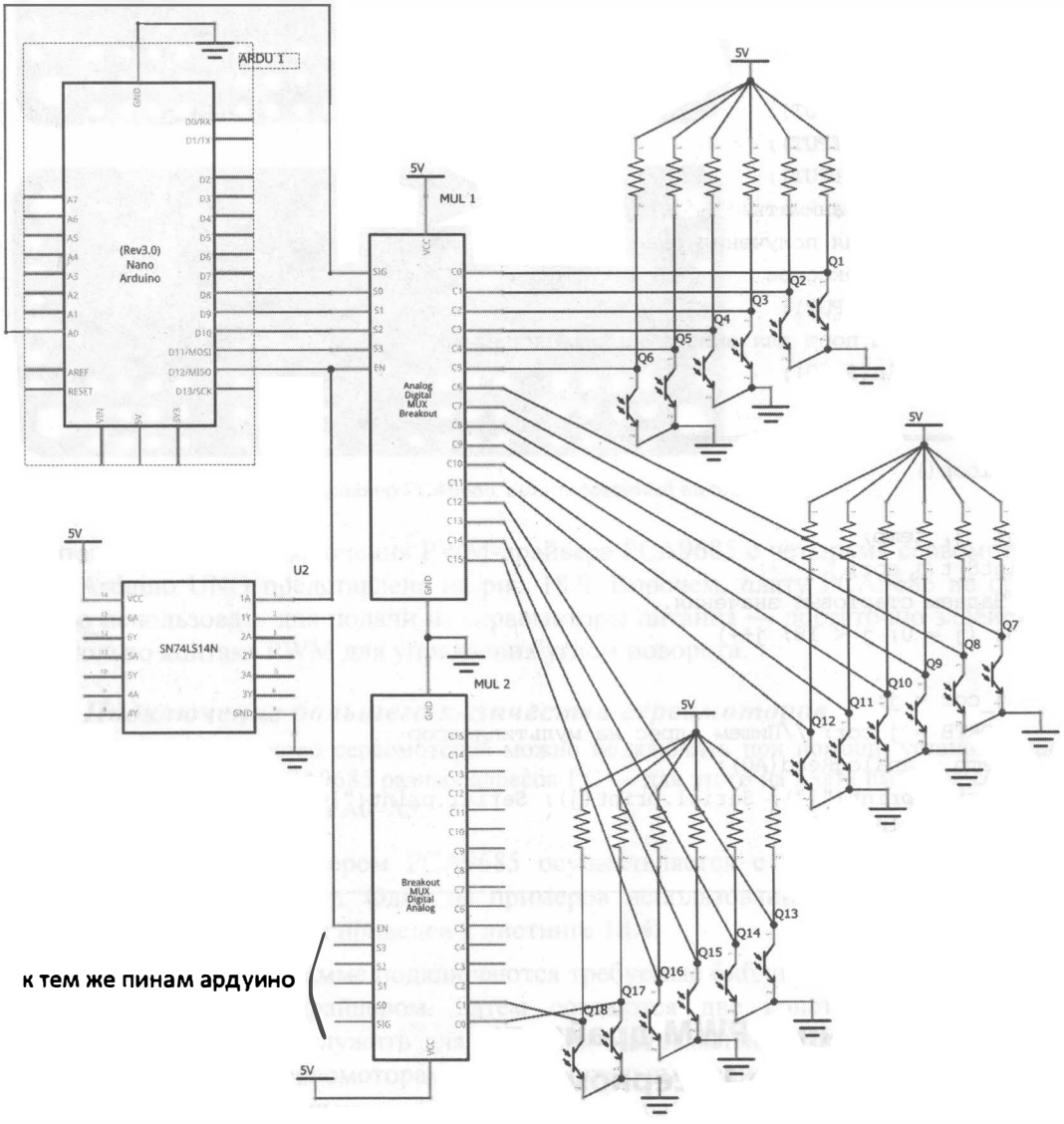


Рис. 18.7. Схема с двумя 16-канальными мультиплексорами и 18-ю аналоговыми датчиками освещенности на основе фототранзисторов

лученному адресу подключают к аналоговому входу A0 Arduino определенный вход C*x* с датчиков. При адресе с 0 по 15 — это входы с C0 по C15 первого мультиплексора, при адресе 16–31 — это входы с C0 по C15 второго.

Листинг 18.3. Поочередный опрос портов спаренных мультиплексоров

```
void setup()
{
// Переводим порты с 8-го по 12-й в состояние вывода
// это адресные порты
  //DDRB = B00111111;
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  //Порт 13 не занимать!!
  //A0 служит для получения аналогового сигнала
  //от мультиплексоров
  pinMode(A0, INPUT);
  //Подключаем порт для вывода информации на ПК
  Serial.begin(9600);
}
//================================
void loop()
{
  int  j, temp;
  uint8_t j_cor;
  //Задаем стартовые значения.
  for (j = 0; j < 18; j++)
  {
    j_cor = j;
    PORTB = j_cor; //Пишем адрес на мультиплексор
    temp = analogRead(A0);
    Serial.print("["); Serial.print(j); Serial.print("]=");
Serial.println(temp);
  }
  Serial.println("-------------------------------------------------");
  delay(1000);
}
```

Многоканальный PWM-драйвер: робот-андроид на 16 сервомоторах

Пусть мы хотим смастерить шагающего робота-андроида по рис. 5.1, для чего требуется подключить 16 сервомоторов.

В такой ситуации может помочь 16-канальный PWM-драйвер PCA9685 (рис. 18.8). Этот драйвер подключается к контроллеру по интерфейсу I²C и может управлять сразу 16-ю сервомоторами или другими устройствами, использующими частотную регуляцию.

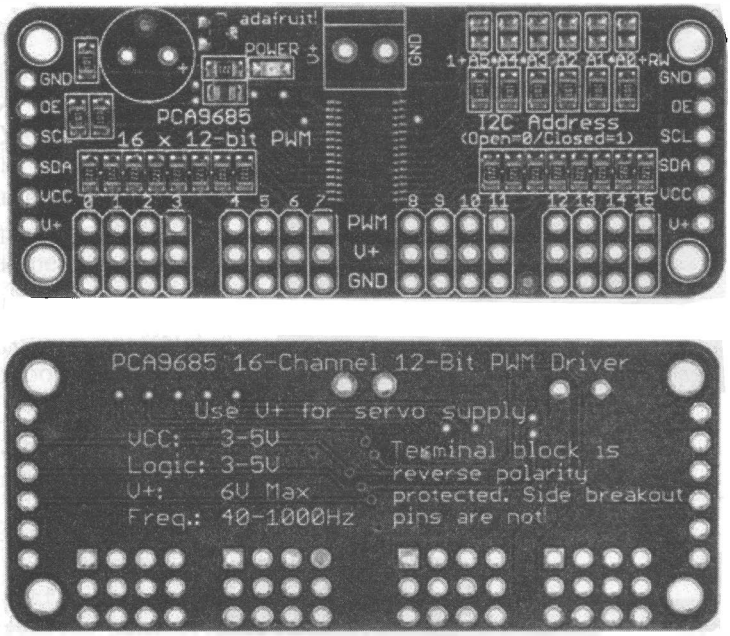


Рис. 18.8. 16-канальный PWM-драйвер PCA9685, смонтированный на плате

Монтажная схема подключения PWM-драйвера PCA9685 с четырьмя сервомоторами к Arduino UNO представлена на рис. 18.9. Впрочем, плату PCA9685 не обязательно использовать для подачи на сервомоторы питания — достаточно задействовать только контакт PWM для управления углом поворота.

Подключение большего количества сервомоторов

Большее количество сервомоторов можно подключить при помощи установки для PWM-драйвера PCA9685 разных адресов I²C — для этого на плате имеются соответствующие перемычки A0–A5.

Управление PWM-драйвером PCA9685 осуществляется с помощью библиотеки `Adafruit_PWMServoDriver.h`. Один из примеров использования ее для управления четырьмя сервомоторами приведен в листинге 18.4.

Первоначально в программе подключаются требуемые библиотеки: для управления шиной I²C и PWM-драйвером. Затем создаются две константы: `maxservo` и `minservo` — они будут служить для хранения экстремальных значений длины импульса управления сервомоторами в относительных единицах (для ограничения угла поворота). Значения `maxservo` и `minservo` для разной частоты управления могут отличаться (в программе они подобраны экспериментально — для частоты 70 Гц и сервомотора sg90).

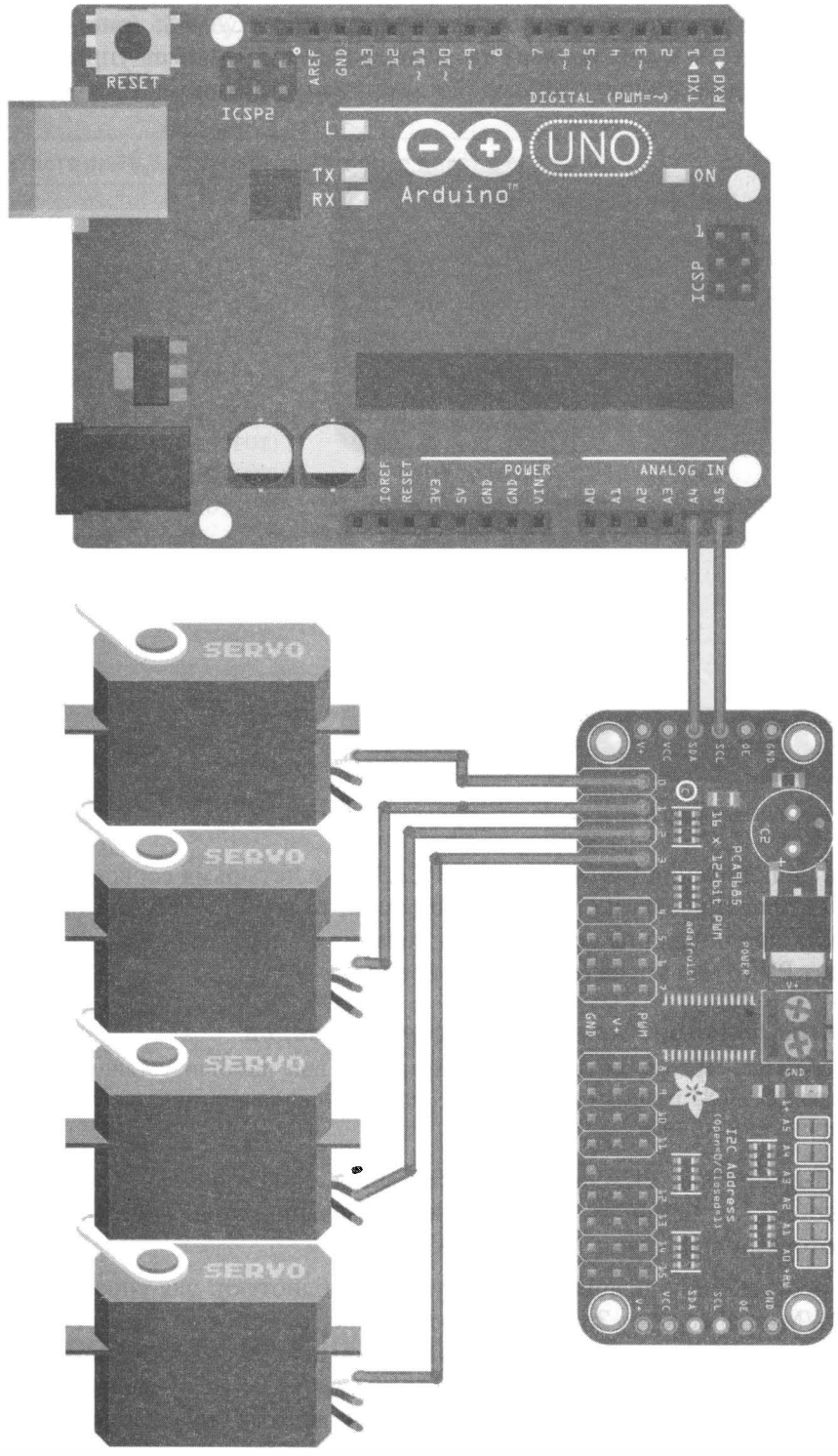


Рис. 18.9. Монтажная схема подключения PWM-драйвера с четырьмя сервомоторами к Arduino UNO

Экстремальные значения длины импульса управления

Угол поворота сервомотора регулируется шириной управляющих импульсов. В нашем случае заданная частотой управления длина такта управления делится на 4096 отрезков. Таким образом `maxservo` представляет максимальную длину импульса управления в этих отрезках, а `minservo` — минимальную. В зависимости от длины импульса управления в такте управления сервомотор решает, на какой угол он должен быть повернут.

Далее в программе создается объект `"серводрайвер"`, который будет служить для управления всеми шестнадцатью выходами драйвера PCA9685. А в функции `setup()` задается частота управления (70 Гц) для сервомоторов, установленных в контакты с 0 по 3, и подается команда генерации импульсов заданной частоты: `pwm.setPWM(i, 0, (maxservo - minservo) / 2)`, по длительности достаточная для перевода валов моторов в среднее положение.

В функции `Loop()` сервомоторы начинают двигаться ежесекундно от максимального: `pwm.setPWM(i,0,maxservo)` до минимального: `pwm.setPWM(i,0,minservo)` значения и обратно.

Листинг 18.4. Управление четырьмя сервомоторами с помощью PWM-драйвера PCA9685

```
#include <Wire.h>
#include "Adafruit_PWMServoDriver.h"
#define maxservo 700
#define minservo 180
// Создаем объект "серводрайвер"
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
int i;
void setup() {
  pwm.begin();
  pwm.setPWMFreq(70);  // Задаем частоту
  // Устанавливаем сервомоторы в центральное положение.
for (i = 0; i < 4; i++)  pwm.setPWM(i, 0, (maxservo - minservo) / 2);
}
//=================================
void loop() {
  delay(1000);
  for (i = 0; i < 4; i++)    pwm.setPWM(i, 0, maxservo);
  delay(1000);
  for (i = 0; i < 4; i++)    pwm.setPWM(i, 0, minservo);
}
```

Универсальное решение: два контроллера Arduino в связке

Можно предложить и универсальное решение — использовать контроллер Arduino в качестве подчиненного устройства или сложного датчика.

Самостоятельно создать сложный датчик или подчиненное устройство, связанное с главным контроллером по интерфейсу I²C, также возможно. Монтажная схема подобной связки приведена на рис. 18.10.

В качестве подчиненного лучше использовать небольшой контроллер вроде Arduino Nano или Arduino Pro Mini. Два таких устройства связываются по интерфейсу I²C. В подчиненный контроллер загружается программа, подобная приведенной в листинге 18.5, а главный контроллер должен в своем коде иметь строки запроса информации с подчиненного (листинг 18.6).

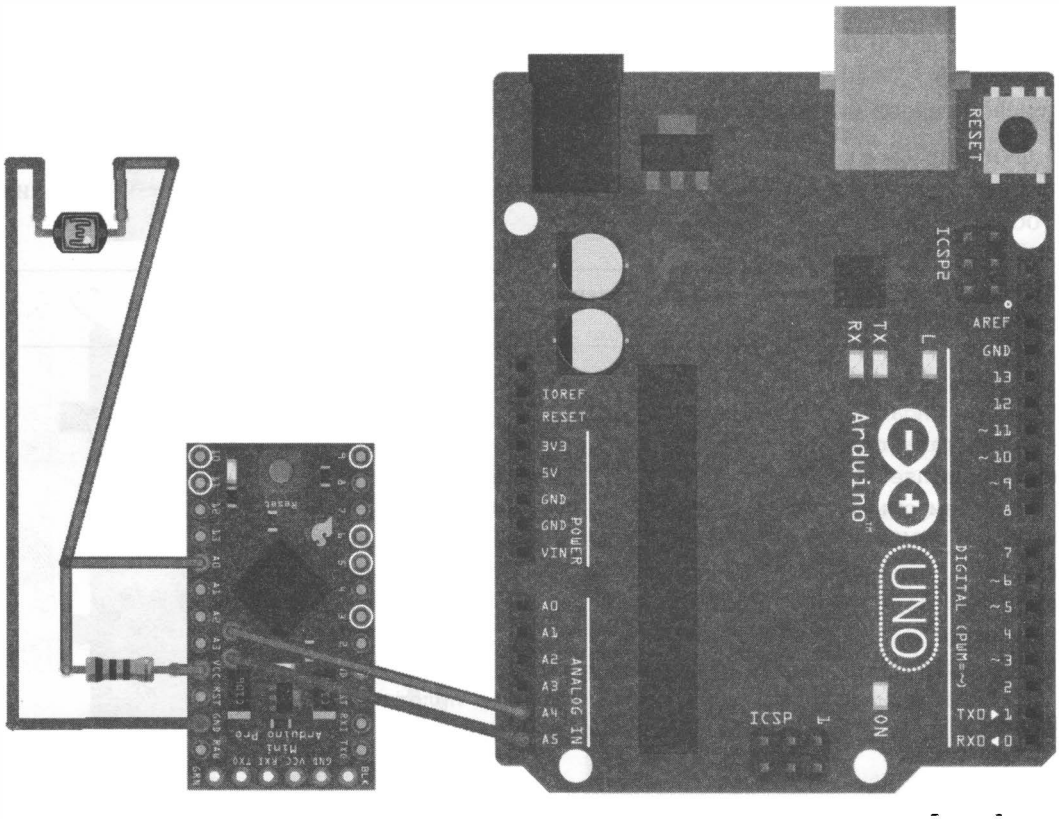


Рис. 18.10. Монтажная схема подключения Arduino Pro Mini в качестве I²C-датчика освещенности к Arduino UNO

Листинг 18.5. Создание подчиненного устройства на шине I²C (44-й адрес)

```
//=Подключение библиотеки i2c
#include <Wire.h>
//================================================
// константа с адресом i2c-датчика
#define sensor_address 44
//================================================
uint8_t wire_array[2];// Массив данных для передачи по wire.
```

```
// Функция, которая выполняется, когда приходит запрос от мастера.
void requestEvent()
{
  //Передача массива из 2 байтов.
  Wire.write(wire_array, 2);
}
void setup()
{
  // Инициализация в качестве подчиненного с i2c адресом 44.
  Wire.begin(sensor_address);
  // Ссылка на функцию, выполняемую при запросе данных.
  Wire.onRequest(requestEvent);
}
//============================
void loop()
{
  int analogData = analogRead(A0);
  //Заполнение массива
  wire_array[0] = (uint8_t)((0xFF00 & analogData) >> 8);
  wire_array[1] = (uint8_t)(0x00FF & analogData);
}
```

Листинг 18.6. Управление платой расширения на основе модуля Arduino по шине I²C

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA)
#include <Wire.h>
// константа с адресом i2c-датчика
#define sensor_address 44
//== Функция опроса датчика с 44 адресом i2c ==========
int Data_from_i2c_arduino()
{
  int  illuminating_intensity;
  // Запрос 2 байтов.
  Wire.requestFrom(sensor_address, 2, true);
  illuminating_intensity = Wire.read() | Wire.read() << 8;
  return illuminating_intensity;
}
//=====================================================
void setup()
{
  Wire.begin();
  Serial.begin(9600);
}
//=== Основная программа.=====
void loop()
```

```
{
  int i_i = Data_from_i2c_arduino();
  Serial.print("illuminating_intensity = "); Serial.println(i_i);
  delay(1000);
}
```

Подключаем шаговые двигатели

Если в позиционировании мобильного робота нужна высокая точность, то при использовании двигателей постоянного тока, особенно подключенных на нестабилизированный источник питания, могут возникнуть проблемы. Это связано с тем, что, разряжаясь, аккумуляторы «слабнут» — начинают отдавать меньший ток и понижать напряжение, что приводит к уменьшению мощности двигателей и снижению скорости вращения их валов. Частично проблему можно решить, подключая двигатели через стабилизатор. Но есть и другие решения, одним из которых является использование шаговых двигателей.

Шаговый двигатель конструктивно выполнен таким образом, что может вращать вал дискретными приращениями — «шагами». В монтажной схеме на рис. 18.11 представлен шаговый двигатель с двумя обмотками (четыре соединительных провода). В качестве драйвера для подобного двигателя можно использовать уже знакомый нам L298N, но предпочтителен специализированный драйвер EasyDriver V44 A3967.

Драйвер EasyDriver V44 A3967 позволяет управлять не только направлением вращения и количеством сделанных шагов, но и величиной шага — угла поворота за одну команду «сделать шаг». Именно в таком режиме он и подключен в схеме на рис. 18.11. При самом простом подключении достаточно использовать два порта Arduino для управления сигналами STEP (сделать шаг) и DIR (направление шага). Дополнительно подключены контакты: SLEEP — включение драйвера, ENABLE — подача напряжения на обмотки, MS1 и MS2 — величина шага. Для работы EasyDriver V44 A3967 требуется напряжение питания от 7 до 30 вольт.

В листинге 18.7 приведен пример управления шаговым двигателем, подключенным согласно схеме, приведенной на рис 18.11. Константы `dirpin`, `steppin`, `MS1`, `MS2`, `Enable` и `Sleep` задают номера портов Arduino, к которым подключены соответствующие входы драйвера. В функции `Setup()` драйвер подготавливается к работе, задаются режимы работы используемых портов, включается драйвер, задается режим его работы. В функции `Loop()` за счет генерации импульсов на входе `steppin` подаются команды двигателю делать шаги, а `dirpin` — изменяет направление. При этом двигатель делает 400 шагов в одном направлении, а затем столько же в обратном.

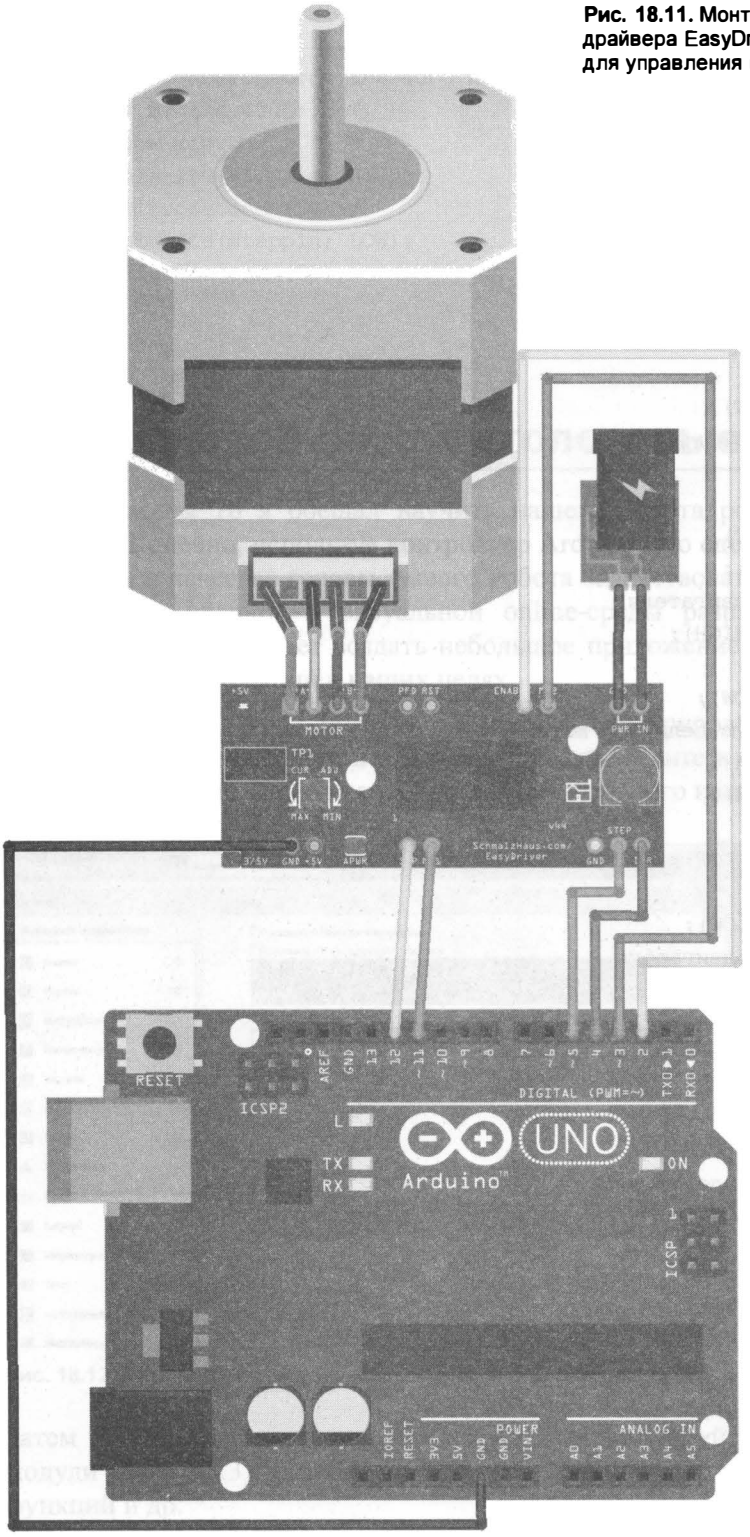


Рис. 18.11. Монтажная схема подключения драйвера EasyDriver V44 A3967 к Arduino UNO для управления шаговым двигателем

Листинг 18.7. Управление шаговым двигателем

```
// константы
#define  dirpin 4
#define  steppin 5
#define MS1 11
#define MS2 3
#define Enable 2
#define Sleep 12
//====================
void setup()
{
  pinMode(dirpin, OUTPUT);
  pinMode(steppin, OUTPUT);
  pinMode(MS1, OUTPUT);
  pinMode(MS2, OUTPUT);
  pinMode(Enable, OUTPUT);
  pinMode(Sleep, OUTPUT);
  // Отключаем обмотки двигателя.
  digitalWrite(Enable, HIGH);
  //= Разбудим драйвер.
  digitalWrite(Sleep, LOW);
  //= Зададим минимальную величину шага.
  digitalWrite(MS1, HIGH);
  digitalWrite(MS2, HIGH);
  //= Ждем шага.
  digitalWrite(steppin, LOW);
  //Задали направление.
  digitalWrite(dirpin, LOW);
  // Подключаем обмотки двигателя.
  digitalWrite(Enable, LOW);
}
//==========================================
void loop()
{
  int i;
  // Устанавливаем направление.
  digitalWrite(dirpin, LOW);
  delay(100);
  // итерации повторяются для 4000 микрошагов.
  for (i = 0; i < 4000; i++)
  { //Делаем шаг.
    digitalWrite(steppin, HIGH);
    delayMicroseconds(500);
    digitalWrite(steppin, LOW);
    delayMicroseconds(50);
  }
```

```
  digitalWrite(dirpin, HIGH); // Смена направления.
  delay(100);
  // итерации повторяются для 4000 микрошагов.
  for (i = 0; i < 4000; i++)
  { //Делаем шаг.
    digitalWrite(steppin, HIGH);
    delayMicroseconds(500);
    digitalWrite(steppin, LOW);
    delayMicroseconds(50);
  }
}
```

Робот, выполняющий голосовые команды

В конце *главы 16* я обещал научить нашего робота реагировать на голосовые команды. Конечно, используя контроллер Arduino, это сделать довольно трудно, но мы можем в качестве головы нашего робота задействовать смартфон. Тогда, с помощью App Inventor — визуальной online-среды разработки приложений для Android, несложно будет создать небольшое приложение, использующее встроенные в смартфон функции в наших целях.

Для работы в App Inventor нужно зарегистрироваться на сайте **http://appinventor.mit.edu/**. Зарегистрировавшись, перейдите в среду создания приложений, создайте новое приложение и подберите для него компоненты (рис. 18.12).

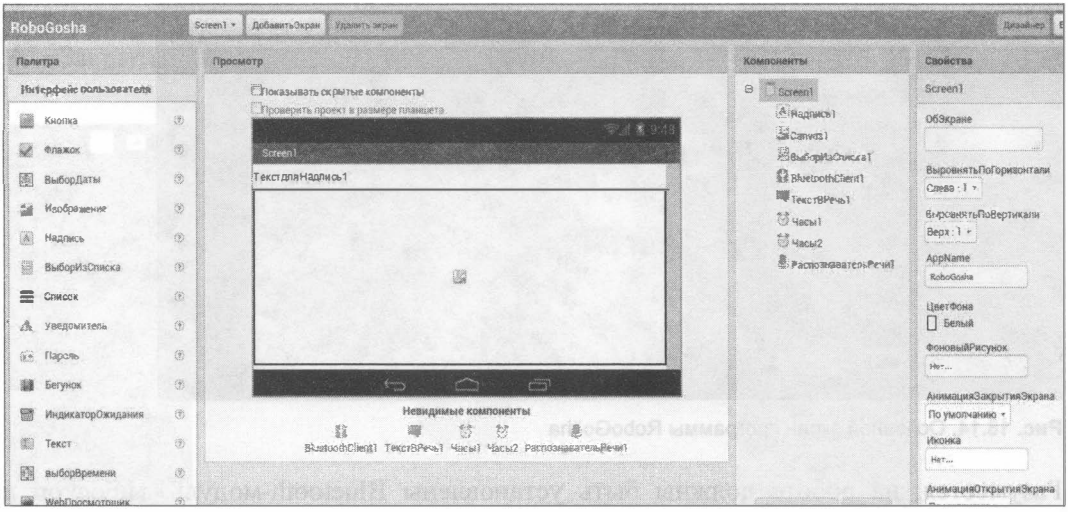


Рис. 18.12. App Inventor: создание приложения (выбор компонентов)

Затем при помощи визуальных элементов сконструируйте основные программные модули (рис. 18.13): реакции на события, графических эффектов, вызова внешних функций и др.

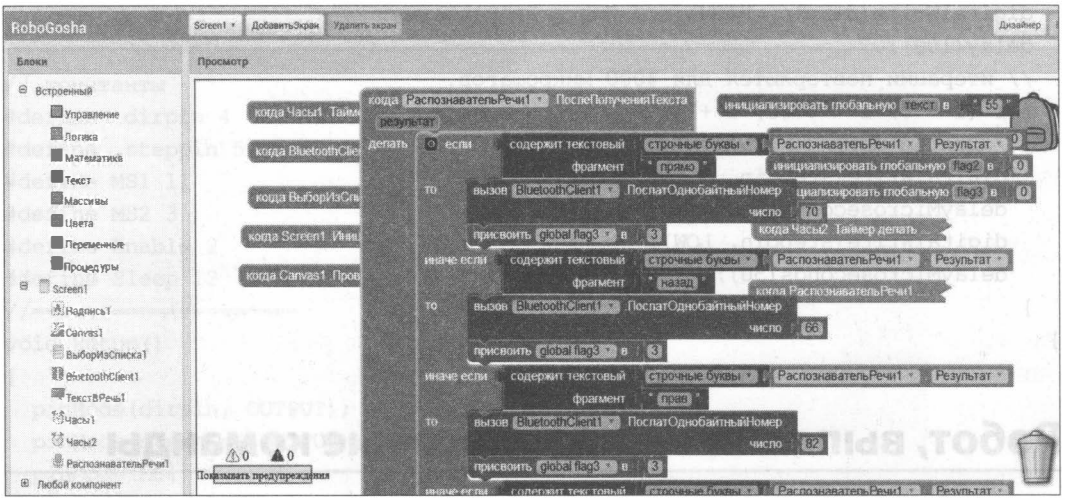


Рис. 18.13. App Inventor: создание приложения (визуальное конструирование алгоритмов)

Итак, создадим программу RoboGosha (рис. 18.14), которая связывается с роботом по Bluetooth, ждет голосовую команду и распознанные команды преобразует в команды управления роботом.

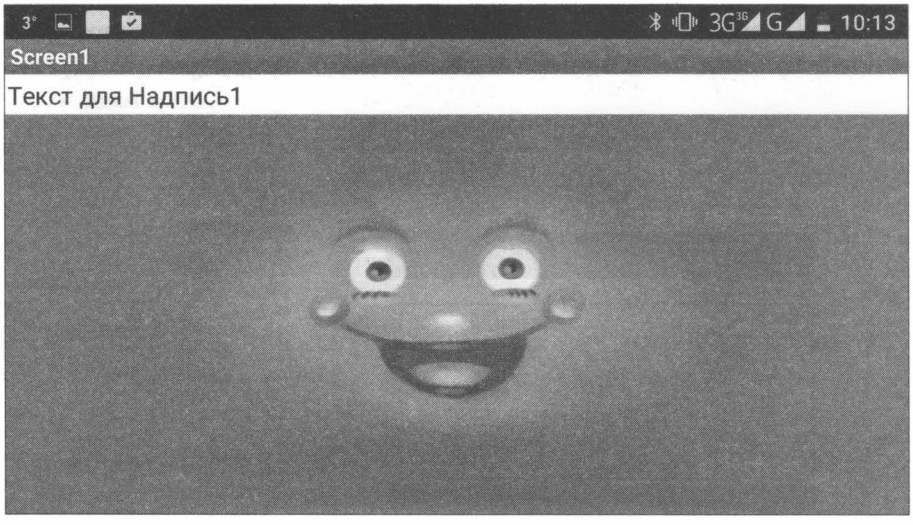


Рис. 18.14. Основной экран программы RoboGosha

Разумеется, на роботе должны быть установлены Bluetooth-модуль, гироскоп и компас.

Перед запуском программы следует выполнить на смартфоне действия по регистрации робота (см. *главу 8*) и включить Bluetooth. Когда программа загрузится, прикоснитесь к экрану — появится окно выбора Bluetooth-соединений, в котором следует выбрать вашего робота (рис. 18.15).

Рис. 18.15. Выбор Bluetooth-соединения для программы RoboGosha

Когда робот и смартфон соединятся, что занимает несколько секунд (в это время экран черный), смартфон произнесет фразу: «Жду команды». Нужно дать ему одну из следующих команд (рис. 8.16):

- «Ехать прямо» — в течение небольшого времени робот двигается вперед;
- «Ехать назад» — в течение небольшого времени робот двигается задним ходом;
- «Повернуть налево» — робот поворачивается на 90 градусов влево;
- «Повернуть направо» — робот поворачивается на 90 градусов вправо;
- «Закрыть программу» — эта команда закрывает приложение.

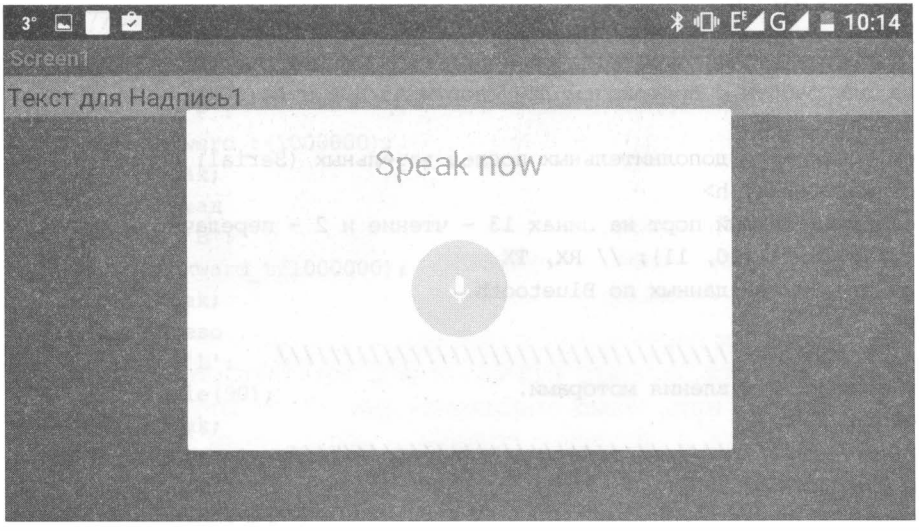


Рис. 18.16. RoboGosha ждет команды

Если RoboGosha в течение некоторого времени не получает команду, то он высвечивает сообщение: **Ничего не найдено**, и вам придется нажать на кнопку **Повторить** (рис. 18.17).

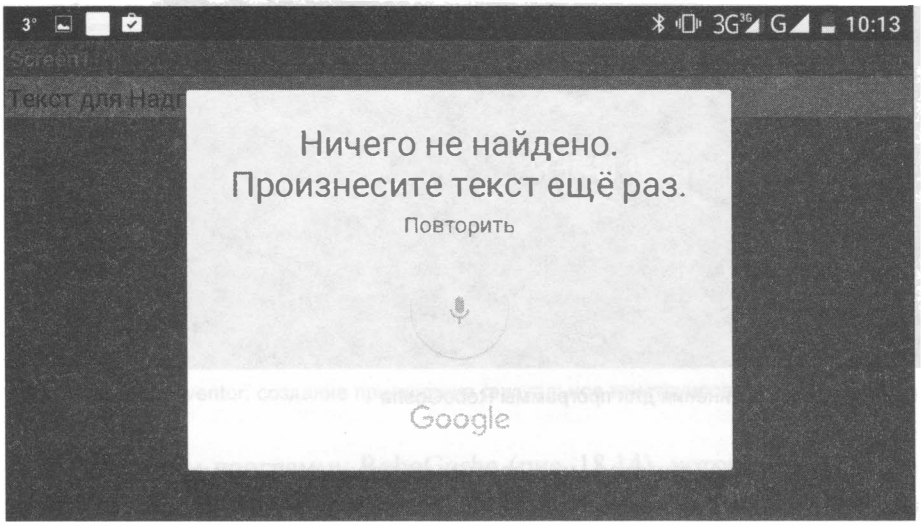


Рис. 18.17. RoboGosha не дождался команды

Прикрепите смартфон к роботу вместо головы, например, при помощи резинок, и вы получите интерактивного робота.

Предварительно в робота следует загрузить программу из листинга 18.8, которая ждет команды от смартфона, и если команда соответствует одному из пунктов оператора `switch`, выполняет ее. Конечно, в каталоге с программой должны находиться файлы библиотек: motors.h, compas.h и gyro_acsel.h.

Листинг 18.8. Робот, управляемый по Bluetooth программой RoboGosha

```
// Библиотека для работы с протоколом I2C (порты A5/SCL и A4/SDA).
#include <Wire.h>
// Подключаем библиотеку дополнительных последовательных (Serial) портов.
#include <SoftwareSerial.h>
//Создаем последовательный порт на пинах 13 - чтение и 2 - передача.
SoftwareSerial BTSerial(10, 11); // RX, TX
// Переменная для приема данных по Bluetooth.
char bt_input;
////////////////////////////////////////////////////////////
//Подключаем функции управления моторами.
#include "motors.h"
////////////////////////////////////////////////////////////
//Подключаем функции HMC5883L.
#include "compas.h"
////////////////////////////////////////////////////////////
```

```
//Подключаем функции MPU-6050.
#include "gyro_acsel.h"
////////////////////////////////////////////////////////////
void setup()
{
  BTSerial.begin(9600);
  // Заносим в переменные номера контактов (пинов) Arduino.
  // Для левых и правых моторов.
  setup_motor_system(2, 3, 4, 5);
  // Двигатели остановлены.
  _stop();
  giroscop_setup();
  //Компас.
  compas_setup();       // инициализация HMC5883L
  delay(150);
  Ang_ = 0;
  //Расчет компенсации утечки нуля гироскопа.
  Calc_CompensatorZ(1000);
  // Запуск таймера для гироскопа.
  timer = micros();
}
//==================================
void loop()
{
  while (1)
  {
    if (BTSerial.available())
    {
      // Читаем команды и заносим их в переменную.
      bt_input = (char)BTSerial.read();
      // Отправляем команду в порт, чтобы можно было ее проверить.
      //Serial.println(bt_input);
      switch (bt_input) {
        // Вперед
        case 'F':
          forward_t(1000000);
          break;
        // Назад
        case 'B':
          backward_t(1000000);
          break;
        // Влево
        case 'L':
          Angle(90);
          break;
        // Вправо
        case 'R':
          Angle(-90);
          break;
```

```
        // Разворот
        case 'O':
          Angle(180);
          break;
        // Стоп
        case 'S':
          _stop();
        case 'X':
          //switch_rejim = 1;
          break;
        case 'x':
          //switch_rejim = 0;
          break;
        case 'D':
          _stop(); //switch_rejim = 0;
          break;
      }
    }
    time_gyro(1);
  }
}
```

Рука для робота

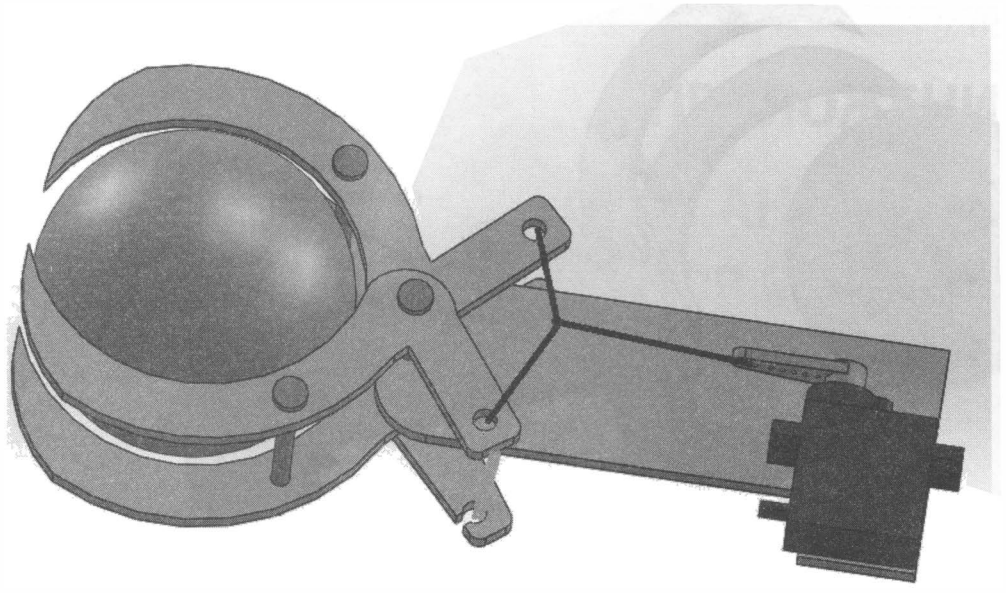
Если прикрепить к мобильному роботу захват, то его функционал значительно расширится, — например, можно будет устроить матч по робобаскетболу. Сделать захват самостоятельно вполне по силам — для этого потребуется один маломощный сервомотор, три 3-мм винта длиной 50 мм, несколько гаек и шайб, резинка, кусочек 3-мм фанеры, и немного поработать лобзиком и дрелью. В результате получится захват для небольшого мяча.

> ***Электронный архив***
>
> Чертежи такого захвата вы найдете в сопровождающем книгу электронном архиве (см. *приложение 2*).

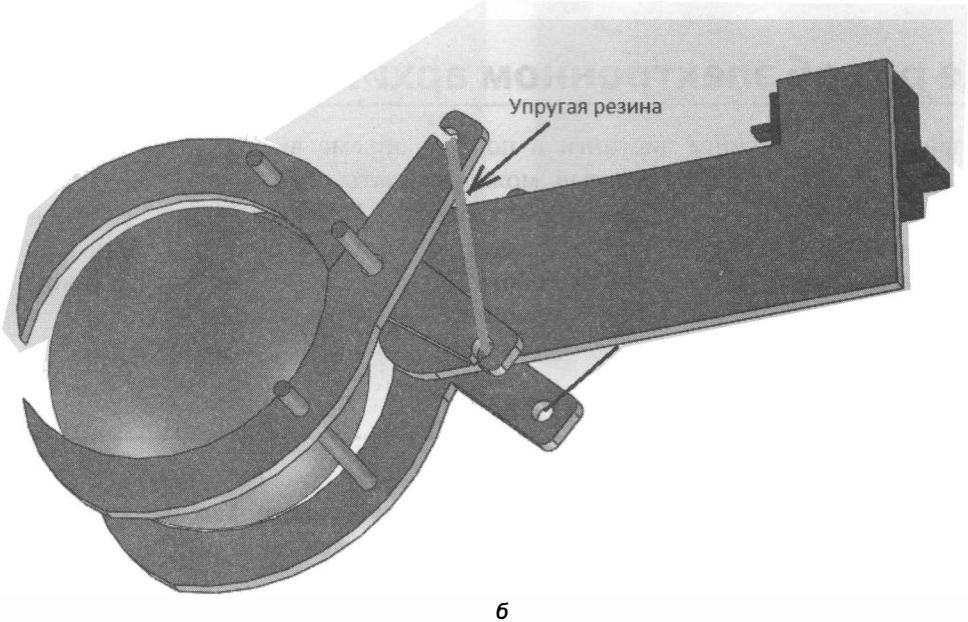
Сервомотор закрепите прочным клеем (подойдет суперклей или термоклей). Винты закрепите гайками, а чтобы они не раскручивались, можно использовать лак (хотя бы и маникюрный), нанеся его на резьбу под гайку.

Работает захват следующим образом. Когда рычажок на валу сервомотора повернут к захвату, нить ослабляется, и захват под силой резинки сжимается (рис. 18.18). Когда рычажок натягивает нить, захват открывается (рис. 18.19). Нить должна быть прочной и не растягиваться, а натяжение резинки — достаточным для сжатия захвата, но допускающим его открытие при работе сервомотора.

Если такие захваты прикрепить к нескольким роботам, можно устроить захватывающий матч среди роботов, управляемых удаленно через Bluetooth.



а



б

Рис. 18.18. Захват в закрытом состоянии: *а* — вид сверху; *б* — вид снизу

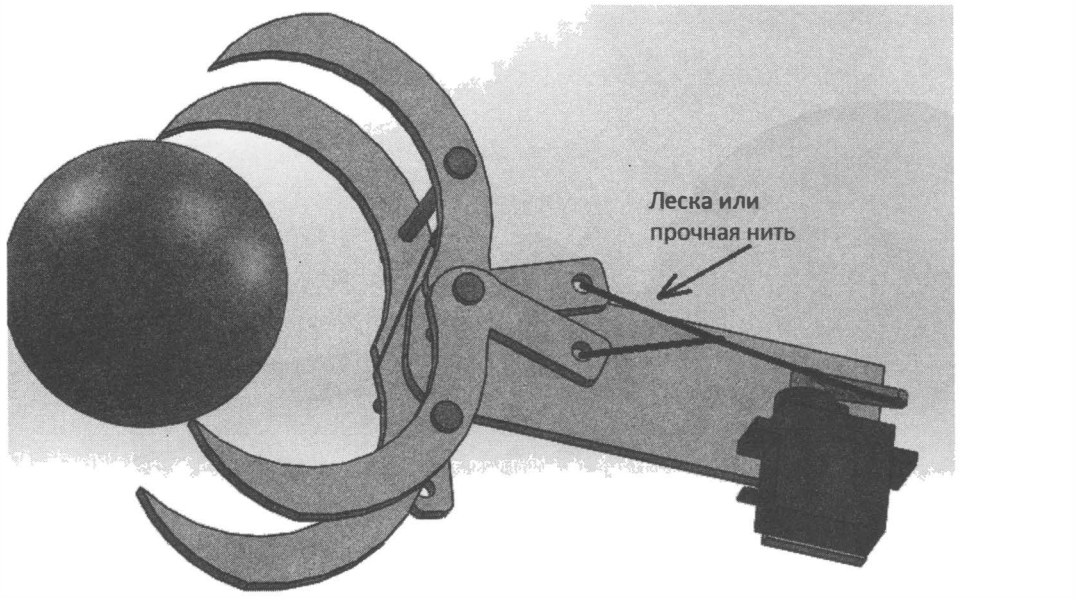


Рис. 18.19. Захват в открытом состоянии: вид сверху

Еще раз об электронном архиве

Все приведенные в книге листинги в полном объеме включены в электронный архив (см. *приложение 2*), который можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538619.zip** или со страницы книги на сайте **www.bhv.ru**. Кроме листингов, в архив включены библиотеки, используемые при программировании робота, а также другие материалы, о которых в книге имеются упоминания о размещении их в электронном архиве.

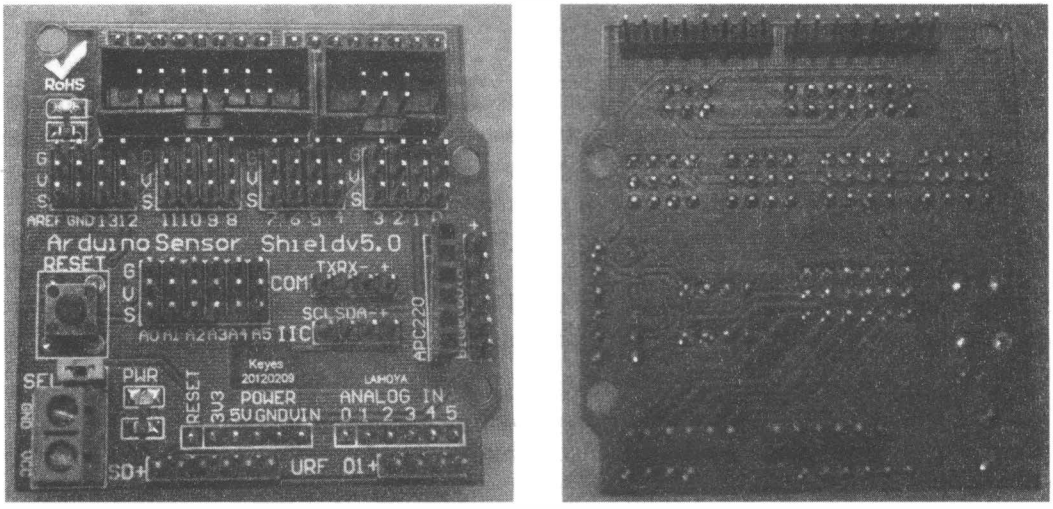
Как связаться с автором?

- Я снимаю видеоролики про роботов и выкладываю их на свой YouTube-канал по адресу: **https://www.youtube.com/user/momotmvu/videos**. Там же можно задавать мне вопросы по существу роликов, я на них отвечу.
- По электронной почте: **momotmvu@yandex.ru**.
- Через сайт **http://zizibot.ru** — это сайт развивающейся лаборатории робототехники, на нем я также оставляю статьи и ссылки на ролики.
- Через социальные сети «ВКонтакте»: **http://vk.com/id161424404** или Facebook: **https://www.facebook.com/profile.php?id=100001169240434**.

ПРИЛОЖЕНИЕ 1

ОПИСАНИЕ ПЛАТЫ ARDUINO SENSOR SHIELD V5.0

Сервисная плата Arduino Sensor Shield v5.0 (рис. П1.1) создана с целью организации комфортного подключения к Arduino UNO различных датчиков, маломощных сервомоторов, приемопередатчиков, дисплеев и т. п.



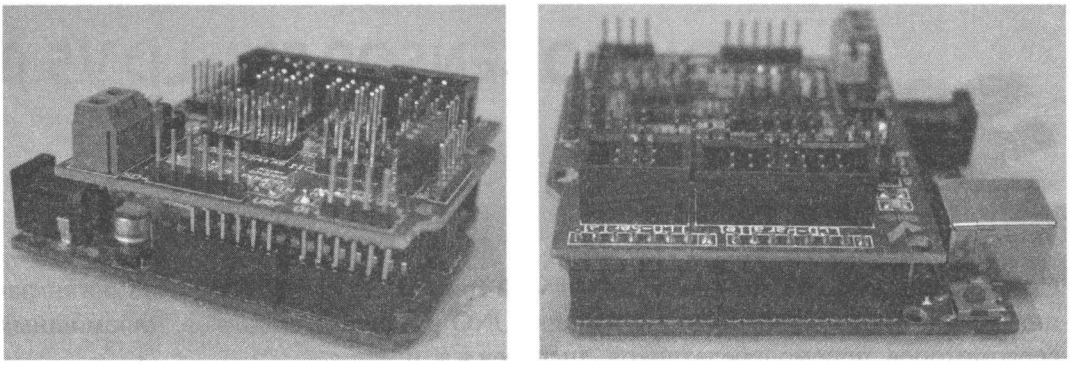
а *б*

Рис. П1.1. Сервисная плата Arduino Sensor Shield v5.0: *а* — вид сверху; *б* — вид снизу

Благодаря удобному расположению контактных площадок и поддержке возможности подключения внешнего источника питания, плата Arduino Sensor Shield v5.0 стала очень популярной, — в отличие от своей предшественницы Arduino Sensor Shield v4.0, которая внешнего питания не поддерживала (только от стабилизатора платы Arduino).

Плата с нижней стороны имеет набор штырьков, расположение которых полностью соответствует коммутационным разъемам на плате Arduino UNO (рис. П1.1, *б*).

Перед использованием плата должна быть установлена точно в указанные разъемы (рис. П1.3) — неточная установка или смещение хотя бы на один разъем приведет к неработоспособности платы и станет причиной выхода из строя датчиков либо самой платы Arduino. Штырьки платы Arduino Sensor Shield v5.0 должны быть утоплены в соответствующие соединительные отверстия платы Arduino UNO до основания.



а *б*

Рис. П1.2. Установка платы Arduino Sensor Shield v5.0 на Arduino UNO:
а — начальный момент; *б* — установка завершена

Плата Arduino Sensor Shield v5.0 расширяет возможности использования платы Arduino и позволяет, помимо информационных контактов, подключить к ней множество питаемых компонентов, что особенно удобно при подключении сервомоторов, для которых выделен целый ряд разъемов, состоящих из трех клемм: G — «земля», V — питание, S — сигнал. Но эти контактные ряды приспособлены не только для сервомоторов — в них можно подключать, например, датчики или светодиоды. Более того, цифровые порты D0–D13 и аналоговые порты A0–A5, помимо указанных рядов контактных площадок, еще и продублированы в специализированных разъемах.

Для выбора источника питания плата имеет перемычку. Если она замкнута, то питание плат Arduino и Arduino Sensor Shield v5.0 возможно через стабилизированный внешний источник (болтовые зажимы GND и VCC на плате), в противном случае питание подается от платы Arduino.

Соответствие портов и разъемов платы Arduino Sensor Shield v5.0 наглядно представлено на рис. П1.3. Например, цифровой порт D02 выведен одновременно на три разъема: в параллельном интерфейсе, в последовательном интерфейсе, а также в ряду разъемов G-V-S. Но это не означает, что во все три выхода D02 можно включить одновременно различные устройства. Подобное возможно только на подключениях типа «общая шина»: SPI и I^2C, D10–D13 и A4–A5.

К недостаткам платы можно отнести наличие всего двух пар разъемов для портов D4–D5, что затрудняет подключение дополнительных устройств на общую шину I^2C.

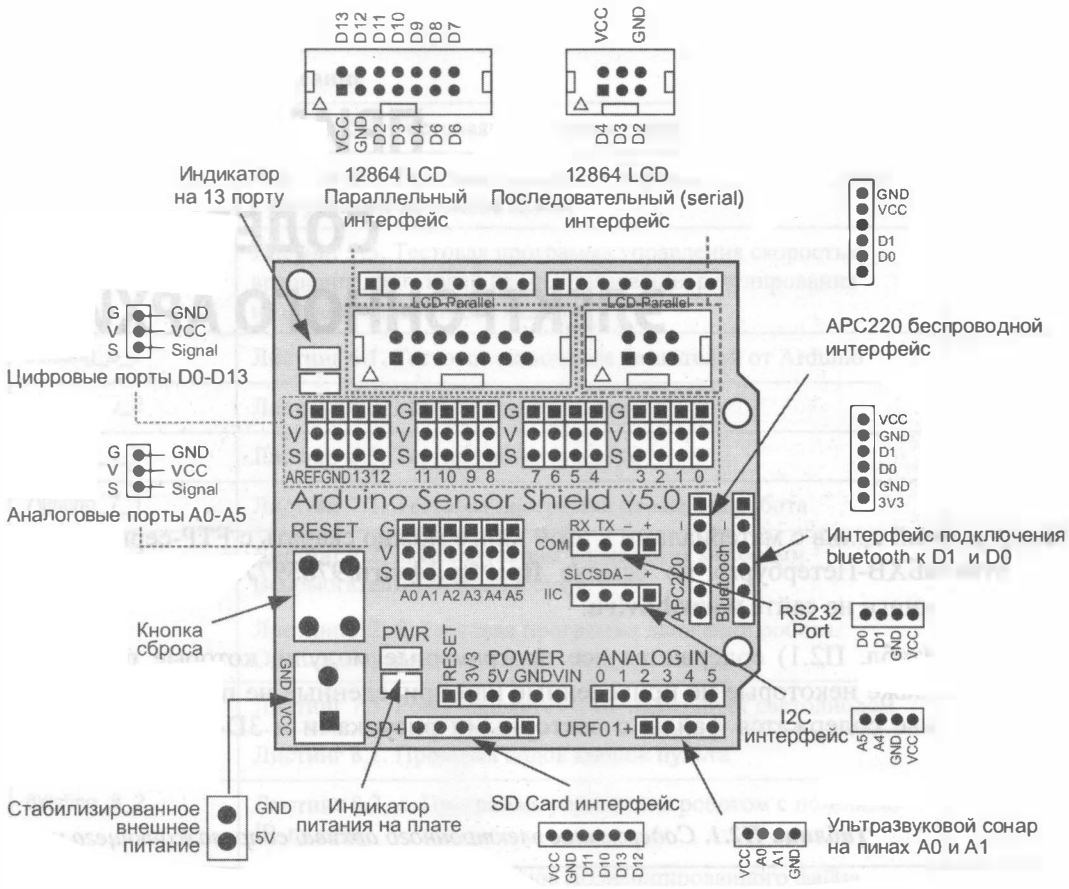


Рис. П1.3. Плата Arduino Sensor Shield v5.0

Порядок подключения платы Arduino Sensor Shield v5.0 следующий:

- установить плату на Arduino UNO, внимательно следить за соответствием клеммных площадок Arduino и Arduino Sensor Shield v5.0;
- определить, какой источник питания будет использоваться, — если предполагается питание от платы Arduino, перемычку следует разомкнуть;
- подключить необходимые информационные и питающие разъемы согласно пояснениям и рис. П1.3.

ПРИЛОЖЕНИЕ 2

СОДЕРЖАНИЕ ЭЛЕКТРОННОГО АРХИВА

Электронный архив с материалами к этой книге можно скачать с FTP-сервера издательства «БХВ-Петербург» по ссылке **ftp://ftp.bhv.ru/9785977538619.zip** или со страницы книги на сайте **www.bhv.ru**.

В архиве (табл. П2.1) содержатся все программные модули, которые приведены в книге, и даже некоторые не приведенные или приведенные не полностью. Кроме них в архиве содержатся файлы с векторными рисунками и 3D-моделью «руки» робота.

Таблица П2.1. *Содержание электронного архива, сопровождающего книгу*

Папки	Описание	Главы
4\listing_4_1	Листинг 4.1. Программа мигания светодиодом	4
4\listing_4_2	Листинг 4.2. Объявление переменных. Зоны видимости. Сообщение об ошибке	4
4\listing_4_3	Листинг 4.3. Получение данных от компьютера через порт ввода/вывода	4
4\listing_4_5	Листинг 4.5. Программа управления миганием светодиода с ПК	4
4\listing_4_6	Листинг 4.6. Программа управления миганием светодиода с ПК с использованием оператора `switch ... case`	4
4\listing_4_7	Листинг 4.7. Программа управления миганием светодиодом с ПК с использованием оператора цикла `while`	4
4\listing_4_8	Листинг 4.8. Пример использования оператора цикла `for` (на 300 повторений)	4
4\listing_4_9	Листинг 4.9. Пример написания и использования функции	4
4\listing_4_10	Листинг 4.10. Пример управления сервомотором	4

Таблица П2.1 (продолжение)

Папки	Описание	Главы
5\listing_5_1	Листинг 5.1. Тестовая программа управления двигателями	5
5\listing_5_2	Листинг 5.2. Тестовая программа управления двигателями с регуляцией на основе ШИМ	5
5\listing_5_3	Листинг 5.3. Тестовая программа управления скоростью вращения двигателей без использования регулирования на основе аппаратного ШИМ	5
6\listing_6_1	Листинг 6.1. Тестовое включение двигателей от Arduino	6
6\listing_6_2	Листинг 6.2. Мигаем светодиодом	6
6\listing_6_3	Листинг 6.3. Тестирование зуммера	6
7\listing_7_1	Листинг 7.1. Тестовая программа движений робота	7
7\listing_7_2	Листинг 7.2, *а*. Все функции управления моторами. Вспомогательный файл motor.h Листинг 7.2, *б*. Тестовая программа движений робота. Основной файл программы listing_7_2.ino	7
7\listing_7_3	Листинг 7.3. Робот движется и сигнализирует светодиодом	7
8\Listing_8_1	Листинг 8.1. Проверка кодов кнопок пульта	8
8\listing_8_2	Листинг 8.2, *а*. Программа управления роботом с помощью IR-пульта Листинг 8.2, *б*. Содержимое модифицированного файла motor.h	8
8\Listing_8_3	Листинг 8.3. Переименование робота и проверка работы Bluetooth	8
8\Listing_8_3_cppc	Листинг 8.3_cppc. Переименование робота и проверка работы Bluetooth	8
8\listing_8_4	Листинг 8.4, *а*. Управление роботом по Bluetooth-каналу Листинг 8.4, *б*. Содержимое файла move_case.h	8
9\listing_9_1	Листинг 9.1. Тестовая программа проверки подключения датчиков линии	9
9\listing_9_2	Листинг 9.2. Движение робота по черной линии	9
10\Listing_10_1	Листинг 10.1. Содержимое файла sonar.h	10
10\Listing_10_2	Листинг 10.2. Программа измерения расстояний	10
10\Listing_10_3	Листинг 10.3. Программа поворота головы робота в разные стороны	10
10\Listing_10_4	Листинг 10.4. Установка сервомотора в среднее положение	10

Таблица П2.1 (продолжение)

Папки	Описание	Главы
11\listing_11_1	Листинг 11.1. Программа обхода препятствий роботом	11
11\listing_11_1m	Листинг 11.1_1m. Программа обхода препятствий роботом	11
11\listing_11_2	Листинг 11.2. Отладочная (неполная) программа обхода препятствий с выводом в порт	11
12\listing_12_1	Листинг 12.1. Программа движения вдоль правой стенки с помощью ультразвукового датчика	12
12\listing_12_3	Листинг 12.3. Программа движения робота с тремя детекторами препятствия	12
12\listing_12_4	Листинг 12.4. Модернизированная программа прохода лабиринта с использованием датчиков препятствия	12
13\listing_13_1	Листинг 13.1. Прием данных от HMC5883L и передача их на ПК	13
13\listing_13_2	Листинг 13.2. Основная программа калибровки компаса	13
13\listing_13_3	Листинг 13.3. Программа, демонстрирующая расчет угла отклонения от магнитного «севера»	13
13\listing_13_4	Листинг 13.4. Программа движения робота в северном направлении	13
14\listing_14_1	Листинг 14.1. Получение данных от MPU-6050	14
14\listing_14_3	Листинг 14.3. Программа управления движениями робота с использованием гироскопа	14
15\listing_15_1	Листинг 15.1. Робот выталкивает все кегли	15
15\listing_15_2	Листинг 15.2. Программа выбивания белых кеглей	15
16\listing_16_1	Листинг 16.1. Тестирование DFPlayer	16
16\listing_16_2	Листинг 16.2. Модифицированная программа объезда препятствий	16
17\listing_17_1	Листинг 17.1. Программа балансировки робота на основе анализа показаний гироскопа	17
17\listing_17_2	Листинг 17.2. Программа балансировки с использованием фильтра Калмана	17
17\listing_17_3	Листинг 17.3. Программа балансировки с использованием комплементарного фильтра	17
18\listing_18_1	Листинг 18.1. Пример управления сдвиговым регистром	18
18\listing_18_2	Листинг 18.2. Пример управления тремя сдвиговыми регистрами	18
18\listing_18_3	Листинг 18.3. Поочередный опрос портов спаренных мультиплексоров	18

Таблица П2.1 (окончание)

Папки	Описание	Главы
18\listing_18_4	Листинг 18.4. Управление четырьмя сервомоторами с помощью PWM-драйвера PCA9685	18
18\listing_18_5	Листинг 18.5. Создание подчиненного устройства на шине I^2C (44-й адрес)	18
18\listing_18_6	Листинг 18.6. Управление платой расширения на основе модуля Arduino по шине I^2C	18
18\listing_18_7	Листинг 18.7. Управление шаговым двигателем	18
18\listing_18_8	Листинг 18.8. Робот, управляемый по Bluetooth программой RoboGosha	18
RoboGosha	Программа для Android RoboGosha.apk	18
Захват	Составные части захвата для робота в формате DXF и STL	18

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Г

Д

Е

Ж

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Э

Я

дерзай!

www.bhv.ru/books/robot

Электроника

МОБИЛЬНЫЕ РОБОТЫ НА БАЗЕ Arduino

2-е издание

Момот Михаил Викторович, доцент кафедры информационных систем Томского политехнического университета. Увлекается робототехникой, поклонник и пропагандист проекта Arduino с 2014 года. Основатель неформального клуба робототехников «Лига роботов ЮТИ ТПУ», объединяющего школьников, студентов, преподавателей и энтузиастов.

КНИГА +
НАБОР ДЕТАЛЕЙ

www.bhv.ru/books/robot

Мобильные роботы любой сложности легко и быстро!

«Эту книгу я составлял как руководство для начинающих Конструкторов, людей, которым нравится конструировать. А за основу взял конструирование мобильных роботов на популярной платформе Arduino, позволяющей реализовывать как простейших, так и достаточно интеллектуальных роботов. Платформа открытая, изготавливать дополнительные модули для нее может любой человек или организация, то же относится и к программам.

Представленные проекты имеют единую колесную базу, но различаются системами датчиков и программным кодом. В процессе сборки вы научитесь программировать на платформе Arduino, обращаться с электронными компонентами, усвоите принципы действия датчиков, с помощью которых роботы следят за внешним миром, научитесь удаленному управлению и сможете конструировать своих оригинальных роботов».

Михаил Момот, автор книги

+

Электронный архив содержит детали робота для печати на 3d-принтере, векторные рисунки для резки лазером, листинги, дополнительные библиотеки и программы. Его можно скачать по ссылке ftp://ftp.bhv.ru/9785977538619.zip, а также со страницы книги на сайте www.bhv.ru.

bhv®

БХВ-ПЕТЕРБУРГ
191036, Санкт-Петербург, Гончарная ул., 20
Тел.: (812) 717-10-50, 339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

ISBN 978-5-9775-3861-9