РОССИЙСКАЯ АКАДЕМИЯ НАУК
Институт проблем машиноведения

Серия «Шаги в кибернетику»

С. А. Филиппов

Робототехника для детей и родителей

Под редакцией
д-ра техн. наук, проф.
А. Л. Фрадкова

Издание 3-е, дополненное и исправленное

Санкт-Петербург
«НАУКА»
2013

УДК 621.86/.87
ББК 32.816
Ф53

Филиппов С.А. Робототехника для детей и родителей. – СПб.: Наука, 2013. 319 с.

ISBN 978-5-02-038-200-8

Уже много лет мы читаем в книгах и газетах, слышим по радио и из телевизора, что скоро нас будут окружать умные, добрые и интересные роботы. Однако в реальной жизни роботов все нет и нет. Лишь несколько лет назад знаменитая датская компания Lego сделала роскошный подарок любителям мехатроники, роботов и других кибернетических игр и игрушек: выпустила робототехнический конструктор Lego Mindstorms NXT, который с успехом используется как дома, так и в учебе.

Эта книга одна из первых на русском языке поможет не только самому строить и программировать разнообразных роботов из Lego, но и научить этому других школьников, студентов. В ней рассматриваются основы конструирования, программирования на языках NXT-G, Robolab и RobotC, а также элементы теории автоматического управления.

В третьем издании добавлены описания усовершенствованых конструкций роботов, а также рассмотрены новые задачи: прохождение лабиринта, роботы-манипуляторы, инверсная линия и др. По-прежнему большое внимание уделено алгоритмам управления: от П- и ПД-регулятора для движения по линии до ПИД-регулятора для балансирующего робота-сигвея.

Предназначена для преподавателей кружков робототехники школ и вузов, для широкого круга читателей.

Рецензент
д-р техн. наук, проф. Б. Р. Андриевский

ISBN 978-5-02-038-200-8

ОГЛАВЛЕНИЕ

Предисловие

Уже много лет мы читаем в книгах и газетах, слышим по радио и из телевизора, что скоро нас будут окружать умные, добрые и интересные роботы. Однако в реальной жизни роботов все нет и нет. И так же часто в научно-технических журналах мы читаем о мехатронике — удивительной науке на стыке механики, электроники, компьютеров и теории управления (кибернетики). Однако и мехатронными устройствами ученые тоже что-то не торопятся нас окружить.

И вот несколько лет назад знаменитая датская компания Lego сделала роскошный подарок любителям мехатроники, роботов и других кибернетических игр и игрушек: выпустила робототехнический конструктор Lego Mindstorms. Из него можно собрать не только фантастические человекоподобные и другие роботы, не только разнообразные мехатронные устройства, но и приборы для измерения, связи, контроля и т.п. Главное же, этот конструктор позволяет легко и с удовольствием научиться самому строить такие штуковины и учить этому молодежь, начиная с возраста 8—10 лет. Следующее поколение киберконструктора, Lego Mindstorms NXT, обладает новыми возможностями: общение по протоколу Bluetooth, богатый набор бортовых датчиков, включая видеокамеры. Неужели скоро мы сами сможем окружить себя кибернетическими помощниками?

Проблема только в одном: нет пока на русском языке подходящих учебников для такого обучения. Однако предлагаемая вниманию читателя книга позволяет, кажется, решить и эту проблему. Из ее названия как раз и ясно, что она предназначена научить практической робототехнике детей и родителей. Причем учить этому, пользуясь советами опытного наставника, который сам прошел все этапы кибертворчества.

Сергей Александрович Филиппов имеет опыт руководства кружками робототехники в нескольких школах Санкт-Петербурга. Ведет семинары и мастер-классы для школьных учителей, методистов, для членов команд города на олимпиадах по роботам. Сам ездит на олимпиады и конференции со своими замечательными учениками[1]. Наверное, поэтому книга получилась и увлекательной, и поучительной, и доступной. Поучительной не только для детей и родителей, купивших конструктор, но и для учителей школ, руководителей кружков и преподавателей вузов, стремящихся помочь своим ученикам сделать первые шаги в мир техники будущего, в мир робототехники и мехатроники.

[1] В ноябре 2012 г. команда из Санкт-Петербурга Hand-Friend под руководством С. А. Филиппова в составе сборной России завоевала золотую медаль на Всемирной олимпиаде роботов в г. Куала-Лумпур, Малайзия, с проектом «Грета играет в ладушки». Вот имена чемпионов: Мария Муретова, Денис Никитин, Андрей Свечинский.

Удовольствие от чтения получат даже те, у кого еще пока нет киберконструктора: книга, как золотой ключик, откроет дверь в фантастическую страну кибернетических игр и игрушек, удивительно похожих на многие серьезные автоматические приборы и системы.

Мне особенно приятно, что часть описанных в книге идей и приемов родилась в ходе нашего совместного проекта «Киберфизическая лаборатория», начатого в 2008 г. физико-математическим лицеем №239 и кафедрой теоретической кибернетики математико-механического факультета СПбГУ под эгидой института проблем машиноведения Российской академии наук (ИПМаш РАН) и поддержанного программой президиума РАН «Поддержка молодых ученых» и федеральной целевой программой «Научные и научно-педагогические кадры инновационной России».

Среди других проектов отмечу Санкт-Петербургские олимпиады по кибернетике, проводимые с 1999 г. ведущими вузами города под эгидой ИПМаш РАН. Дальнейшую информацию об олимпиадах, книгах и других наших проектах можно найти на сайте *www.cyber-net.spb.ru*.

Желая книге С. А. Филиппова успеха у читателей, отмечу и то, что за ней должны последовать другие, поскольку она открывает серию научно-популярных книг и учебных пособий «Шаги в кибернетику», предназначенную как для школьников и студентов, так и для родителей и преподавателей. Книги серии помогут выбрать будущую профессию, а тем, кто уже сделал свой выбор, помогут сделать первые шаги на пути к профессионализму, познакомиться «изнутри» с современной кибернетикой: роботами и киборгами, оптимизацией и адаптацией, искусственным интеллектом и управлением хаосом. Девиз серии «Учись играя» означает, что книги будут нацелены не только на обучение, но и на развлечение, на воспитание новых поклонников и фанатов увлекательной науки кибернетики, которой так много предстоит сделать в XXI веке.

В серии «Шаги в кибернетику» в 2011—2013 гг. вышли следующие книги:

— В. Г. Быков «От маятника к роботу. Введение в компьютерное моделирование управляемых механических систем»,

— Р. М. Лучин «Программирование встроенных систем. От модели к роботу»,

— С. А. Филиппов «Робототехника для детей и родителей», 2-е и 3-е издания,

— «Санкт-Петербургские олимпиады по кибернетике 1999—2012».

Зав. лабораторией «Управление сложными системами»
Института проблем машиноведения РАН
доктор технических наук, профессор
А. Л. Фрадков

Предисловие автора к третьему изданию

Во третьем издании книги добавлено несколько тем, которые могут быть полезны начинающим робототехникам, улучшены иллюстрации, добавлены примеры на RobotC, а также исправлен ряд опечаток и ошибок.

Из нового отмечу следующие темы: улучшенная модель одномоторной тележки, робот для лабиринта, скоростной робот для движения по линии, робот-манипулятор, шестиногий шагающий робот, массивы и файлы в RobotC. Самые интересные алгоритмические примеры сосредоточены в главах «Алгоритмы управления» и «Задачи для робота».

Благодарю всех коллег и учеников, которые так или иначе приняли участие в работе над третьим изданием. Особенно рад выделить помощь Евгения Михайловича Сырова, благодаря содействию которого были выявлены многие неточности и опечатки.

Надеюсь, чтение этой книги будет интересным, а более всего принесут пользы практические опыты с роботами.

Пожелания и замечания прошу присылать по следующему адресу: *robobook@mail.ru*.

Глава 1. Знакомство с конструктором

Введение

В современном сознании, сформированном не одним поколением фантастов, робот представляет собой некоторый человекоподобный механизм, выполняющий полезную людям работу (или, наоборот, бунтующий и чрезвычайно опасный). Однако промышленные роботы редко похожи на людей или животных.

Само слово «робот» является существительным, обозначающим неодушевленный предмет, и мы говорим: «строим роботы». Сравните: «строим мосты» и «разводим слонов». Но ребенку свойственно анимировать попадающую ему в руки игрушку, т. е. воображать ее подобной живому существу, одушевленной. А разве взрослым не хочется того же? Отчасти поэтому допустимы два варианта склонения.

Роботы очаровательны. Идея неживой материи, которая самостоятельно выполняет сложные задания, просто поразительна! С тех пор как роботы стали такими технологически сложными и современными, можно было бы подумать, что для их конструирования и программирования необходимы большие знания и навыки. Однако серия *кибернетических* конструкторов Lego Mindstorms делает робототехнику легкой и увлекательной как для взрослых, так и для детей.

Серия конструкторов Mindstorms началась еще в 1998 г. с робототехнической изобретательской системы (Robotics Invention System — RIS), созданной на базе контроллера RCX. Устройства вроде моторов, датчиков и микрокомпьютеров могли совмещаться с другими обычными деталями Lego для создания действующих роботов (рис. 1.1). RIS также была оснащена доступным языком программирования, который позволял самостоятельно запрограммировать действия самодельных роботов на базе RCX.

Рис. 1.1. Робот на базе RCX.

Начиная с 2006 г. с новым набором Lego Mindstorms NXT пользователи получили многочисленные усовершенствования по сравнению с RIS, делающие создание роботов еще проще и увлекательнее.

Однако конструктор NXT выходит за пределы простых усовершенствований «железа» и программного обеспечения. Новый набор открывает робототехнику для всех возрастов.

Как он может попасть к Вам в руки

Если за последнее десятилетие Вам не удалось познакомиться с RIS или другими наборами на базе RCX, не стоит огорчаться. Практически все их возможности и даже гораздо больше можно получить, используя новое поколение конструкторов — NXT. Гладкие детали от Lego Technic[1], усовершенствованные моторы с датчиками и принципиально новый контроллер — вот основные внешние отличия от коробкообразных роботов прошлого поколения.

Рис. 1.2. Наборы серии Lego Minstorms NXT с роботом Alpha Rex на обложке: слева 8527, справа 8547 NXT 2.0.

Практически в любом отделе Lego магазина игрушек есть набор Lego Mindstorms NXT с кодами 8527 или 8547 (рис. 1.2). На его обложке изображен робот, напоминающий андроида: сплющенная голова с круглыми глазками, руки без кистей, ноги с широченными ступнями и контроллер NXT вместо туловища. Забавно, но не стоит обольщаться: самое интересное будет не в этой модели Alpha Rex, которая служит в основном для привлечения внимания покупателей, а на деле не очень функциональна. Инструкцию по сборке вместе с соответствующим программным обеспечением можно найти на прилагающемся к набору компакт-диске. Но настоящее творчество начнется в тот момент, когда из тех же деталей счастливый обладатель конструктора соберет и запрограммирует совершенно нового робота, которого придумает сам.

[1] Если у Вас уже есть Lego Technic, будьте уверены: они с Lego Mindstorms дополнят друг друга.

Набор 8547 носит гордое имя NXT 2.0, хотя изменений в нем совсем немного: разработаны несколько новых деталей и конструкций, изменен состав датчиков и улучшена среда программирования для малышей. Неприятным открытием оказалось уменьшение числа шестеренок, которые так важны юному робототехнику. Недостающие детали теоретически можно приобрести у компании Lego, но в России это сделать трудно.

В Интернет-магазине робототехнические наборы будут стоить немного дешевле[1], чем в обычном. Это так называемое «коммерческое Lego», версия для дома.

Существует также версия «образовательного Lego», представленная компанией Lego Education. Такой набор найти в отделе игрушек в России нельзя. Поставки конструкторов Lego Mindstorms NXT Edu с кодом 9797 ведутся централизованно по школам через представительство Lego Education в России. Однако такой же конструктор, а также ресурсный набор к нему (с кодом 9695) можно приобрести через Интернет-магазины, которые не так давно появились в России, хотя за границей это будет существенно дешевле. К сожалению, из Интернет-магазина *http://www.legoeducation.us* доставка в Россию так называемого «образовательного Lego» не осуществляется (по крайней мере, на момент написания этой главы), поэтому, желая сэкономить, придется искать обходные пути, приобретать через посредников либо на Интернет-аукционах. Кроме того, к набору 9797 не прилагается программное обеспечение. Его можно приобрести отдельно.

Если наш читатель уже продвинутый робототехник и готов усовершенствовать конструктор, дополнив его новыми датчиками, то в этом помогут производители дополнительных устройств и расширений для Lego Mindstorms NXT: компании HiTechnic (*www.hitechnic.com*), MindSensors (*www.mindsensors.com*), Vernier (*www.vernier.com*) и др. Интернет-магазины, расположенные на сайтах этих компаний, как правило, осуществляют доставку в нашу страну. Дополнительные комплектующие теоретически можно приобрести и в Интернет-магазине Lego (*http://shop.lego.com*), но, как было сказано выше, с доставкой заказов в Россию у Lego не все гладко.

[1] В связи с прекращением выпуска набора 8527 в некоторых Интернет-магазинах остались раритетные экземпляры, цена на которые может быть завышена. Зато цена набора 8547 пока что держится стабильной.

Наборы для школы и дома

Итак, наборы Lego Mindstorms NXT продаются двух видов: для школы (9797) и дома (8527, 8547). Набор для школы (рис. 1.3) уложен в красивый белый пластиковый контейнер с двухуровневым хранилищем деталей внутри: сверху в оранжевых ячейках — основные строительные элементы; внизу — электронные элементы, колеса и некоторые другие крупные детали. На специальных карточках нарисовано, в какой ячейке сколько должно быть деталей определенного типа. Такой набор можно использовать для работы в нескольких различных группах и всякий раз в начале и в конце занятия проверять, все ли детали на месте.

Рис. 1.3. Образовательный набор Lego Mindstorms NXT 9797 (слева) и ресурсный набор 9648 (справа).

Детали набора для дома хранятся все вместе в красочной картонной коробке, и рассортировать их представляется непростой задачей. Находчивые робототехники приобретают недорого в строительных магазинах контейнеры для хранения мелких деталей, и конструктор переезжает на новое место жительства. Однако, несмотря на некоторый беспорядок, набор для дома содержит многие полезные элементы, отсутствующие в школьной версии. В связи с этим вместе с конструктором 9797 рекомендуется приобретать ресурсный набор 9695 (ранее 9648), который стоит недорого и содержит все необходимое (рис. 1.3).

Школьный набор укомплектован также некоторыми устройствами, отсутствующими в наборе для дома. И здесь тоже не все гладко. Во-первых, следует упомянуть аккумулятор Lego, который позволяет заменить шесть пальчиковых аккумуляторов или батареек, но без блока питания его использование не имеет смысла (а этот блок питания к набору

не прилагается). Во-вторых, провода-конвертеры для поддержки устройств RCX и три соответствующие лампочки. И наконец, дополнительный датчик касания, для которого, по необъяснимым причинам, не предусмотрено место в коробке с датчиками.

Ни к одному из наборов не прилагается Bluetooth-адаптер для соединения с компьютером, его надо покупать отдельно. А если решите использовать свой адаптер, будьте внимательны при установке драйверов: для соединения с NXT у Lego есть определенные требования[1]. Правда, для загрузки программ на NXT в этом нет необходимости: к каждому набору прилагается стандартный USB-кабель.

Основной состав набора: что мы купили?

Электроника

Компания Lego продает базовый набор, содержащий все основные детали системы NXT. Он включает в себя несколько электронных устройств, среди которых микрокомпьютер, датчики и моторы. Микрокомпьютер называется процессорным блоком (контроллером) NXT, и это разумный, управляемый компьютером блок, играющий роль «мозга» ваших робототехнических конструкций. Программы управляют им для получения входных данных с датчиков, для активации моторов, проигрывания звуков и многого другого. Сам по себе он является интеллектуальным компьютерным строительным блоком Lego, который дает возможность роботу Mindstorms становиться «живым» и выполнять различные операции.

Процессорный блок NXT (рис. 1.4) имеет семь основных портов, два из которых связаны с возможностью загружать на него программы. На одной стороне процессорного блока есть порт для подключения USB-кабеля. После того как кабель уже подключен, можно использовать его для закачки программ на NXT. У процессорного блока также есть встроенный Bluetooth, который делает возможной беспроводную загрузку программ и сообщение с другими процессорными блоками, мобильными телефонами, оборудованными Bluetooth, и другими BT-

[1] Поддерживаемое программное обеспечение для адаптера Bluetooth – Widcomm® Bluetooth для Windows не ниже версии v.1.4.2.10 SP5 и драйверы для поддержки технологии Bluetooth, включенные в Microsoft Windows XP с Service Pack 2 или Service Pack 3, Windows Vista или Vista Service Pack 1, Apple MacOS X 10.3.9, 10.4 и 10.5.

устройствами. Четвертый порт датчиков оснащен возможностью соединения двух контроллеров обычным проводом NXT по стандарту HS485.

LCD-дисплей на верхней панели процессорного блока может показывать тексты и рисунки, а динамик может проигрывать музыку (как мог и RCX), так же как и заранее записанные звуковые файлы. Например, вы можете запрограммировать вашего робота говорить фразы типа «Привет!» или «Как дела?» через динамик. Это свойство позволяет вывести роботов на новый уровень контакта с человеком и дает детям еще больше удовольствия от игры.

Рис. 1.4. Порты контроллера NXT.

Кнопки NXT выполняют следующие функции:

— оранжевая кнопка — включение/ввод/запуск;

— светло-серые стрелки — используются для перемещения вправо и влево в меню NXT;

— темно-серая кнопка — очистить/назад/выход.

Для управления моторами и получения входных данных от датчиков у блока NXT есть три выходных и четыре входных порта. Датчики могут быть подключены к входным портам, пронумерованным от одного до четырех, соединительными кабелями, которые также прилагаются в системе NXT. Как только датчики подсоединяются к устройству, они начинают посылать информацию об окружающей среде процессорному блоку, и эта информация впоследствии используется для воздействия на поведение робота. Моторы могут быть подключены к трем выходным портам — A, B и C — после этого они служат для того, чтобы робот ходил, поднимал предметы или проделывал многие другие движения.

Моторы NXT являются сервомоторами. Они более мощные, чем моторы RCX, поэтому позволяют создавать более сильных и быстрых роботов. У них также есть встроенные датчики вращения, которые измеряют обороты мотора (в градусах или в полных оборотах), — эта особенность позволяет делать движения робота очень точными.

Всего в стандартной системе NXT существует четыре вида датчиков: 1) касания (Touch Sensor), 2) звука (Sound Sensor), 3) освещенности (Light Sensor), 4) ультразвуковой датчик (Ultrasonic Sensor) (рис. 1.4). В версии 8547 появился новый датчик цвета (Color Sensor), который заменяет собой датчик освещенности и, кроме того, может определять цвета. Однако его быстродействие существенно ниже.

У датчиков касания есть кнопка, которая чувствует, когда на нее нажимают, отпускают или ударяют по ней. Этот датчик может быть полезен для роботов, которые должны обнаруживать препятствия или реагировать на прикосновение.

Датчик звука контролирует громкость звуков окружающей среды. Роботы могут использовать этот датчик для реагирования на голосовые команды.

Датчики освещенности выявляют интенсивность света вокруг них, и они также оборудованы красным светодиодом, так что ваш робот может определять интенсивность отраженного света. Эти датчики позволяют роботу делать множество вещей, например, оценивать уровень освещенности в помещении или двигаться по линии. В некоторых задачах могут быть использованы сразу три или четыре таких датчика.

Датчик цвета в наборе 8547 совмещен с датчиком освещенности и обладает широким спектром возможностей по определению цветовых оттенков. С помощью него можно, например, сортировать цветные кубики или шарики.

Ультразвуковой датчик измеряет время, которое требуется звуковой волне, чтобы отразиться от объекта и вернуться, для измерения расстояния между датчиком и объектом. У этого датчика много видов применения, таких как картографирование окружающей среды робота, выявление препятствий, предотвращение столкновений, выявление движения и др.

Технические параметры блока NXT

— 32-битовый микроконтроллер ARM7: тактовая частота 48МГц, оперативная память (RAM) 64 Кбайт, внешняя память (FLASH) 256 Кбайт;

— 8-битовый микроконтроллер AVR: тактовая частота 8МГц, оперативная память (RAM) 512 байт, внешняя память (FLASH) 4 Кбайт;

— беспроводной канал Bluetooth (устройство соответствует требованиям Bluetooth Class II V2.0);

— скоростной порт USB (12 Мбит/с);

— четыре порта входа, шестипроводной кабель для цифровой платформы (один из портов включает порт расширения, соответствующий требованиям IEC 61158 Type 4/EN 50 170 для использования в будущем);

— три порта выхода, шестипроводной кабель для цифровой платформы;

— графический ЖК-дисплей 100 × 64 пикселя;

— громкоговоритель — качество аудио 8 КГц, аудиоканал с 8-битовым квантованием и частотой семплирования 2—16 КГц;

— источник питания: шесть батарей типа AA или аккумулятор[1] Lego.

Детали для конструирования

Для создания корпуса робота в системе NXT имеются строительные части, какие можно было бы ожидать от набора Lego. Однако они не являются типичными деталями Lego: у большинства из них нет выступов. Как уже было упомянуто ранее, строительные детали системы NXT — серии Technic. И хотя может показаться, что придется потратить много времени, чтобы привыкнуть к конструированию с этими деталями без выступов, они дают больше гибкости и силы конструкциям.

Наряду с базовыми деталями серии Technic, такими как балки, штифты, оси, базовый набор NXT включает и другие, которых не было в RIS. Например, этот набор включает в себя два шарика Lego, поворотные диски и зубцы. Одни из этих деталей были добавлены для облегчения создания конструкций на основе серии Technic, а другие — просто для раскрытия бóльших возможностей. В наборе 8547, а также в новой версии набора 9797 v.95 добавлены дополнительные детали, которые оказались наиболее востребованы пользователями.

В общем и целом разнообразие составных частей, включенных в набор, обеспечивает вас почти бесконечным запасом конструкций роботов. Если не брать во внимание малое число крупных зубчатых колес, с 612 элементами вряд ли ощутится недостаток деталей (или идей!) для конструирования в ближайшем будущем.

Что потребуется еще?

Убедитесь, что Вы не забыли укомплектовать конструктор 6-ю (а лучше 12-ю) пальчиковыми аккумуляторами типа AA и зарядным устройством для них. Запасной комплект аккумуляторов иметь полезно, чтобы не терять время, если они сядут в самый неподходящий момент. Батарейки тоже подойдут, на них роботы будут двигаться несколько резвее, но все хорошее быстро кончается, и придется снова идти в магазин за элементами питания.

---

[1] Входит в комплект образовательного набора Lego Mindstorms NXT 9797.

Если говорить о выборе батареек для NXT, то по этой теме проведена масса исследований. Главный критерий в том, что приобретать стоит батарейки для высокотехнологичных устройств. По мнению автора, неплохим выбором являются: Varta High Energy (высокая длительность работы), Energizer Ultimate Lithium (наиболее стабильное напряжение, но стоят они дороже). Из самых доступных и разрекламированных вариантов можно назвать Energizer Maximum и Duracel Turbo, хотя они имеют средние показатели.

Еще потребуются гладкая светлая однотонная поверхность площадью не менее 1 м^2 (стол, щит или пол), черная изолента или самоклеющаяся пленка и разнообразные вспомогательные предметы: горки, коробки, пластиковые стаканчики, банки из-под лимонада и т.п. Кстати, картонные коробочки, в которые были упакованы детали конструктора, не рекомендуем выбрасывать — они тоже могут пригодиться.

Обзор дополнительных возможностей

В настоящий момент помимо датчиков, поставляемых в стандартном наборе, существуют также датчики «компас», датчики ускорения, гироскопические датчики, цветовые и температурные датчики, и пока вы читаете это, их выпускается еще больше. Компания Lego и компании-партнеры, такие как HiTechnic (*http://www.hitechnic.com*) или Mindsensors (*http://www.mindsensors.com*), посвящают много времени увеличению числа датчиков, работающих с NXT (рис. 1.5). С их помощью можно значительно расширить функциональность роботов.

Вас интересует новая электроника? Теперь с контроллером NXT могут работать почти любые сервомоторы благодаря разработке компании Mindsensors — сервоконтроллеру NXTServo. В январе 2013 г. была анонсирована новая серия конструкторов Lego Mindstorms EV3, которые совместимы с датчиками и моторами NXT, но обладают бóльшими возможностями. Однако можно быть уверенным, что с тем множеством расширений, которые были созданы для платформы NXT, еще долгие годы она будет использоваться и в учебе, и в науке, и для развлечений. Lego с партнерами осознает популярность Mindstorms и активно работает над усовершенствованием старых деталей, над новыми деталями и устройствами для пользователей, которые жаждут создавать еще более быстрых, умных и сложных роботов.

Рис. 1.5. Датчик от Hitechnic.

Программное обеспечение

Зарубежные разработки

Серия конструкторов Lego Mindstorms нашла своих поклонников как среди детей, увлеченных изобретательством, так и среди взрослых инженеров, занимающихся серьезными разработками. Поэтому и программное обеспечение для роботов NXT было выпущено с ориентацией на различный возраст и уровень подготовки пользователей.

Вместе с наборами «для дома» поставляется оригинальная графическая среда программирования Lego Mindstorms NXT. Версия Lego Mindstorms NXT Edu, предназначенная для школ, отличается от «домашней» только тремя буквами в названии и электронным руководством пользователя. Язык программирования системы NXT, именуемый NXT-G, — это графический, drag-and-drop язык, который является не только очень простым для освоения, но еще и мощным. Если вы использовали программное обеспечение ROBOLAB с RCX, возможно, вы обнаружите некоторую схожесть.

Однако в школах, по мнению автора, для изучения робототехники следует использовать именно ROBOLAB версии 2.9, которая поддерживает NXT. Это связано с ресурсоемкостью среды NXT-G: при достаточно широких возможностях в ней можно создавать только очень маленькие программы. Причем не на всех компьютерах NXT-G нормально заработает. Обе среды были разработаны как дополнения к высоко оцениваемому профессиональному языку программирования, называемому LabVIEW, и многим обязаны ему. LabVIEW, далеко не игрушка, используется в сложных системах сбора данных и системах управления по всему миру, служит гибким и мощным орудием для ученых и инженеров[1]. Robolab по своим возможностям существенно ближе к LabVIEW и менее требователен к ресурсам, чем NXT-G. Одним из его достоинств Robolab 2.9 можно назвать наглядность и схожесть с языком блок-схем. Приобрести его можно, например, в Интернет-магазине Lego Education по адресу *http://www.legoeducation.us*. Полноценная поддержка осуществляется на сайте *http://www.legoengineering.com*; там же следует скачивать патчи, расширяющие возможности Robolab, в том числе по работе с датчиками различных производителей.

Надо признать, что большим сюрпризом в NXT-G стало то, что его чрезвычайно просто освоить. Пользователи, у которых совсем нет опыта программирования, могут втянуться очень быстро. Lego мудро ре-

[1] В 2010 г. в России вышла книга «Программируем микрокомпьютер NXT в LabVIEW» [8], ориентированная на старших школьников.

шила включить множество инструкций и рекомендаций по программированию в программное обеспечение; они демонстрируют многие основные управляющие блоки, а также различные техники программирования, которые принесут пользу как начинающим, так и продвинутым пользователям. Графический пользовательский интерфейс так прост в обращении и интуитивно понятен, что многие, погружаясь в него, начинают экспериментировать с программным обеспечением, постигая его работу путем проб и ошибок. Поэтому, надеясь на увеличение мощностей компьютеров в будущем (объем памяти, частота процессора, размеры экрана), стоит не отвергать NXT-G и позиционировать его, как язык для начального самостоятельного освоения программирования роботов, тем более, что он поставляется вместе с конструкторами 8527 и 8547 «для дома».

Гибкость системы NXT допускает программирование и на других языках. Три наиболее общепринятых — это NBC, NXC и RobotC. NBC и NXC — свободные языки, созданные Джоном Хансеном. Оба они текстовые, а NXC похож на язык C (NXC расшифровывается как Not eXactly C — не совсем C). Их можно бесплатно скачать на сайте *http://bricxcc.sourceforge.net/nbc*. Надо признать, что эти языки не раскрывают всю мощь текстового программирования для NXT. RobotC — тоже текстовый язык, очень похожий на C, — обладает существенно большими возможностями. Продукт Carnegie Mellon University's Robotics Academy может быть скачан с *http://www.robotc.net*. Полнофункциональная 30-дневная демоверсия RobotC бесплатна, по прошествии этого срока можно приобрести лицензию за доступную сумму (80$ США).

Остановив свой выбор на трех языках — NXT-G, Robolab 2.9 и RobotC, — рассмотрим классификацию по возрасту и уровню подготовки пользователей, приведенную в табл. 1.1.

Таблица 1.1. Среды программирования роботов на базе NXT

Среда	Язык	Возраст	Назначение
Lego Mindstorms NXT Software	NXT-G	8—12 лет (дети и родители)	Самостоятельное изучение дома, основы
Robolab 2.9.4[1]	Robolab	8—16 лет, (дети, родители, учителя)	Изучение на уроках робототехники, использование на состязаниях роботов
RobotC for Mindstorms	RobotC	14—99 лет (преимущественно программисты)	Использование личного опыта программирования на языке C для создания роботов с широкими возможностями

[1] Robolab 2.9 с установленным патчем до версии 2.9.4.

У компании Lego Education свой взгляд на возраст пользователей конструкторов. Он выражен в диаграмме с сайта *http://www.legoengineering.com*, относящейся ко времени появления среды Lego Mindstorms NXT (рис. 1.6).

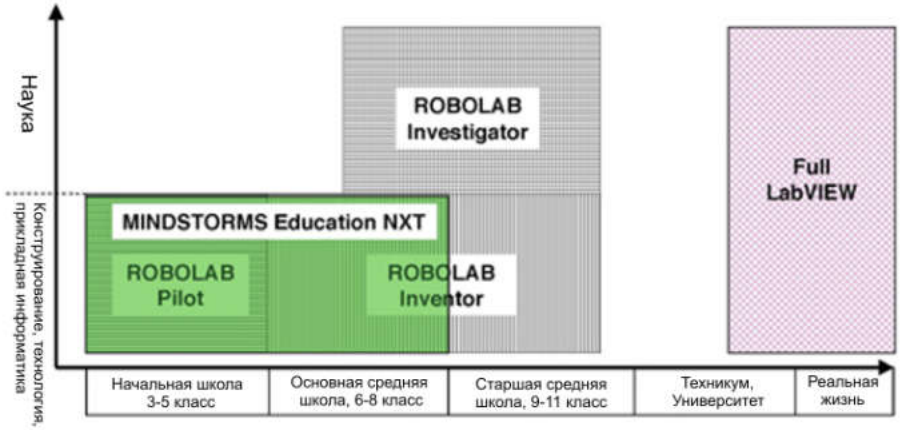


Рис. 1.6. Программное обеспечение для Mindstorms к августу 2006 г.

В 2010 году Lego совместно с National Instruments выпустила продукт LabView for Mindstorms для старшей школы, чтобы заполнить существующий на данный момент пробел между «игрушечной» средой графического программирования Lego Mindstorms NXT и «взрослой» средой LabView, которую используют инженеры (рис. 1.7).

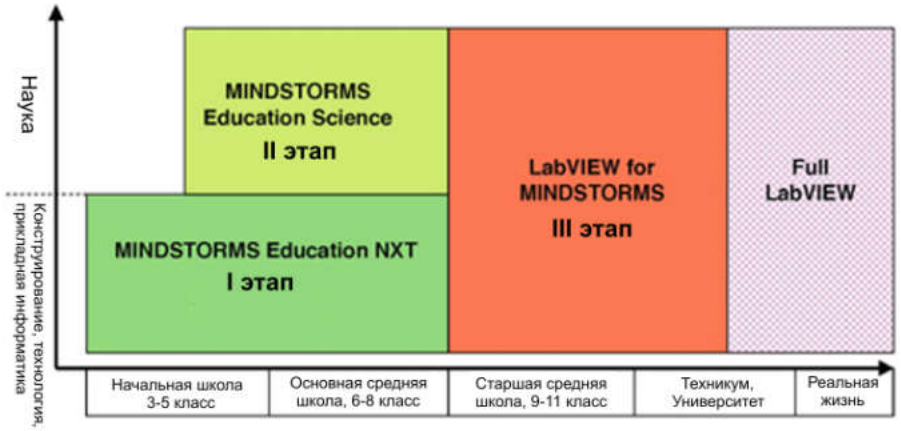


Рис. 1.7. Планы развития программного обеспечения к 2010 г.

До тех пор обновленная версия Robolab 2.9.4 была призвана временно заполнить пробел и обеспечить безболезненный переход к NXT-программированию. Однако, несмотря на появление новой версии

LabVIEW для школ, стандартом по-прежнему остается Robolab, полюбившийся пользователям за свою функциональность, простоту и наглядность. Последним подтверждением преимущества Robolab стало разработанное дополнение к образовательной версии LabView, которое полностью реализует его функционал и носит название Robolab 2.9.4d.

Обратите внимание на то, что в диаграммах отсутствует даже намек на RobotC или аналогичный язык. К сожалению, текстовые среды программирования в школах не распространены в силу всеобщей ориентации на более доступный графический интерфейс. Такая тенденция присутствует во всем. В итоге робототехникой может заниматься человек, который по сути не является программистом. В этом есть и плюсы, и минусы: с одной стороны, роботы входят в нашу жизнь, становятся реальностью, с которой необходимо считаться. Благодаря средам графического программирования можно существенно повысить общий уровень грамотности учащихся в этой сфере. С другой стороны, серьезными специалистами, скорее всего, станут только те, кто имеет глубокую алгоритмическую подготовку.

Отечественные разработки

Россия сильна своими математиками и программистами. И хотя нет пока отечественного робототехнического конструктора, но уже появилась серьезная альтернатива зарубежным средам программирования роботов, которая в ближайшем будущем сможет превзойти их по всем параметрам. Это разработка ГУП «Терком», базирующегося на математико-механическом факультете Санкт-Петербургского государственного университета. Программный продукт QReal:Robots — это среда графического проектирования, позволяющая не только быстро создавать программы, похожие на блок-схемы, но и сразу просматривать их текстовый аналог на языке Си.

Для генерации исполняемого кода используется свободнораспространяемая операционная система реального времени nxtOSEK, задача которой состоит в управлении контроллером NXT. Для специалистов nxtOSEK сама по себе интересна быстродействием и эффективным использованием ресурсов контроллера.

Для детей и преподавателей QReal:Robots интересна как многофунциональная среда, содержащая возможности графического и текстового программирования одновременно. Начиная с красочных пиктограмм, учащиеся постепенно переходят к строгому и функциональному коду на языке Си. Кроме того, QReal снабжена режимом моделирования поведения робота в виртуальной среде. Этим могут «похвастаться» разве что Microsoft Robotics Studio, RobotC Virtual Worlds и MatLab с соответствующими надстройками.

Глава 2. Конструирование

Способы крепления деталей

Если читатель уже имеет опыт конструирования на основе Lego Technic, то этот раздел можно смело пропустить и переходить сразу к разделу «Волчок». Если же подобный конструктор у вас в руках впервые, стоит изучить эту главу даже раньше, чем инструкцию Quick Start, предлагаемую Lego.

Различия принципов конструирования RIS и NXT

Хотя система NXT и была разработана, чтобы усовершенствовать RIS, многие приверженцы старой системы, как это часто случается, были сначала недовольны некоторыми изменениями. Одна из претензий была связана с новым типом строительных деталей в системе NXT (детали без выступов). В RIS большинство строительных элементов похожи на обычные строительные блоки Lego, у них есть выступы или шипы (такие маленькие круглые части, которое выпирают на верхушке деталей, придавая им классический вид Lego). Строительные детали системы NXT практически все относятся к серии Technic и не имеют выступов. Кстати говоря, вы можете услышать, как многие пользователи говорят о «гладком» конструировании, имея ввиду именно эти новые детали серии Technic! Однажды привыкнув к конструированию с использованием деталей с выступами, может быть, сложно перейти к «гладкому» конструированию. Однако, скорее всего, вы обнаружите, что на самом деле оно упрощает построение более сильных и гибких конструкций.

Конструирование с деталями Technic действительно улучшает модели; детали крепче соединяются друг с другом и благодаря разнообразию их форм NXT-наборы предлагают много новых модификаций в форме роботов.

Используя RIS, пользователи часто жаловались, что модели выглядят коробкообразно, и «квадратные» роботы были нормой, так как строились с помощью стандартных кирпичиков Lego (а они прямоугольные). Роботы NXT, напротив, могут иметь различные формы, и пользователи на самом деле получают удовольствие, создавая уникальный, никем не виданный дизайн. Кроме того, роботы NXT выглядят более похожими на реальных роботов.

Другой причиной жалоб была хрупкая природа роботов на базе RCX. Многих из них постигла неприятная судьба: в результате падения с высоты стола роботы разбивались вдребезги на дюжины или даже сотни кусочков. Этого больше не повторится! Детали NXT Technic более шершавы и крепко держатся вместе; они, конечно, разъединятся, если робот будет сброшен с достаточной высоты, но все равно скрепляются вместе намного лучше своих предшественников.

В общем и целом работа с деталями Technic достаточно легка для понимания, крепление деталей и создание новых моделей происходит довольно быстро.

Первая игра: фантастическое животное

Эта игра очень проста. Участвуют двое. Надо разделить часть имеющихся деталей на два одинаковых комплекта (чем младше участники, тем меньше деталей в комплекте). Один игрок втайне от второго строит из своих деталей некое фантастическое животное. Затем второй игрок берется за свой комплект. Оба отворачиваются друг от друга и второй игрок под диктовку первого строит копию этого фантастического животного, ни разу не взглянув на него. А первый тоже не должен видеть, что делает второй. К концу строительства обе конструкции сравниваются (рис. 2.1). Во втором раунде игроки меняются ролями.

Рис. 2.1. Не все пары получаются похожими, но что-то в этом есть.

Интереснее всего будет зрителю, который наблюдает со стороны. «Возьми серую загогулину, в которой по четыре дырочки. Прикрепи к ней оранжевый зуб с помощью синей палочки…» — это самое понятное из того, что можно будет услышать в ходе игры. Вывод напрашивается сам собой. Чтобы понимать друг друга, надо знать названия деталей. К сожалению, эти названия не прилагаются к конструкторам и доступны только в методических пособиях. Но часть из них мы все-таки публикуем в Приложении 1.

Высокая башня

Долго рассказывать о способах крепления деталей бессмысленно, надо сразу начинать конструировать. Первая самостоятельная задача, которую стоит поставить перед начинающим робототехником, — это строительство высокой башни из всех возможных деталей конструктора. Возможно, до потолка. Правда, было замечено, что при высоте более одного метра башня начинает терять равновесие, падать и разваливаться. В связи с этим стоит принять за правило не вешать микроконтроллер на саму верхушку. Это понятно: шпиль должен быть легким, а основная масса сосредоточена внизу. Для того чтобы создать устойчивую конструкцию, волей-неволей придется перепробовать многие способы крепления деталей (рис. 2.2).

Рис. 2.2. Лишь бы не упала!

Механический манипулятор

Эту игрушку можно назвать по-разному. Иногда на ее конце располагается голова клоуна, которая внезапно высовывается из потаенного ящичка на большое расстояние. Иногда она похожа на длинную свернутую трубочку — «тещин язык», которая выпрямляется с противным свистом. Иногда на конце располагается маленькая боксерская перчатка. Добавим игрушке немного функциональности и назовем ее

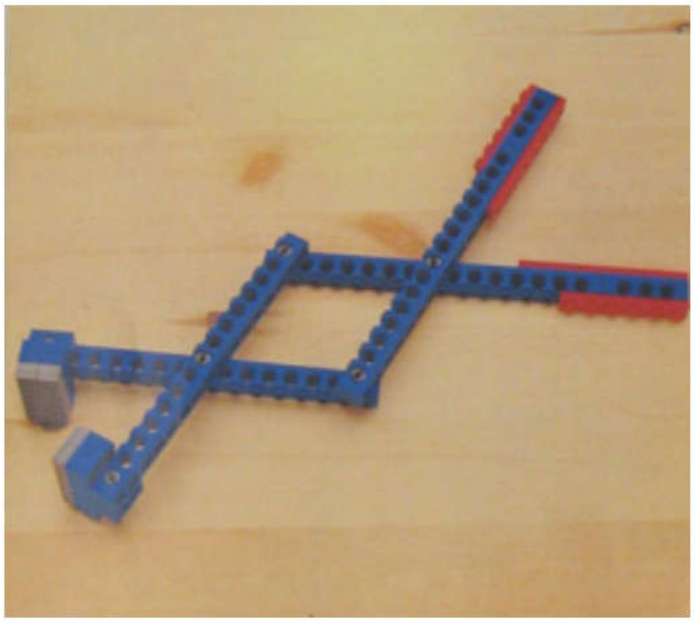


Рис. 2.3. «Хваталка», созданная из старинного набора «Простейшие машины и механизмы».

механическим манипулятором или просто «хваталкой». И постараемся сделать ее как можно длиннее (рис. 2.3).

Опишем требования к конструкции:

— хватательный механизм должен иметь минимальную длину в сложенном состоянии и максимальную в разложенном;

— у механизма должно быть две ручки, как у щипцов, и многоколенчатое соединение, ведущее к хватательной части;

— изобретатель должен суметь взять с помощью «хваталки» некоторый предмет (например, колесо из набора) и перенести его с места на место.

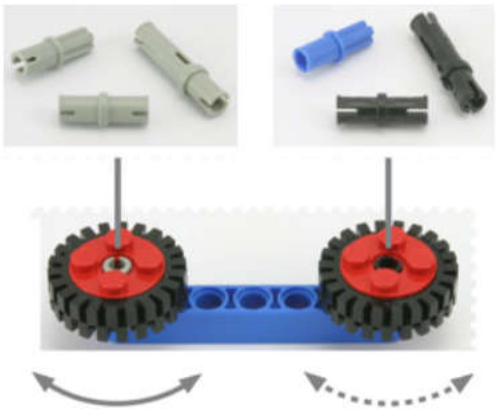


Рис. 2.4. Серые штифты и бежевые штифты-полуоси предназначены для вращения, а черные и синие – для фиксации [3].

Начальный этап создания конструкции прост: шарнирные соединения с помощью штифтов в трех

точках на каждой из используемых балок: посередине и по краям. Здесь обратим внимание на типы используемых штифтов. Часть из них гладкие, а другие имеют небольшие ребра для фиксации в отверстиях (рис. 2.4).

Конечно, в нашей конструкции лучше использовать гладкие, пусть даже трехмодульные детали, поскольку штифты с фиксатором затрудняют вращение вокруг них. Но они не блокируют движение полностью, поэтому при отсутствии достаточного числа гладких фиксирующие штифты тоже подойдут.

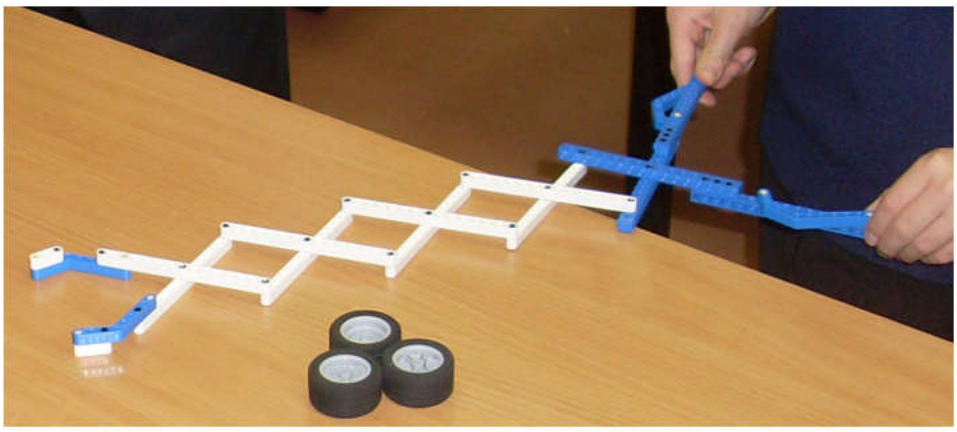


Рис. 2.5. Задача для механического манипулятора — сложить пирамидку из колес и переместить ее с места на место.

На втором этапе необходимо соорудить хватательную часть, которой будут удерживаться предметы (рис. 2.5). А третий этап — научиться пользоваться манипулятором только одной рукой. Оставляем это для фантазии читателя.

Механическая передача

Важнейшей частью почти каждого робота является механическая передача. В разных конструкторах предлагается несколько ее видов: зубчатая, ременная, цепная и др. Передача бывает необходима, для того чтобы передать крутящий момент с вала двигателя на колеса или другие движущиеся части робота. Довольно часто требуется передать вращение на некоторое расстояние или изменить его направление, например на 180 или 90 градусов.

Передаточное отношение

При всякой передаче существенную роль играет особая величина — передаточное отношение (а также передаточное число), которое надо научиться рассчитывать. Для этого необходимо знать число зубчиков на шестеренках при зубчатой или цепной передаче и диаметр шкивов при ременной передаче. На крупных шестеренках число зубцов указано: например, «Z40» на самой большой. На мелких нетрудно сосчитать их самостоятельно.

Теперь посмотрим, что происходит при зубчатой передаче. Во-первых, направление вращения ведомой оси противоположно направлению вращения ведущей. Во-вторых, можно заметить, что разница в размере шестеренок влияет на угловую скорость вращения ведомой оси. Каким образом?

Ведущая меньше ведомой — скорость уменьшается. Ведущая больше ведомой — скорость увеличивается.

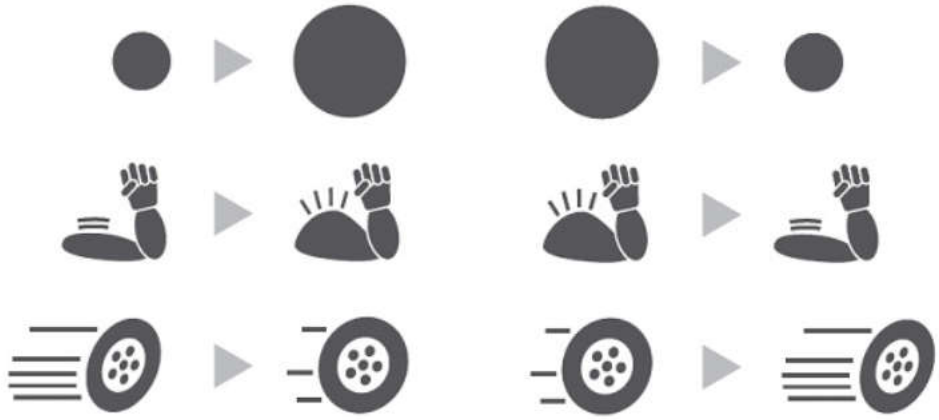


Рис. 2.6. При передаче с малого колеса на большое выигрываем в силе, но теряем в скорости. При передаче с большого на малое — все наоборот [3].

Однако надо понимать: выигрыш в скорости должен обернуться проигрышем в чем-то ином. И наоборот. Что же мы теряем при увеличении скорости? Очевидно, тяговую силу. А при понижении скорости выигрываем в силе (рис. 2.6). Это замечательное свойство зубчатой передачи используется во множестве механизмов, созданных человеком, — от будильника до автомобиля.

Как точно узнать, во сколько раз увеличилась тяговая сила? За это отвечает специальная величина, именуемая «передаточное отношение». Для нашего конструктора мы определим ее следующим образом:

$$i = \frac{z_2}{z_1},$$

где i — передаточное отношение, z_2 — количество зубцов на ведомой шестерне, z_1 — число зубцов на ведущей шестерне.

Таким образом, при $i < 1$ тяговая сила уменьшается, а угловая скорость возрастает (рис. 2.7); при $i > 1$ сила увеличивается, а скорость падает. Очевидно, что при $i = 1$ и сила, и скорость остаются прежними. В этом случае мы можем ощутить изменения только за счет потерь при трении.

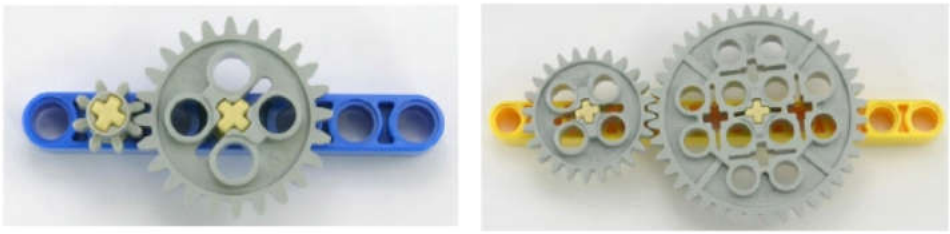


Рис. 2.7. Передача с понижением скорости: слева i = 3 : 1, справа i = 5 : 3 [3].

Если в передаче используется несколько подряд установленных зубчатых колес, то при расчете передаточного отношения учитывается только первое и последнее из них, а остальные называются «паразитными» (рис. 2.8). Паразитные шестерни исполняют полезную функцию только при необходимости передачи вращения на некоторое расстояние. В остальных случаях они лишь увеличивают потери на трение.

Рис. 2.8. Две промежуточные шестерни — паразитные [3].

Однако зубчатую передачу можно построить таким образом, чтобы каждая шестерня выполняла полезную функцию и служила либо для увеличения, либо для уменьшения передаточного отношения.

В этом случае каждая вторая пара соседних шестеренок должна находиться на одной оси. А общее передаточное отношение рассчитывается как произведение всех передаточных отношений соприкасающихся шестеренок.

$$i = i_{12} \cdot i_{34} \cdot i_{56} \ldots, \text{ где } i_{12} = \frac{z_2}{z_1},\ i_{34} = \frac{z_4}{z_3},\ i_{56} = \frac{z_6}{z_5} \ldots$$

Нетрудно догадаться, что шестеренки, находящиеся на одной оси, вращаются абсолютно одинаково и их передаточное отношение равно единице. Следовательно, эти значения в произведении могут не участвовать (рис. 2.9).

Рис. 2.9. Двухступенчатая передача [3].

И, наконец, определим понятие «передаточное число». Его используют, когда необходимо вычислить коэффициент изменения скорости или силы вне зависимости от направления возрастания. Таким образом, передаточное число можно определить как наибольшее из отношений $u = i / 1$ или $u = 1 / i$. Следовательно, передаточное число всегда не меньше единицы: $i \geq 1$. Для примера, при передаточном отношении $i = 1 : 15$, как и при $i = 15 : 1$, передаточное число $u = 15$ (рис. 2.10).

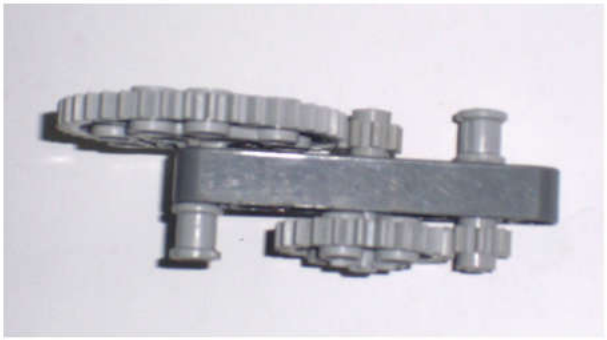


Рис. 2.10. Передаточное число 15.

Червячная передача — это частный случай зубчатой (рис. 2.11). В нашем конструкторе она обладает определенными свойствами. Во-первых, один оборот червяка соответствует одному зубцу любой шестерни. Значит, при расчете передаточного отношения число зубцов червяка можно считать равным единице: $z_ч = 1$. Во-вторых, червячная пере-

дача работает только в одном направлении от червяка к шестерне и блокирует движение в обратном направлении.

Задача. Постройте механическую передачу с максимальным передаточным отношением. По приблизительным подсчетам, из всех шестеренок конструктора 8527 можно построить передачу, увеличивающую силу вращения (и понижающую скорость) примерно в 2 млн раз. Это, конечно, теоретически. Но по сути это означает, что для одного полного оборота ведомой оси потребуется около 2 млн оборотов ведущей. Многовато.

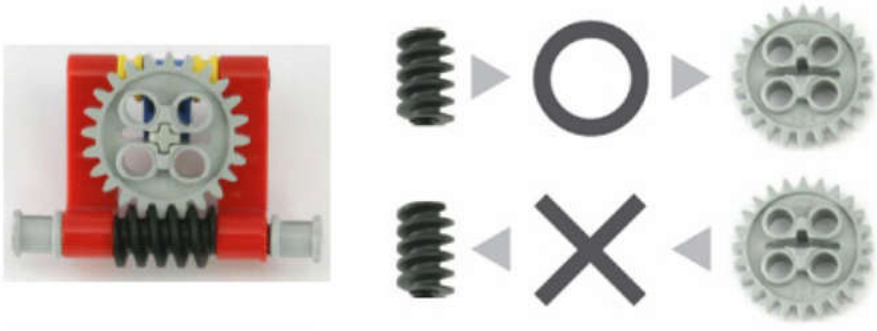


Рис. 2.11. Червячная передача работает только в одну сторону: от червяка к шестерне [3].

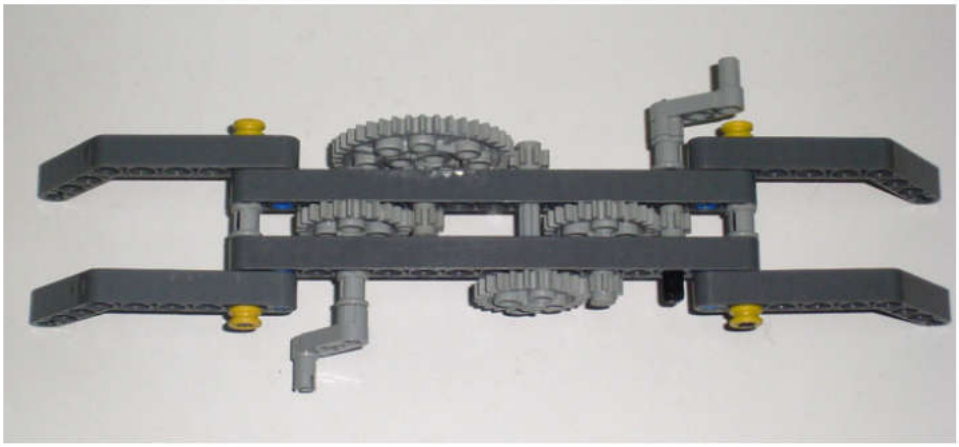


Рис. 2.12. Механическая передача с передаточным числом 135.

Для начала попробуйте построить передачи с передаточным числом 9, 27, 45, 135 (рис. 2.12). А если не получится, то поможет следующий параграф. Только в нем мы будем не замедлять, а ускорять движение.

Волчок

Каждый из нас с детства знаком с замечательной игрушкой — юлой. Несколько движений — и юла около минуты держит вертикальное положение. С волчком еще проще. Стержень, закрепленный на нем диск, быстрое круговое движение пальцами, и с волчком происходит то же самое, что и с юлой, только без какого-либо механизма. Но так или иначе, через некоторое время волчок падает, исчерпав заложенный в него ресурс. Хорошая тренировка для пальцев, однако скорость вращения запущенного рукой волчка невысока.

Можно ли создать механизм, который многократно увеличит начальную скорость вращения волчка? Наших знаний о передаточном отношении должно быть достаточно для решения этой задачки. Вторая задача — максимально продлить время вращения волчка.

Требования к волчку и механизму:

— волчок должен иметь ось вращения и достаточно тяжелый диск-маховик, который сохранит инерцию вращения;

— центр тяжести волчка должен быть расположен достаточно низко, но и не слишком, чтобы края диска не цеплялись за поверхность стола;

— на оси вращения волчка необходимо установить шестерню для начального сцепления с механизмом;

— на механизме должны присутствовать две детали: для удержания одной рукой и придания вращения другой;

— в момент раскручивания волчок должен иметь плотное соприкосновение с механизмом;

— сразу после раскручивания волчок должен свободно отделяться от механизма.

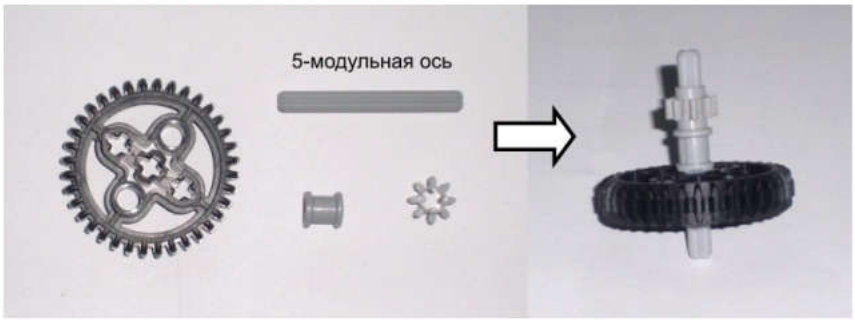


Рис. 2.13. Волчок.

Предлагаем читателю нашу версию волчка и запускающего механизма (рис. 2.13). Начинающий изобретатель наверняка сможет ее усовершенствовать, подобрав наиболее подходящий маховик и оптимальное передаточное отношение.

В качестве маховика волчка может быть любое колесо, не обязательно зубчатое. Отметим, что волчок с более тяжелым маховиком вращается дольше.

Далее построим механизм для запуска (рис. 2.14). Для этого возьмем 8-модульную ось в качестве ведущей.

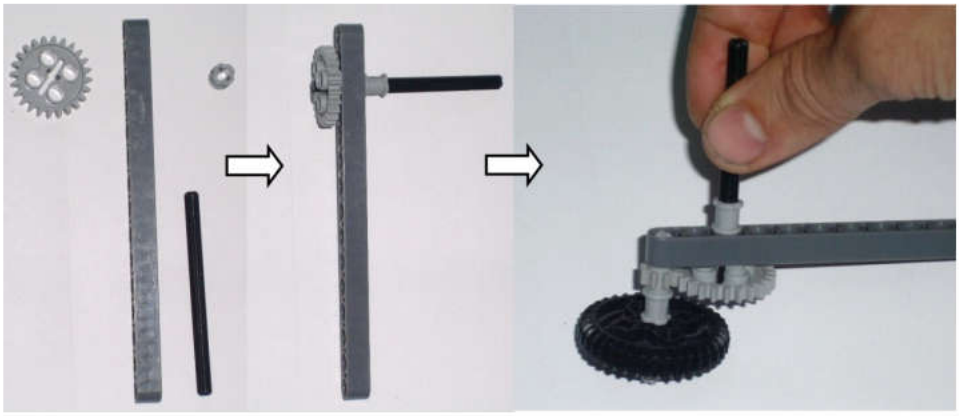


Рис. 2.14. Простейший запускающий механизм 1 : 3.

При запуске необходимо быстро провернуть ведущую ось на механизме и сразу поднять его, чтобы волчок вращался свободно. Попробуйте самостоятельно построить механизм с передаточным отношением $i = 1 : 5$.

Переходим к двухступенчатой передаче. Следует заменить первую ось на более короткую (например, 4-модульную), а вместо втулки поставить малую 8-зубую шестеренку.

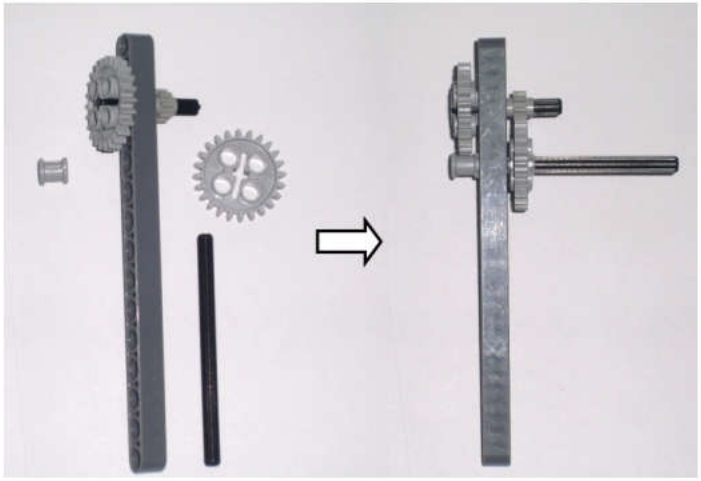


Рис. 2.15. Двухступенчатая передача с передаточным отношением 1 : 9 (с учетом малой шестерни волчка).

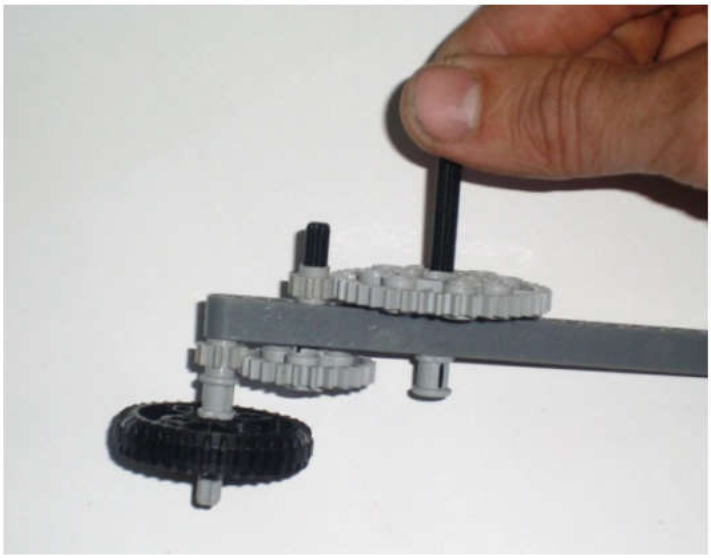


Рис. 2.16. Передача 1 : 15.

Теперь соберите самостоятельно передачу с передаточным отношением 1 : 15, которая изображена на рис. 2.16.

Третью ступень передачи можно построить с помощью 40-зубой шестерни. Передаточное отношение в этом случае будет 1 : 45. Соответственно и усилие для раскручивания потребуется в 45 раз большее. Поэтому на ведущую ось следует поставить рычаг или дополнительное колесико, которое облегчит задачу запускающему (рис. 2.17).

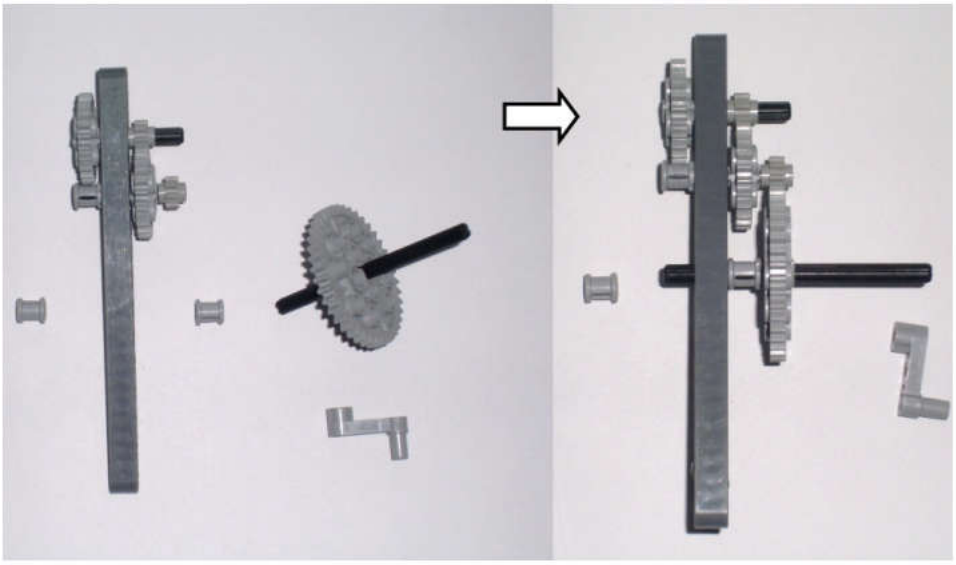


Рис. 2.17. При добавлении третьей ступени может потребоваться рычаг.

Когда при запуске слышен характерный треск зубчиков шестеренок, который возникает на холостом прокручивании, стоит побеспокоиться. Если ваши роботы будут трещать, зубчатые колеса быстро выйдут из строя. Это происходит из-за непрочного крепления осей в балках: они с легкостью отклоняются на несколько миллиметров. От этого можно защититься, добавив дополнительные балки и удлинив тем самым отверстие, в котором расположена ось (рис. 2.18).

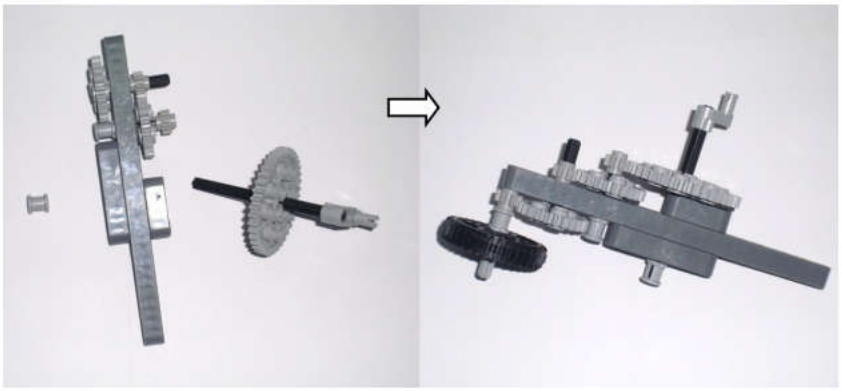


Рис. 2.18. Защита от холостого прокручивания шестеренок.

Наша защита действует, но при чрезмерных перегрузках сдает и она. Подумайте, как еще можно укрепить механизм дополнительными балками, например над шестеренками.

Даже из одного конструктора можно сделать несколько волчков и запускающих механизмов и устроить состязания с товарищами: чей волчок продержится дольше?

Механизм, который мы построили для запуска волчка, в технике носит название *мультипликатор*. Его назначение – повышение скорости вращения ведомой оси.

Редуктор

Этот механизм используется совместно с двигателями для преобразования и передачи крутящего момента. Чаще всего он служит для понижения частоты вращения и повышения крутящего момента вместе с тяговой силой. Существует множество разновидностей редукторов. То, что мы сделаем сегодня — это, с одной стороны, полезный механизм, с другой — головоломка для товарищей.

Суть проста: внутри «черного» ящика (картера) строится зубчатая передача таким образом, чтобы ведомая ось находилась на одной прямой с ведущей (рис. 2.19). Поскольку вращаться они могут с разной скоростью и даже в разные стороны, построить такую же головоломку

представляется простой, но довольно интересной задачей. Желательно, чтобы никакие другие (внутренние) оси не выступали из корпуса редуктора во избежание механических помех.

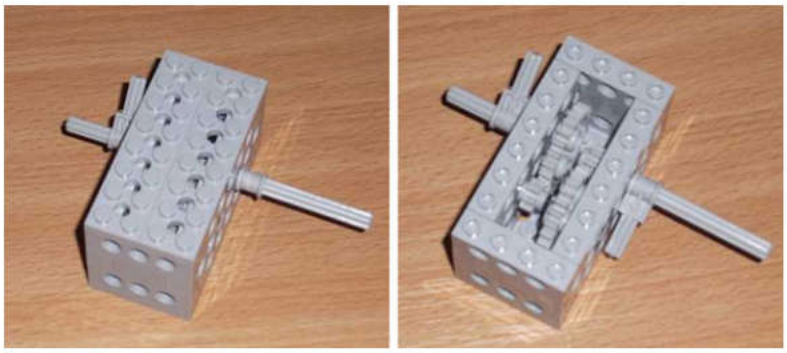


Рис. 2.19. Осевой редуктор с передаточным числом 9.

В этой модели передаточное отношение 9 : 1. Более короткая (ведомая) ось обладает в 9 раз большей силой тяги и меньшей угловой скоростью вращения. Отношение рассчитывается следующим образом:

$$i = \frac{24}{8} \cdot \frac{24}{8} = \frac{9}{1}.$$

Более сложный вариант редуктора имеет передаточное отношение 15 : 1 и противоположное направление вращения.

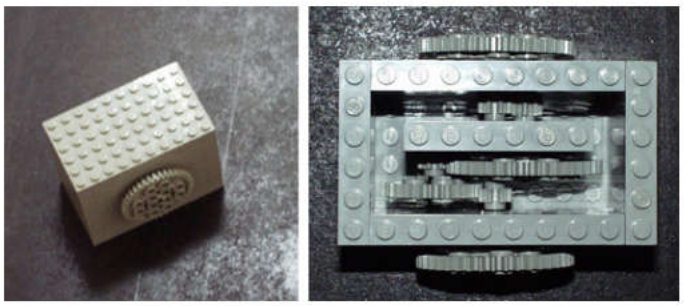


Рис. 2.20. Осевой редуктор с передаточным числом 15 и обратным направлением вращения.

Интересной задачей является построение осевых редукторов с передаточными отношениями 1 : 1, 3 : 1, 4 : 1 с прямым и противоположным направлениями вращения ведомой оси.

Используя червячную передачу или конические шестеренки, можно изменить направление ведомой оси на 90 градусов. Попробуйте построить черный ящик с несколькими выходящими из него ведомыми осями в разных направлениях.

Глава 3. Первые модели

Моторы вперед!

Следующие несколько проектов можно выполнить, не вдаваясь особо в программирование. В каждом из них достаточно будет включить один из моторов. Это делается несколькими различными способами. Составим программу, включающую мотор в каждой из трех сред, с которыми мы познакомимся подробнее в главе «Программирование», а также с помощью встроенной оболочки самого NXT.

NXT Program

Во встроенной оболочке NXT есть возможность включить моторы B и C с мощностью около 75 %, не прибегая к компьютеру (рис. 3.1). При этом в некоторых версиях оболочки (Firmware) по умолчанию требуется, чтобы были подсоединены обязательно оба мотора. В случае если хотя бы одного из них не хватает, алгоритм пробуксовывает. Однако вращение так или иначе происходит. В частности, при оригинальной прошивке Lego Mindstorms NXT такое движение будет прерывистым. Избавиться от этого можно подсоединением второго мотора.

Итак, в квадратных ячейках требуется разместить всего пять команд (см. рис. 3.1):

1) **Forward (Backward),**
2) **Empty,**
3) **Forward (Backward),**
4) **Empty,**
5) **Loop.**

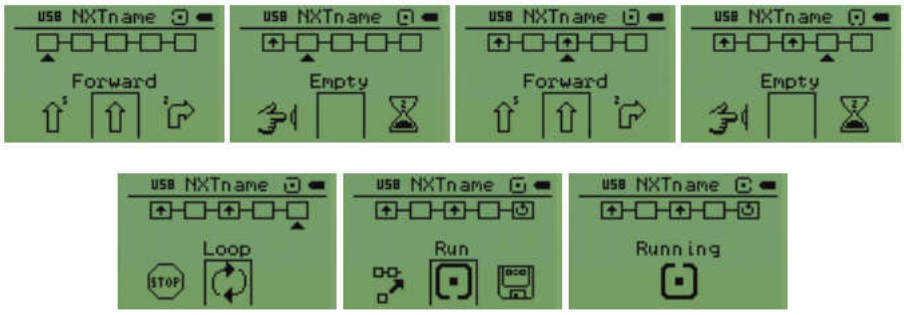


Рис. 3.1. Составление и запуск программы для включения мотора B или C.

Бывает так, что конструктивно лучше дать моторам команду «Назад». Для этого команду Forward следует заменить на Backward в обоих ячейках.

Созданную программу можно сохранить, и она появится в меню NXT Files, расположенном в разделе My Files.

Обратите внимание, что мотор A в этом случае останется неподвижным.

У этих команд управления моторами есть особенность, из-за которой моторы продолжают вращаться и после принудительного завершения программы. Остановить их можно или выключением NXT, или выполнением программы с командой Stop в последнем блоке (рис. 3.2).

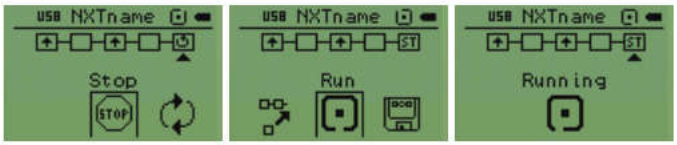


Рис. 3.2. Остановка моторов с помощью команды Stop в конце программы.

Чтобы заменить команду Loop на Stop, достаточно нажать пару раз темно-серую кнопку, откатившись в режим редактирования программы, и выбрать Stop. Затем остается снова выбрать Run. Сама по себе возможность запуска одного мотора не предусмотрена разработчиками NXT Program, а найденное решение не идеально, но позволяет сэкономить немного времени.

Подробнее о встроенной оболочке NXT Program читатель узнает после конструирования двухмоторной тележки.

NXT-G

В этой среде программа, запускающая мотор A вперед, выглядит так, как показано на рис.3.3.

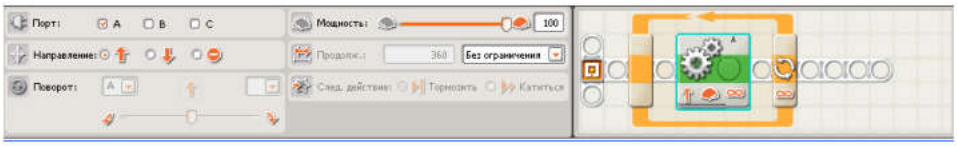


Рис. 3.3. Мотор A вперед на языке NXT-G.

Создайте цикл, который будет выполняться постоянно, а в него поместите пиктограмму «Движение». В окне свойств установите галочку напротив мотора A, задайте максимальную мощность и установите продолжительность «Без ограничения».

Загрузить программу в NXT можно, щелкнув мышкой кнопку «Загрузка» на командном центре (рис. 3.4), предварительно соединив NXT с компьютером и включив его.

Программа появится в памяти NXT в меню My Files → Software Files с именем, которое вы ей дадите в среде при сохранении файла. По умолчанию, это Untitled.

Рис. 3.4. Загрузка программы на NXT.

Robolab 2.9

В среде Robolab включить мотор A можно аналогичным способом.

В разделе «Программист» кликните дважды пункт Inventor 4 и на белом поле создайте программу, изображенную на рис. 3.5.

Рис. 3.5. Мотор A вперед на языке Robolab.

Для загрузки программы в NXT необходимо кликнуть по белой стрелочке в левом верхнем углу экрана. Если NXT ответил звуковым сигналом, значит все прошло успешно. Программа появится в меню My Files → Software Files с именем «rbl».

RobotC

Вот почти самая короткая программа на этом замечательном языке:

```
task main()
{
  while(1)
    motor[motorA] = 100;
}
```

Для загрузки программы нажмите F5, после чего, не дожидаясь звукового сигнала, ищите ее в меню NXT My Files → Software Files.

Если у вас возникнет желание работать сразу с несколькими средами, имейте ввиду, что между ними нет совместимости на уровне прошивки микроконтроллера. Поэтому при каждом переходе придется заново загружать прилагающуюся версию операционной системы (Firmware) NXT.

Тележки

Роботы призваны заменить человека во множестве жизненных ситуаций. Для этого они должны обладать различными качествами: силой, мобильностью, прочностью, интеллектом, памятью и т. д. Для каждой задачи требуются свои характеристики, поэтому никого не удивит, если одни роботы вообще не будут иметь конечностей, а другие, например, будут ездить на колесах, как автомобили. До искусственного интеллекта нам пока далеко, а первую мобильную конструкцию назовем «Тележка». Рассмотрим ее простейший вариант.

Какие требования предъявим к этому механизму:

— четыре колеса, плото сцепленные с поверхностью земли, попарно соединенные осями; колеса должны свободно вращаться;

— отсутствие других частей конструкции, соприкасающихся с поверхностью;

— наличие электромотора, приводящего в движение передние и / или задние колеса;

— наличие источника питания для электромотора и программы управления им;

— по команде оператора тележка должна двигаться вперед или назад и транспортировать полезный груз.

Одномоторная тележка

Тележка в первой версии не может поворачивать, поэтому ограничимся одним мотором с передним приводом (рис. 3.6—3.11).

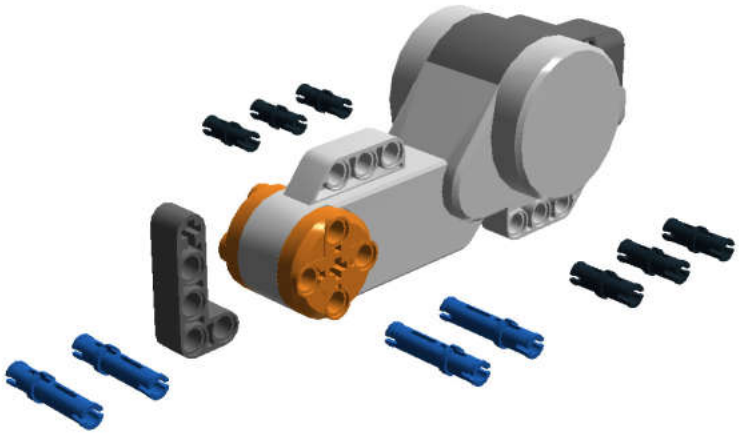


Рис. 3.6. Одномоторная тележка. Синие трехмодульные штифты вставляются в угловую балку и верхнюю дужку мотора симметрично с краю.

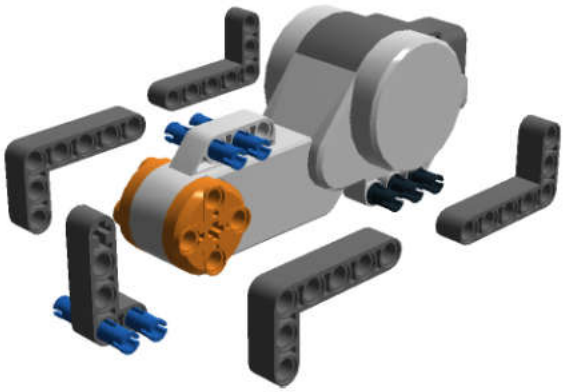


Рис. 3.7. Уголки 3 × 5 крепятся на все выступающие части штифтов.

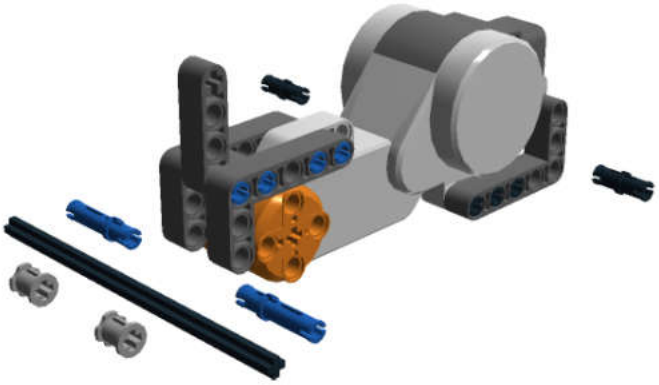


Рис. 3.8. В оранжевый диск мотора вставляется 12-модульная ось, которая будет ведущей.

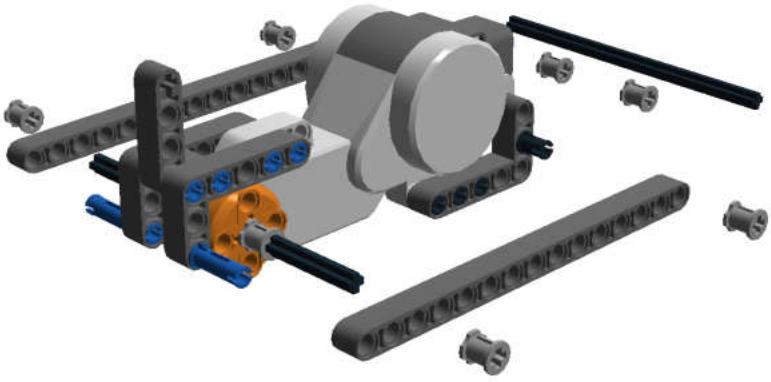


Рис. 3.9. Такая же ось сзади крепится в крайние отверстия несущих балок.

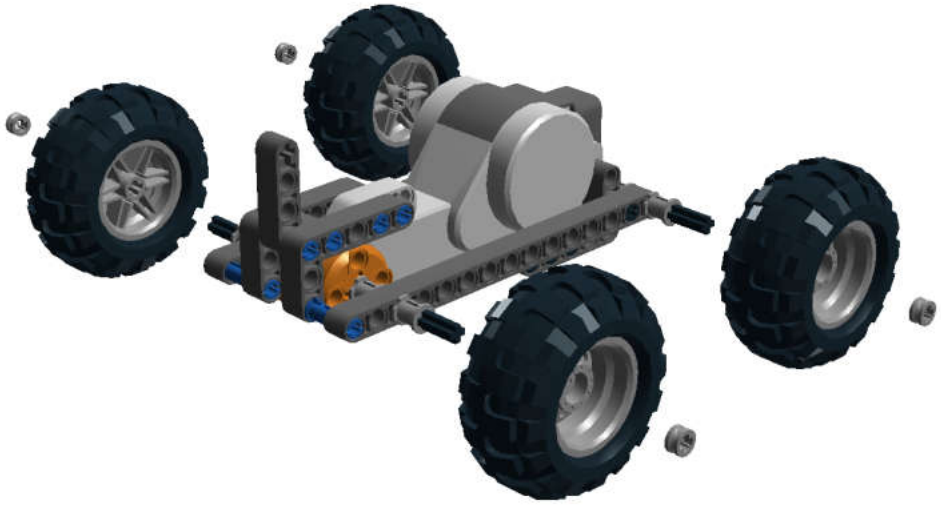


Рис. 3.10. Колеса крепятся так, чтобы не было трения с балками.

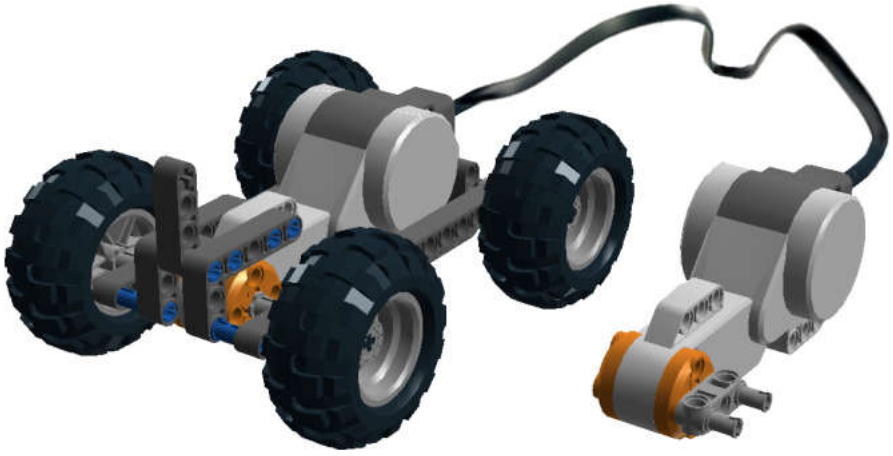


Рис. 3.11. Вращая управляющий мотор, добьемся движения тележки.

Управлять такой тележкой нетрудно, но, к сожалению, она связана с нами кабелем. Тем не менее стоит проехаться по столу, преодолеть препятствия и убедиться, что это возможно. Однако гораздо эффективнее по пересеченной местности движется тележка с полным приводом.

Полноприводная тележка

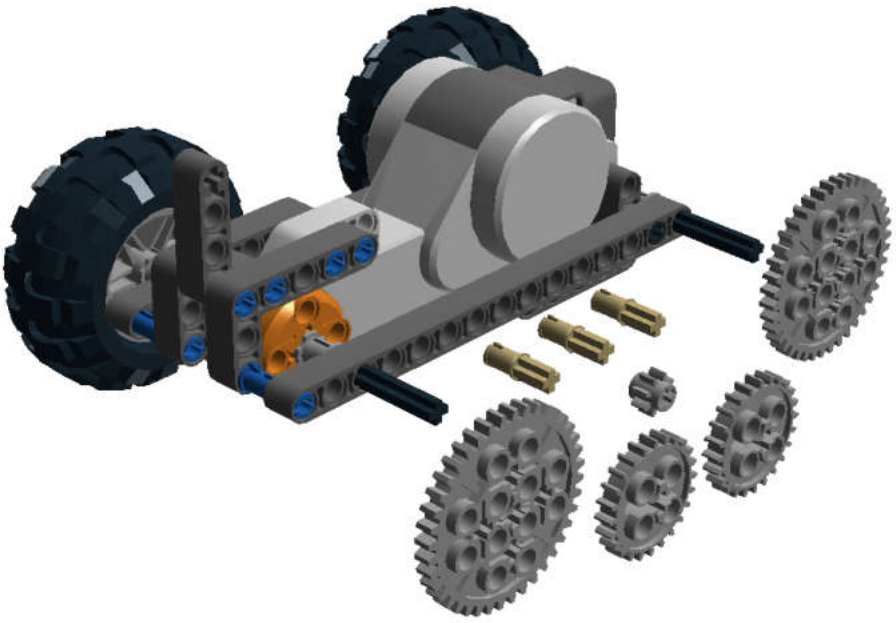


Рис. 3.12. Наиболее эффективное расположение шестеренок для полноприводной тележки.

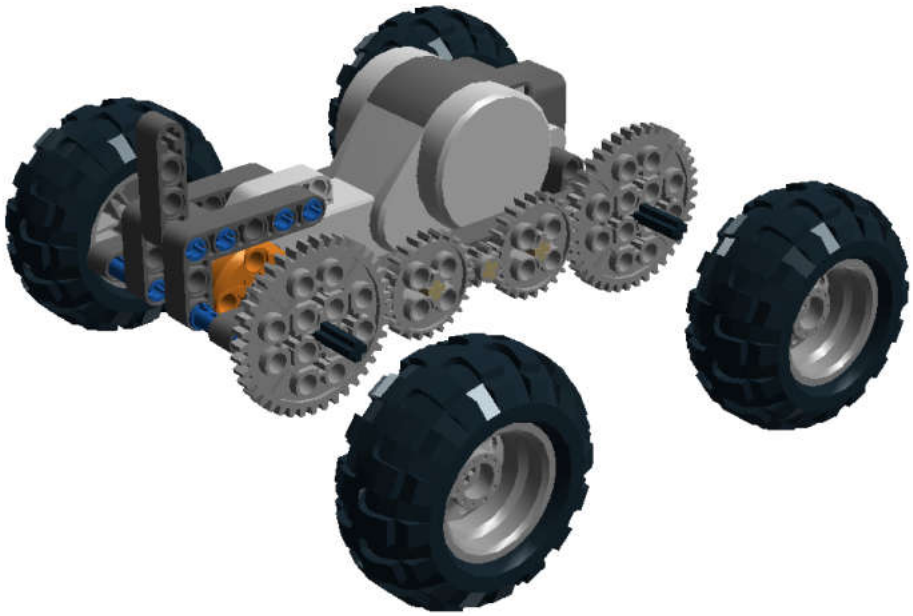


Рис. 3.13. Три из пяти шестеренок паразитные, но польза от них есть.

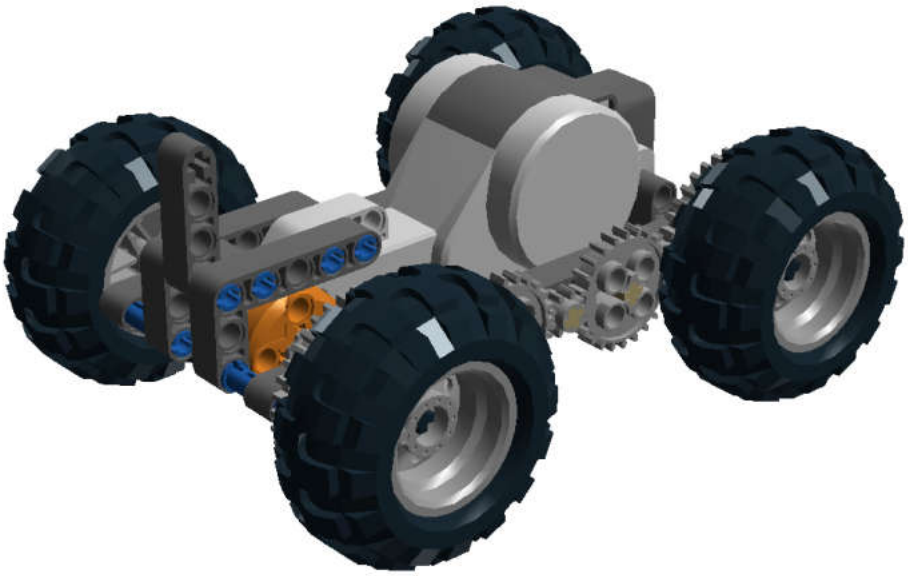


Рис. 3.14. При таком расположении колес возникает трение с соседними шестеренками.

Передаточное отношение между осями будет 1 : 1, поскольку три промежуточных шестеренки паразитные и влияют только на изменение направления вращения (рис. 3.13). Нечетное число паразитных шестеренок позволяет сохранить направление. Малая шестерня по центру не занимает пространство и не мешает преодолению бугристых препятствий.

Если колеса насажены слишком глубоко (рис. 3.14), может возникнуть нежелательное трение с соседними шестеренками, которые вращаются в противоположную сторону. Этого можно избежать: заменить 12-модульную ось двумя 8-модульными, состыковав их в оранжевом моторном цилиндре, или просто удлинить оси специальными втулками. Но об этом позже.

Итак, прототип «вездеходика» с ручным управлением готов.

Тележка с автономным управлением

Теперь давайте разберемся с полезным грузом. Для начала поставим задачу, чтобы тележка смогла вывезти сама себя. Микроконтроллер NXT с шестью аккумуляторами весит немало. Разместим его на корпусе тележки (рис. 3.15—3.20). В следующей версии массу груза можно будет довести до килограмма или двух.

Установка шестеренок и колес для полноприводной тележки уже знакома по предыдущей модели, поэтому вдаваться в детали не станем (рис. 3.21-3.23).

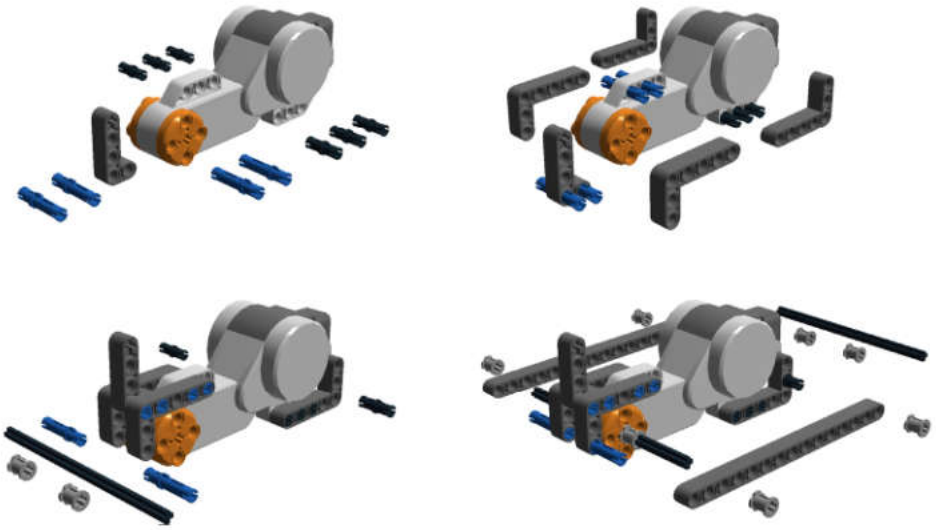


Рис. 3.15. Повторение основы тележки.

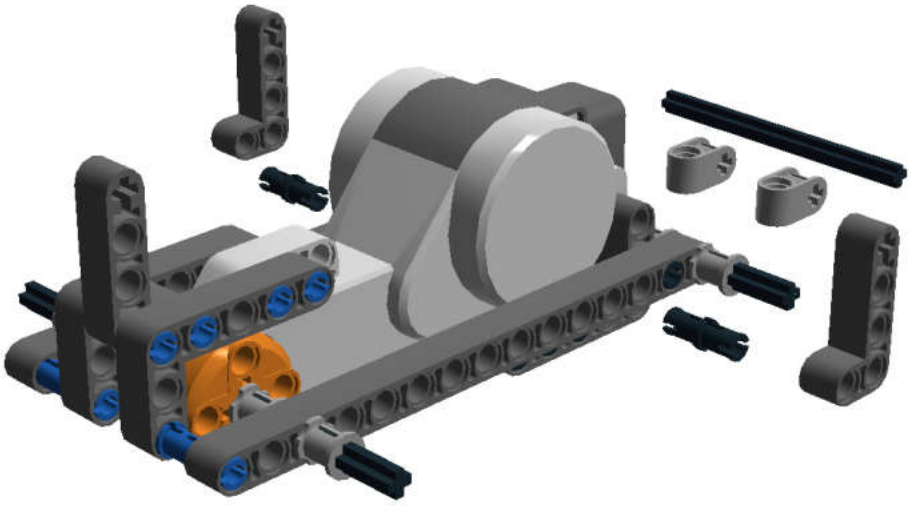


Рис. 3.16. Заднее крепление для NXT на базе одномоторной тележки.

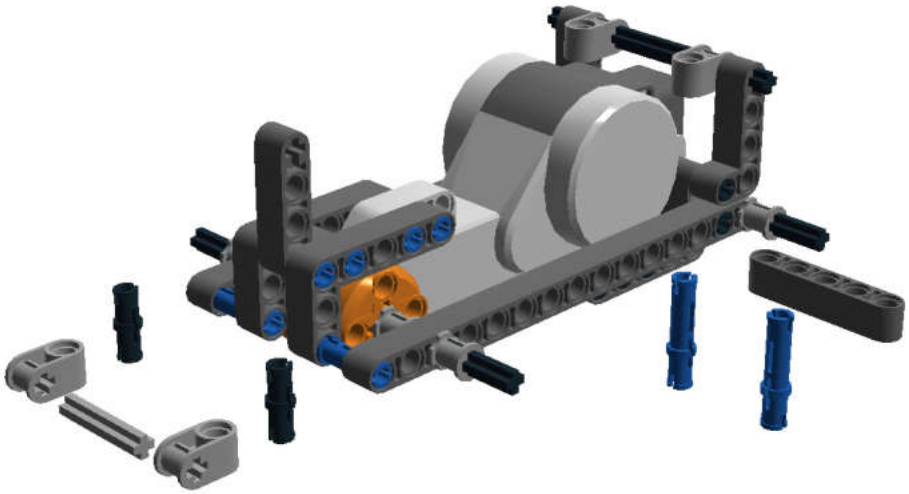


Рис. 3.17. Установка вертикальных штифтов для крепления к NXT снизу.

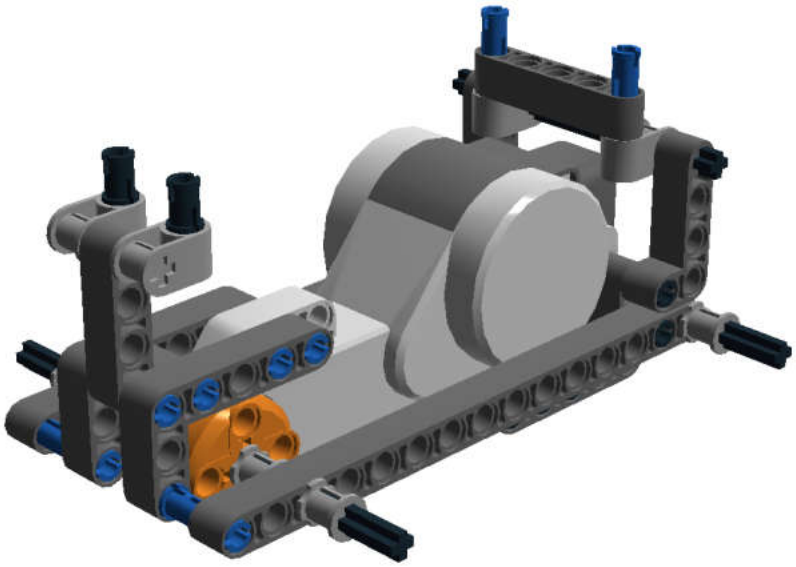


Рис. 3.18. Можно ставить контроллер.

Рис. 3.19. Для надежности колеса закрепляются полувтулками.

Рис. 3.20. Мотор подключается на порт B.

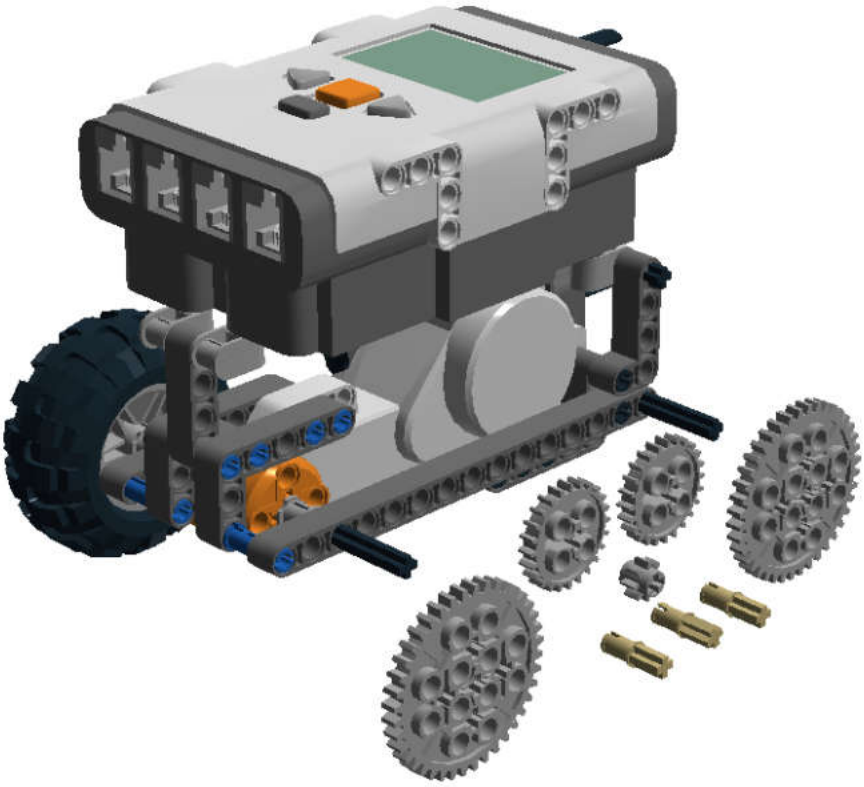


Рис. 3.21. Установка полного привода.

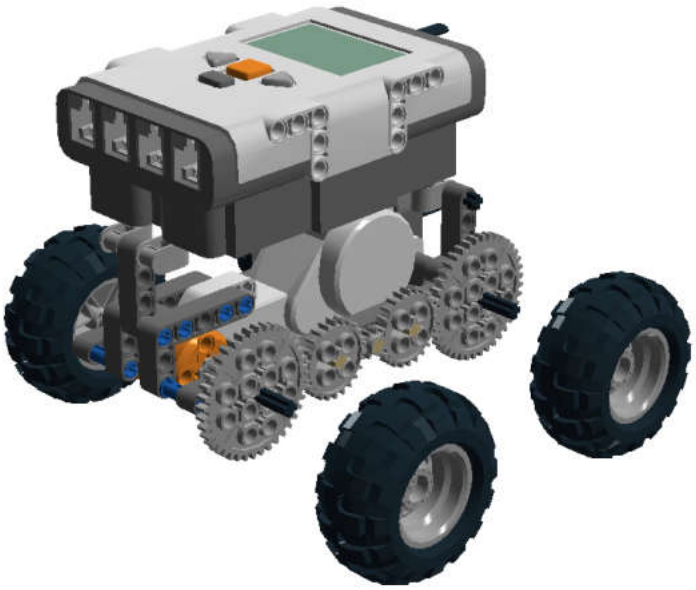


Рис. 3.22. Установка колес возможна с удлинителями осей.

Если вы до сих пор еще не начали программировать на NXT, то простую инструкцию по движению одно- или двухмоторной тележки найдете в начале данной главы. При запуске берегите пальцы – зубцы шестеренок соприкасаются с большой силой!

Рис. 3.23. Вид сбоку.

Тележка с изменением передаточного отношения

Итак, первая задача выполнена, тележка тронулась с места. Теперь попытаемся сделать из нее гоночный автомобиль. Понятно, что нагружать ее при этом пока не будем. Для увеличения скорости достаточно увеличить передаточное отношение (рис. 3.26).

Приступим к строительству. Необходимо снять с предыдущей модели колеса, шестеренки и несущие балки (рис. 3.24), модифицировав конструкцию (рис. 3.25—3.26).

Попробуйте поэкспериментировать. Можно заметить, что при достаточно высоком передаточном числе тележка просто не тронется с места: ее придется самим разгонять и подталкивать. А в приподнятом состоянии колеса будут крутиться быстро-быстро. Чего же не хватает? Очевидно, тяговой силы. Выиграв в скорости, мы потеряли в силе — моторам уже не хватает мощности для старта. Автомобилисты сталкиваются с этим при переключении коробки передач. Самое большое усилие развивается на низких передачах. Воспользуемся ими.

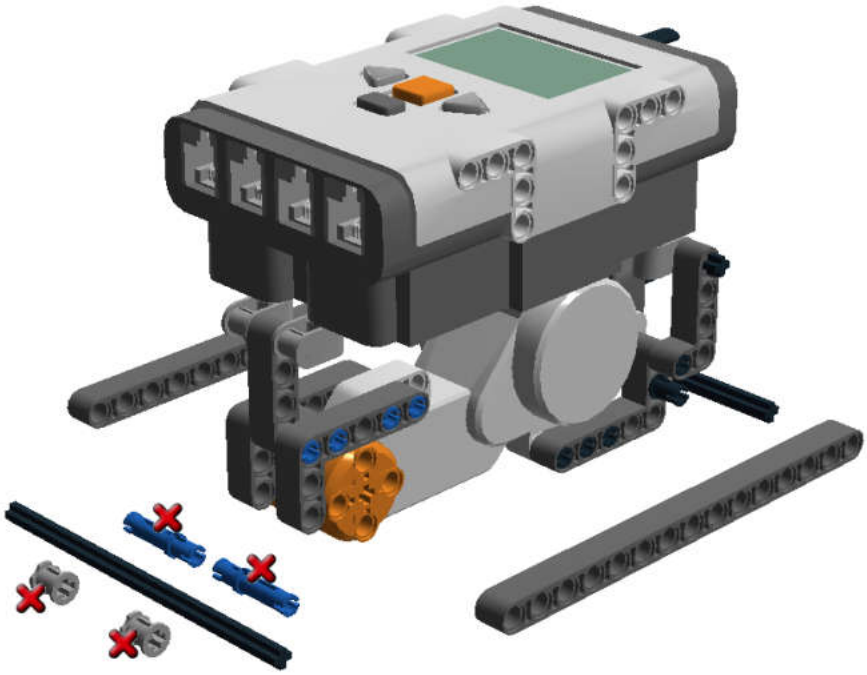


Рис. 3.24. Синие трехмодульные штифты и втулки будут убраны, а на их место встанет 12-модульная ось.

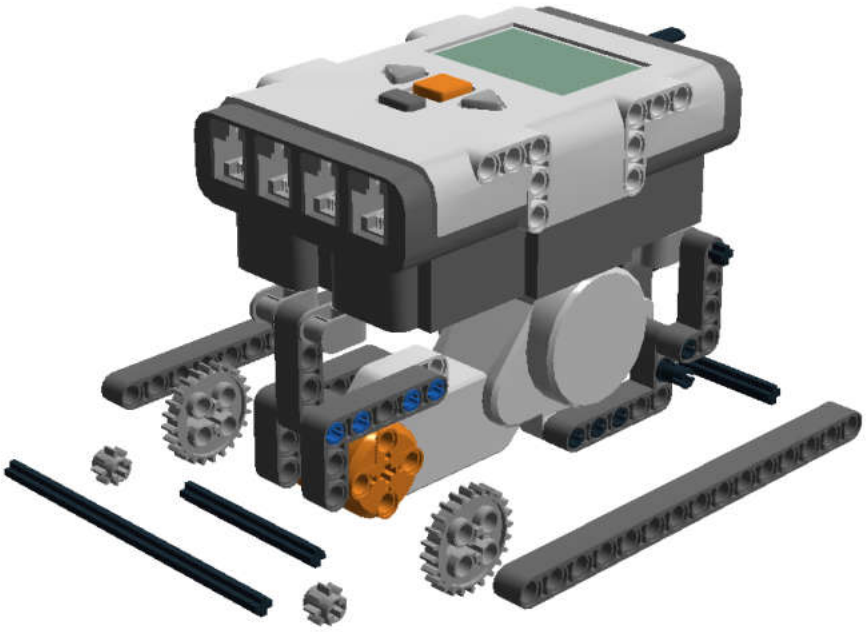


Рис. 3.25. В оранжевый диск мотора установливается 6-модульная ось.

Рис. 3.26. Повышающая передача 1 : 3 для скоростной тележки.

Робот-тягач

Следующая задачка — робот-тягач. Теперь наша тележка должна ехать не спеша, но двигать максимально возможный груз. Его можно расположить спереди, но для этого потребуется бампер, который защитит колеса, а можно и сзади прицепить на шнурке. Что следует поменять? Верно — опять передаточное отношение (рис. 3.27). Замените шестерни вновь поставьте робота на колеса.

Чем медленнее будет двигаться тележка, тем больший груз она сможет вывезти. Но и здесь есть ограничение. Когда груз лежит в кузове, он придавливает тележку к земле, сцепление колес с поверхностью увеличивается, все работает. Но если груз расположен вне тележки, ситуация резко меняется. Колеса будут прокручиваться так, словно тележка уперлась в стену. В такой ситуации надо искать золотую середину между передаточным отношением, массой тележки, массой груза и площадью поверхности колес.

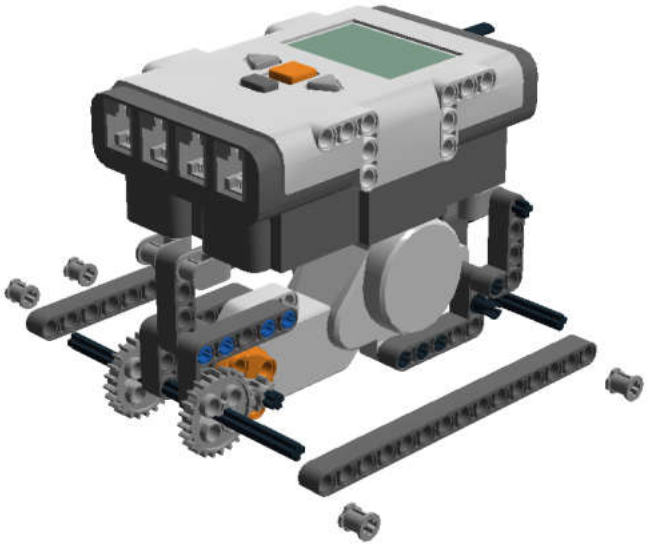


Рис. 3.27. Понижающая передача 3 : 1 для робота-тягача.

Еще одна подсказка. Для того чтобы увеличить площадь сцепления ведущих колес с поверхностью, можно построить полноприводный тягач (рис. 3.28—3.31).

Поскольку робот силовой, то потребуется закрепить шестерни, чтобы они не вываливались при больших нагрузках. Оптимальное решение — установка фиксирующей балки (рис. 3.30). Полученный таким образом редуктор будет обеспечивать не только большое усилие, но и высокую надежность при нагрузках.

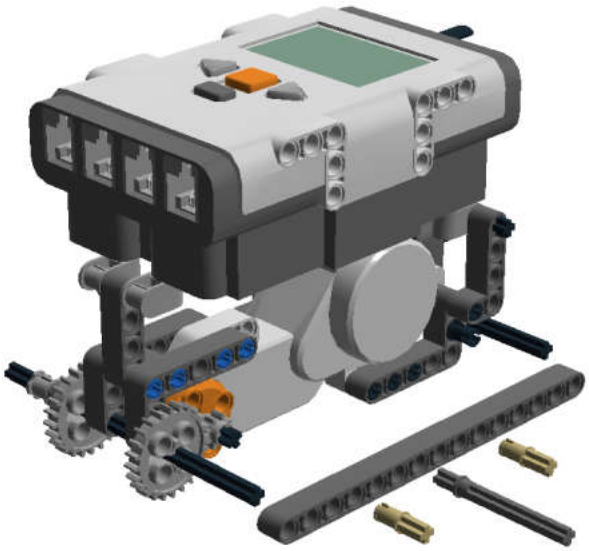


Рис. 3.28. Установка дополнительной оси с упором для фиксации шестеренок.

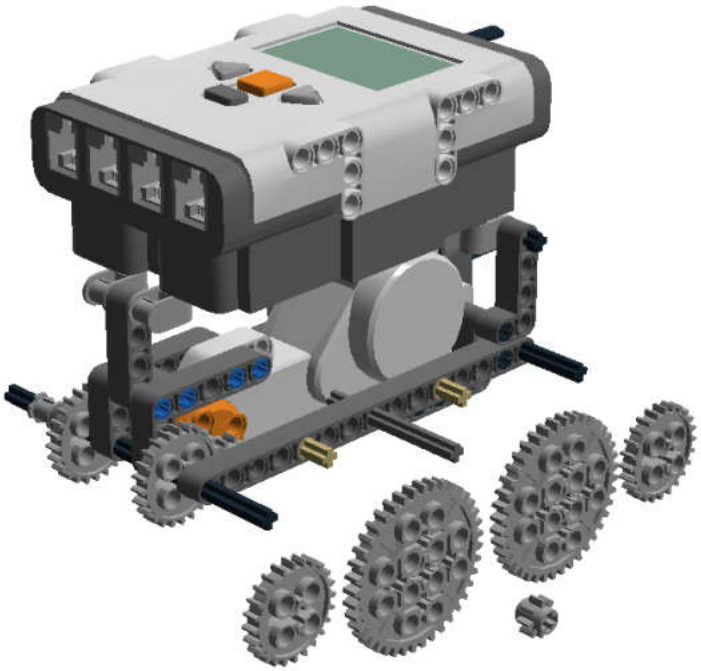


Рис. 3.29. Полный привод для робота-тягача на 2 модуля длиннее обычной тележки.

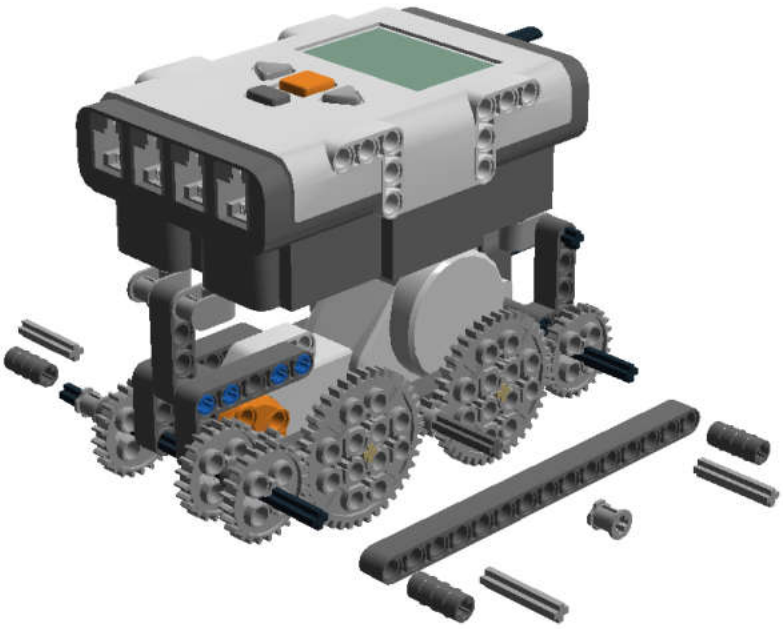


Рис. 3.30. Установка балки, фиксирующей шестеренки.

Рис. 3.31. Робот-тягач с полным приводом готов.

Любители экстрима могут погонять своего робота по пересеченной местности (естественно, желательно, не на улице): преодолевать горки, бугристые поверхности и мягкие покрытия. Какие особенности можно заметить в этом случае? Чтобы переехать горку, не достаточно быстро разогнаться, нужно уверенное неторопливое движение в течение длительного времени (рис. 3.32). Кроме того, требуется определенная конструкция днища тележки: между передними и задними колесами должен быть небольшой промежуток и достаточное пространство для высоких препятствий.

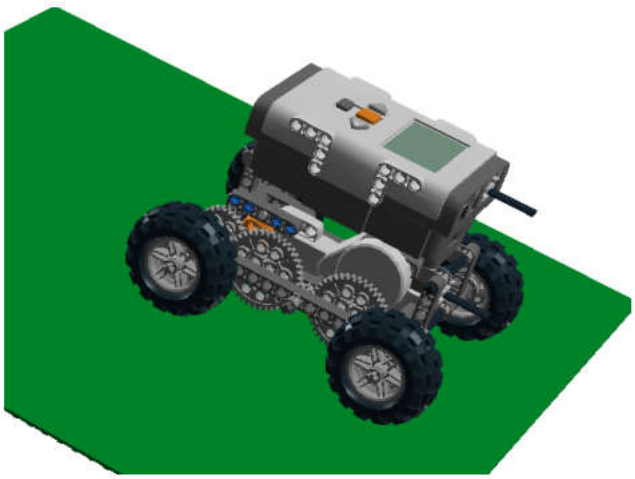


Рис. 3.32. Преодоление горки.

Если у вас есть два конструктора, можно устроить с товарищами перетягивание каната или состязания роботов Сумо.

Вот конструкция бампера, которую можно использовать, для того чтобы оторвать противника от пола (рис. 3.33—3.35).

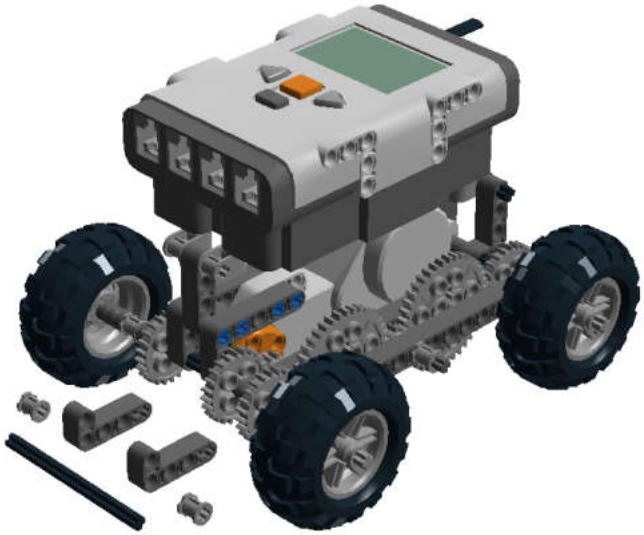


Рис. 3.33. Серую 3-модульную ось заменим на черную 6-модульную.

Рис. 3.34. Изогнутые балки – основа для бампера.

Рис. 3.35. Бампер не касается поверхности.

В созданной таким образом конструкции изогнутые балки не должны касаться поверхности. На них можно прикрепить небольшой кузов и тележка станет выполнять полезную работу.

Шагающие роботы

Введение

В представлении любителей фантастики робот — человекоподобное создание. Конечно, сейчас уже существует множество технологически сложных роботов, которые даже отдаленно не похожи на людей. Тем не менее, человек придуман достаточно хорошо для нашего мира, и в будущем неоднократно возникнет потребность заменить его в таких ситуациях, где потребуются качества, пока не востребованные в промышленности. Одно из них — умение ходить. Создать двуногого робота и заставить его эффективно перемещаться — задачка, над которой трудятся тысячи специалистов по всему миру. Уже сделан бегающий робот-андроид Asimo и множество его аналогов. Робот AlphaRex, рекламируемый Lego, тоже может перемещаться на двух ногах, но ходьбой это можно назвать с натяжкой. Однако даже его конструкция довольно

сложна (ее можно найти на компакт-диске, прилагающемся к наборам 8527 и 8547, в электронном учебнике, встроенном в среду программирования Lego Minstorms NXT).

Тема шагающих роботов очень обширна. Мы коснемся лишь малой ее части. Однако и это небольшое знакомство, как правило, вызывает восторг детей и удивление взрослых. Простота базовых конструкций и минимальное количество деталей позволяют собрать NXT «на ножках» практически из любого набора. Но кажущаяся легкость требует высокой точности, что многим не сразу дается. Вместо движения вперед робот начинает прихрамывать или «танцевать рок-н-ролл». В чем ошибка? Для того чтобы разобраться, пройдем весь путь.

Наиболее подходящим для начального изучения представляется четвероногий робот (рис. 3.32—3.46). Двуногому потребуется увеличивать площадь стопы, как у Alpharex, или вовремя переносить центр тяжести, что является сложной инженерной задачей. Большее число ног (8, 12, 16) может придать роботу маневренности, но значительно усложнит проект.

Робот на четырех ногах устойчив, как табурет. При правильном соединении он будет двигаться по прямой линии, притоптывая подобно маленькому слонику. Как и у тележки, его первая задача проста: идти вперед.

Сформулируем требования к конструкции:

— механизм должен стоять на поверхности, упираясь только на четыре конечности, каждая из которых не может совершать вращательное движение вокруг одного центра;

— движение конечностей должно быть возвратно-поступательным;

— в конструкции робота запрещено использование колес, соприкасающихся с поверхностью земли;

— конечности робота приводятся в движение одним мотором с помощью механической передачи;

— мотор подсоединен к источнику питания;

— центр тяжести робота должен быть смещен вперед по ходу движения.

Последнее требование нетрудно понять, если понаблюдать за тем, как мы ходим: каждый шаг — это падение. Раз падение, значит центр тяжести смещен вперед. Главное, вовремя выставить ногу.

О расположении центра тяжести стоит позаботиться заранее, чтобы не приходилось потом навешивать дополнительные грузы. Важно правильно (относительно расположения шестеренок) прикрепить микроконтроллер, в котором сосредоточена основная масса — «тело» робота.

Далее в разделе рассмотрены две похожие конструкции, предназначенные для разных наборов Lego. Если читатель не испытывает недостатка в шестеренках, то подойдет первая, ориентированная на набо-

ры 9797 и 8547. Если в руках читателя только конструктор 8547 версии NXT 2.0, оснащенный ограниченным числом зубчатых колес, то специально для него разработана вторая конструкция, в которой используются практически все доступные в наборе шестеренки.

Четвероногий пешеход

На рис. 3.36—3.48 приведена инструкция по сборке шагающего робота на базе набора 9797.

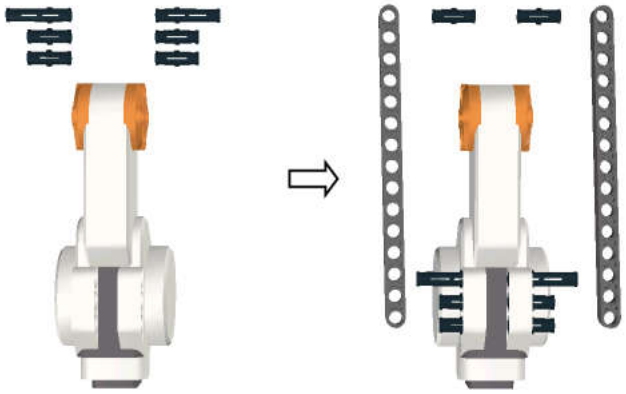


Рис. 3.36. Единственный привод обеспечит прямолинейное движение робота.

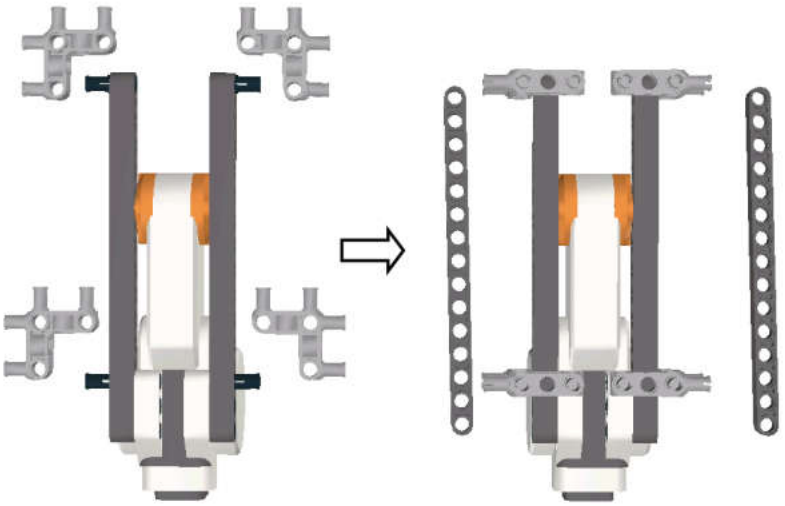


Рис. 3.37. 15-модульные балки крепятся к мотору и к угловым фиксаторам.

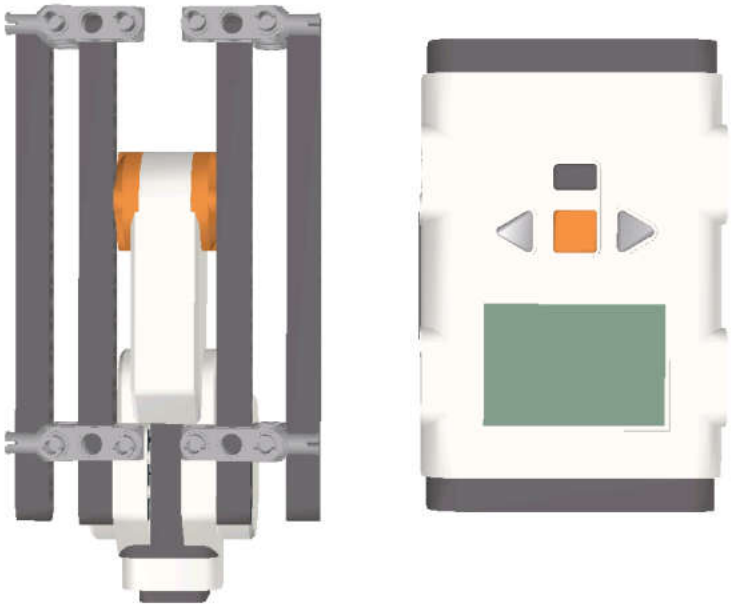


Рис. 3.38. NXT крепится к угловым балкам.

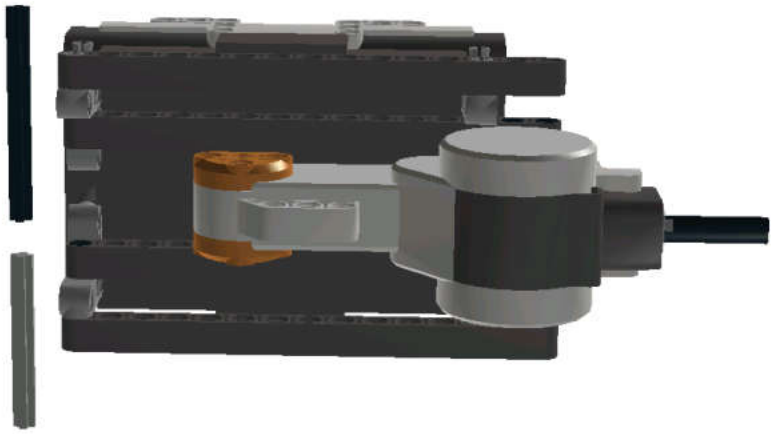


Рис. 3.39. Вид снизу. Ведущие оси будут разной длины: пять и шесть модулей.

Немного странным может показаться выбор длины осей: потребуется 5-и и 6-модульная ось (рис. 3.39). Они вставляются в мотор с двух сторон так, чтобы из соответствующих балок выступали части осей длиной, соответствующей ровно одному модулю.

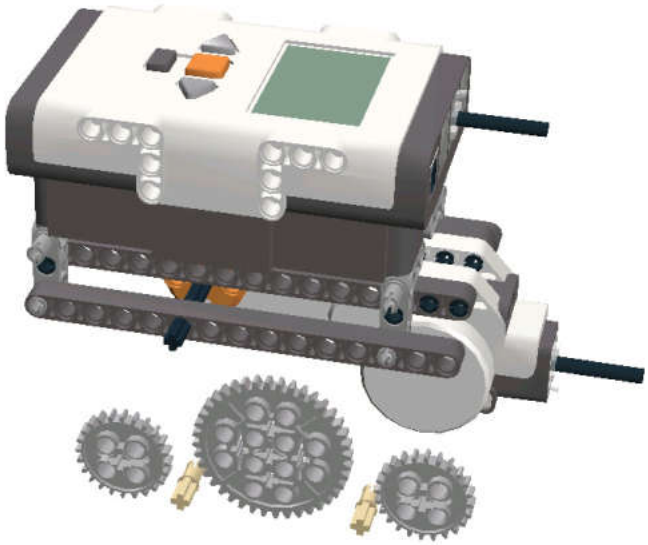


Рис. 3.40. На ведущие оси насаживаются 24-зубые шестерни.

На эти оси надеваются 24-зубые шестеренки, остальные устанавливаются на светлые штифты-полуоси: серые или бежевые (рис. 3.40—3.41). Синие для этой цели не подойдут, поскольку создают очень большое трение.

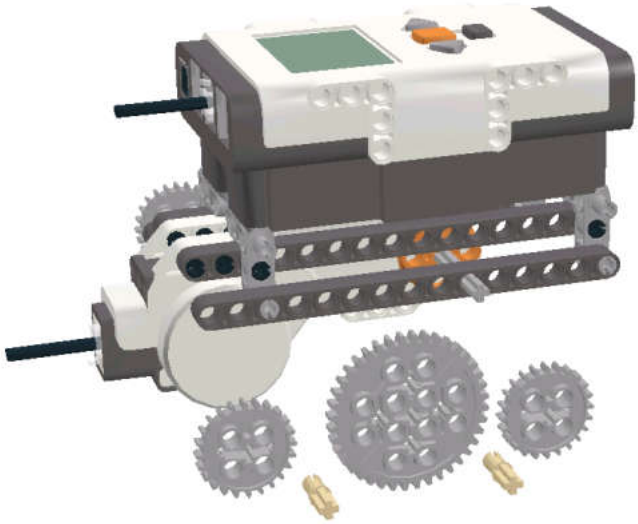


Рис. 3.41. Остальные шестерни крепятся на серые или бежевые штифты-полуоси.

Наиболее сложная задача — выравнивание шестеренок. Все 4 основные шестерни (24 зуба) должны быть расположены так, чтобы пары отверстий в них были строго параллельны друг другу (рис. 3.42). Расположение 40-зубых центральных шестеренок не играет роли.

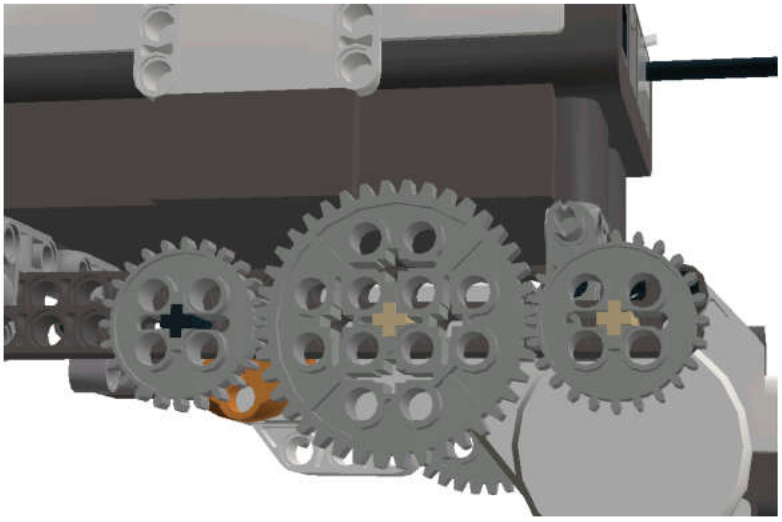


Рис. 3.42. Выравнивание крайних шестеренок.

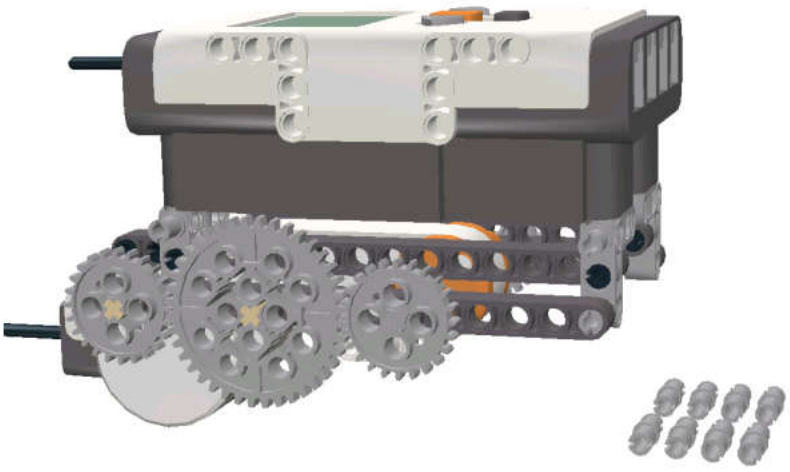


Рис. 3.43. Другой способ выравнивания с поворотом на 45°.

Рис. 3.44. Гладкие серые штифты вставляются в противоположные отверстия соседних шестеренок и на NXT.

Когда шестерни повернуты правильно, можно приступать к установке штифтов, на которые будут крепиться «ноги» (рис. 3.44).

Часть из восьми гладких штифтов вставляются в 24-зубые шестеренки симметрично относительно центра большой шестерни (рис. 3.44—3.45). И ровно наоборот с другой стороны: если справа от NXT они были ближе к центру, то слева должны быть дальше от центра.

Кроме того, по два штифта с каждой стороны устанавливаются в NXT в точности в соответствии с рис. 3.45.

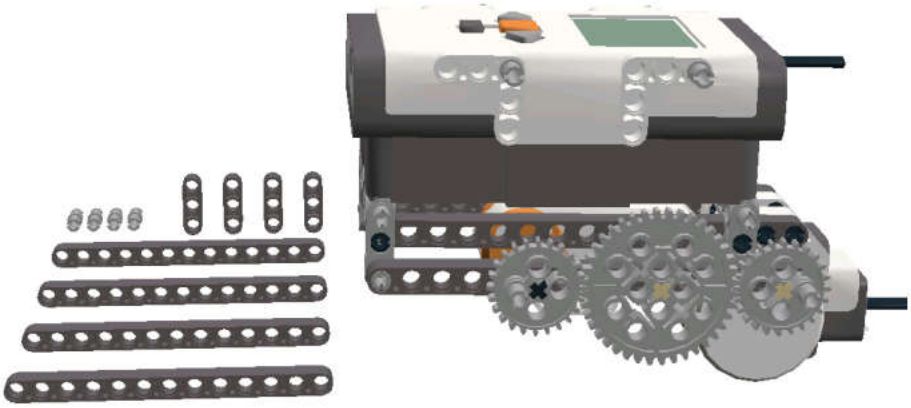


Рис. 3.45. Со второй стороны штифты вставлены в шестерни противоположно относительно первой.

Преобразование вращательного движения в поступательное совершается с помощью системы из двух коленец (балок), скрепленных гладкими штифтами (рис. 3.46). С подобными креплениями мы столкнулись, когда строили игрушку «Хваталка». По сути же нам придется построить подобие кривошипно-шатунного механизма, который используется, например, в двигателе внутреннего сгорания.

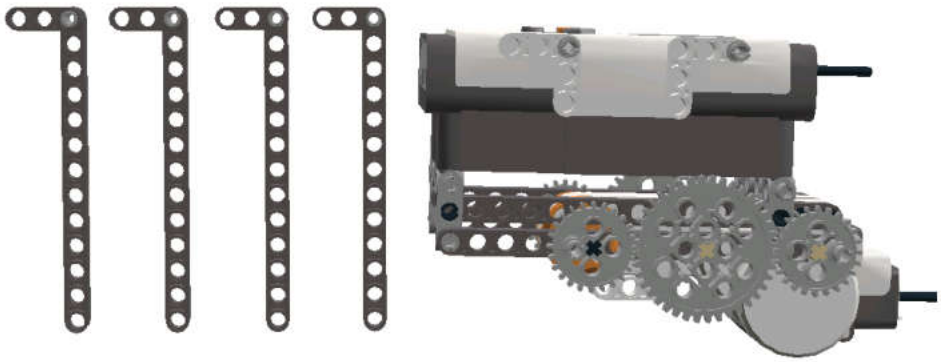


Рис. 3.46. Ноги робота должны быть одинаковыми, но их длину можно менять.

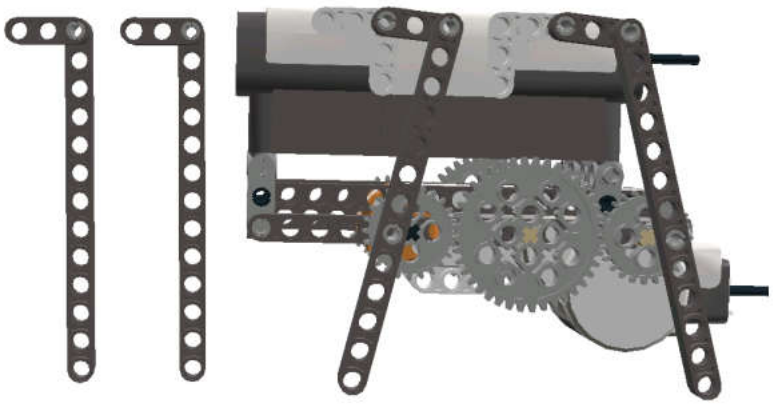


Рис. 3.47. Установка конечностей.

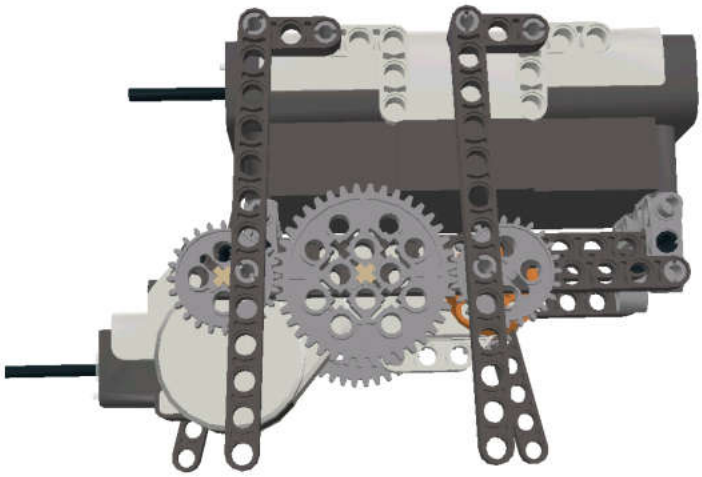


Рис. 3.48. С другой стороны конечности встают в противофазе.

Все коленца одинаковы. Нам остается их правильно подсоединить (рис. 3.47—3.48). Балки к шестеренкам следует крепить по принципу: диагонально-противоположные конечности движутся одинаково, а соседние отличаются от них на полоборота. При этом следует использовать гладкие штифты, они обычно имеют светло-серый цвет.

Для поддержания равновесия конечности робота могут быть немного расставлены вперед-назад (рис. 3.49). Для этого каждый «сустав» должен находиться над осью вращения соответствующей шестерни, немного смещенный к центральной. В соответствии с этим правилом надо подобрать длину коленца, а также точку его крепления к NXT.

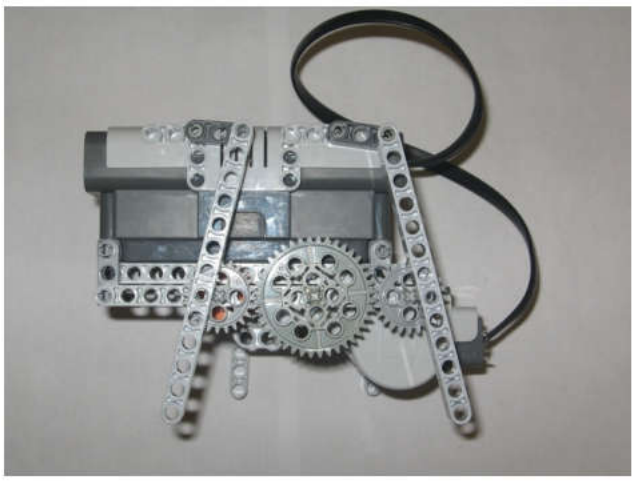


Рис. 3.49. Шагающий робот готов.

Для запуска шагающего робота потребуется такая же простая программа, как и для запуска тележки. Можно даже не загружать ее заново, а воспользоваться тем, что есть. Однако следует обратить внимание на то, что в предложенной конструкции мотор должен вращаться не вперед, а назад (рис. 3.50). Если запустить его вперед, робот будет пританцовывать на месте, почти не перемещаясь.

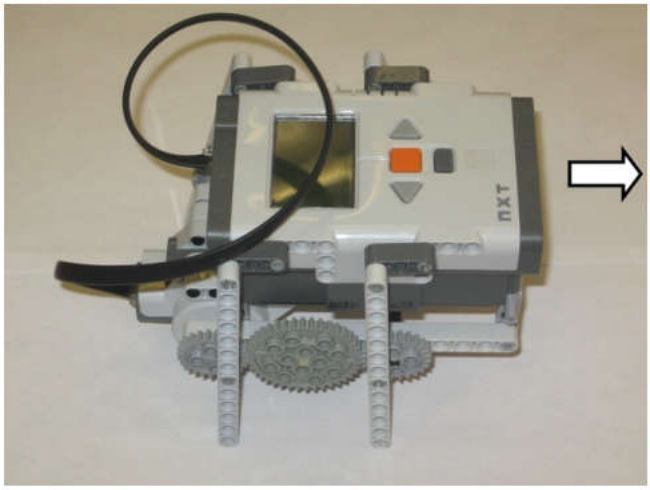


Рис. 3.50. При движении «хвостик» находится сзади.

Итак, робот пошел. Теперь вам представляется свобода творчества. Варьируя длину конечностей, размер «плеча» и «колена», смещение центра тяжести, можно получить самые неожиданные эффекты в «походке» робота. Можно нацепить «башмачки» или даже колесики с храповым механизмом.

Если есть два набора, стоит провести небольшой конкурс «Гонки шагающих роботов». Правила просты: пройти по коридорчику шириной 20—40 см расстояние в один-два метра. Для эффективного прохождения этой дистанции потребуется небольшая доработка конструкции, которая предлагается для следующего робота.

Универсальный ходок для NXT 2.0

Набор Lego Mindstorms NXT 2.0 неприятно удивил любителей робототехники малым числом шестеренок (их там всего шесть штук). Тем не менее и на такой элементной базе можно построить полноценного шагающего робота (рис. 3.51—3.68). Принцип движения останется тот же, даже скорость немного повысится по сравнению с предыдущей моделью.

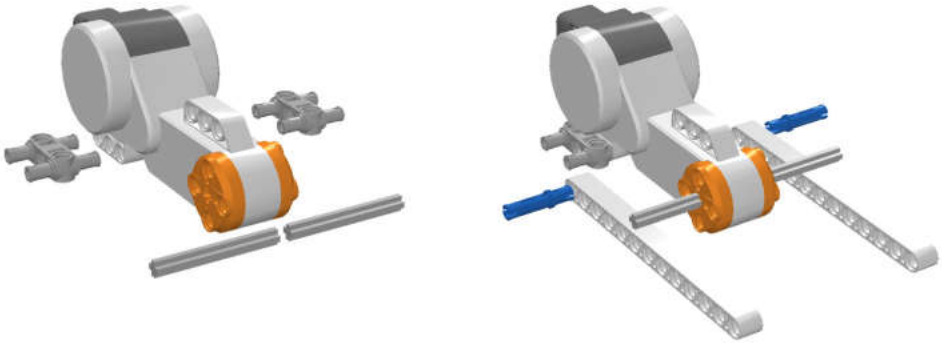


Рис. 3.51. Две 5-модульные оси вставляются в вал двигателя несимметрично: одна выступает на модуль больше.

Рис. 3.52. Несущие 13-модульные балки крепятся в трех точках.

Рис. 3.53. Шестерня 12-зубая ставится на бежевый штифт с поворотом на ползубца.

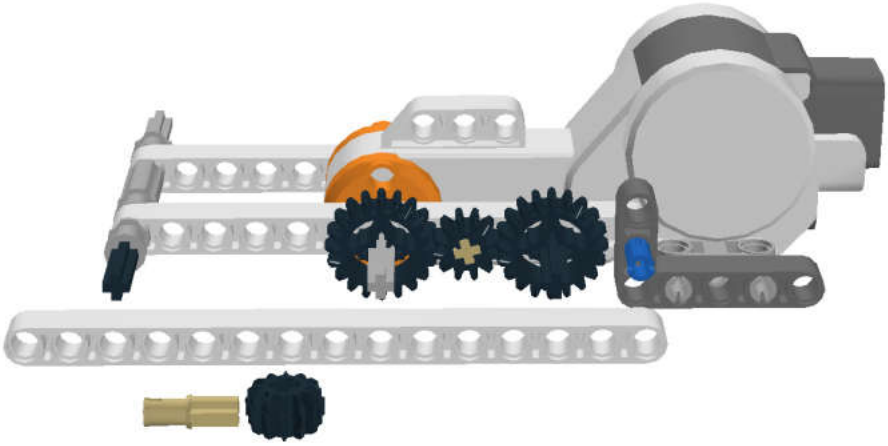


Рис. 3.54. Шестерни 20-зубые стыкуются с 12-зубой.

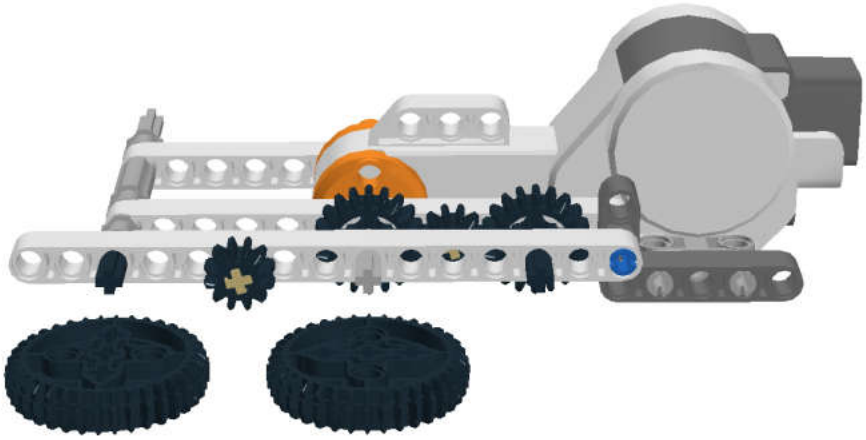


Рис. 3.55. Вторая 12-зубая шестерня тоже поворачивается на ползубца для стыковки с 36-зубыми.

Рис. 3.56. Черная двухмодульная балка заменит недостающую шестерню.

Рис. 3.57. Установка гладких серых штифтов для крепления конечностей завершает правую сторону.

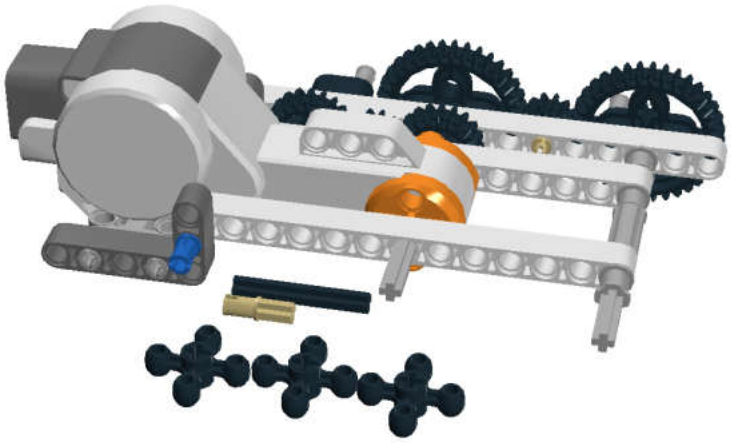


Рис. 3.58. На левой стороне используем 4-зубые шестерни.

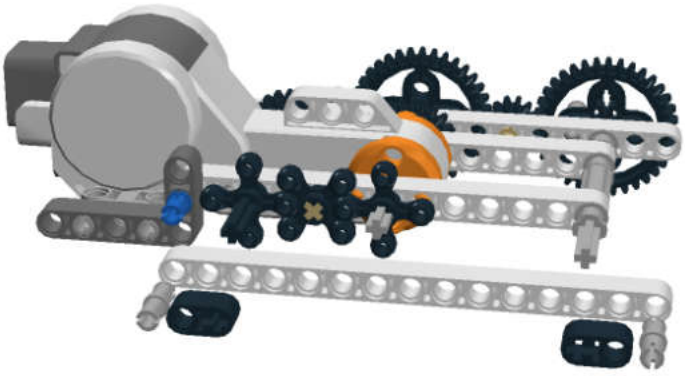


Рис. 3.59. Для стыковки 4-зубых шестеренок следует повернуть центральную.

Рис. 3.60. После установки черных 2-модульных балок с двумя серыми штифтами переходим к креплению для контроллера.

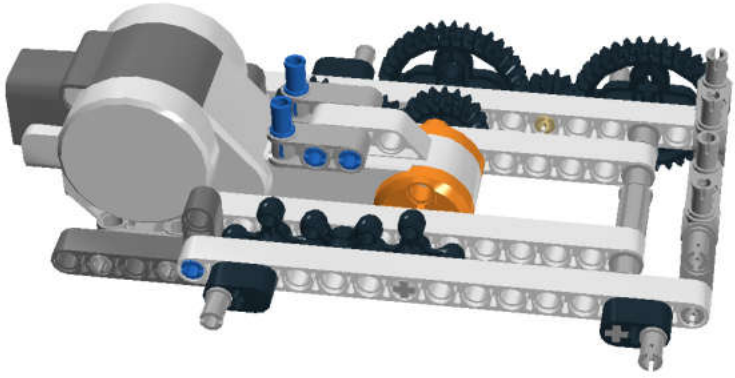


Рис. 3.61. Передняя часть крепится трехмодульными штифтами на петлю двигателя, задняя — на несущие балки одним штифтом с каждой стороны.

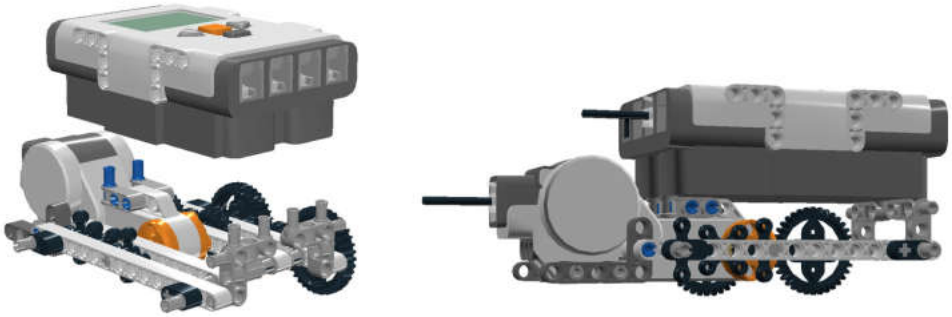


Рис. 3.62. Штифты размещаются на двигателе и несущих балках точно под нижними отверстиями на корпусе NXT.

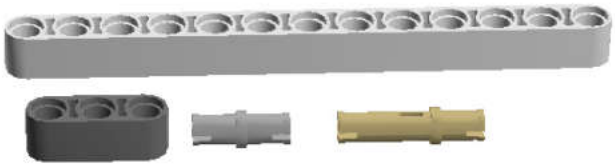


Рис. 3.63. Все конечности состоят из однотипных деталей, штифты желательно использовать гладкие, но подойдут и черные фиксирующие.

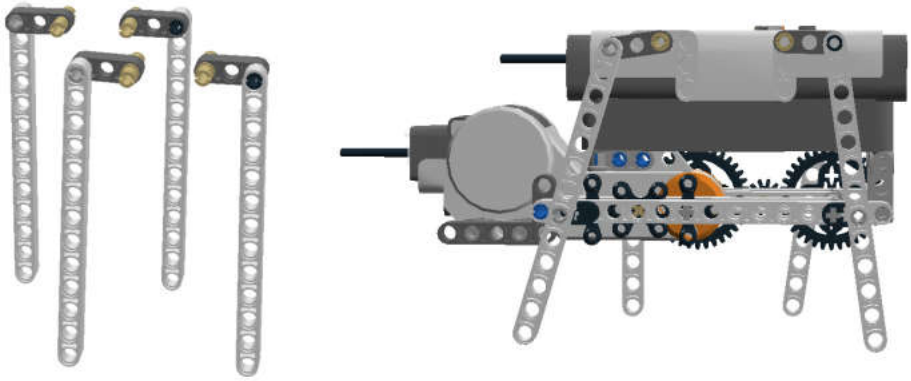


Рис. 3.64. Конечности крепятся на 6 отверстие снизу, а верхние суставы на второе и четвертое отверстия на борту NXT.

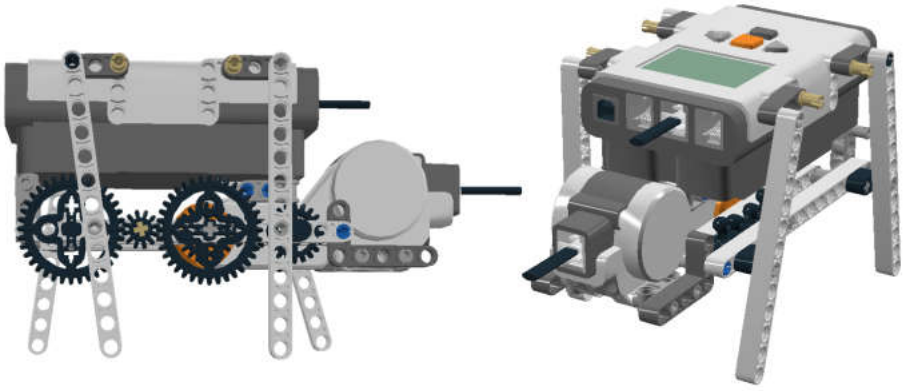


Рис. 3.65. Перед у робота получился со стороны мотора. Запуск — мотор назад.

Рис. 3.66. Балки 11-модульные на черных штифтах крепятся под двигателем.

Рис. 3.67. Три вертикальных оси для установки колесных дисков.

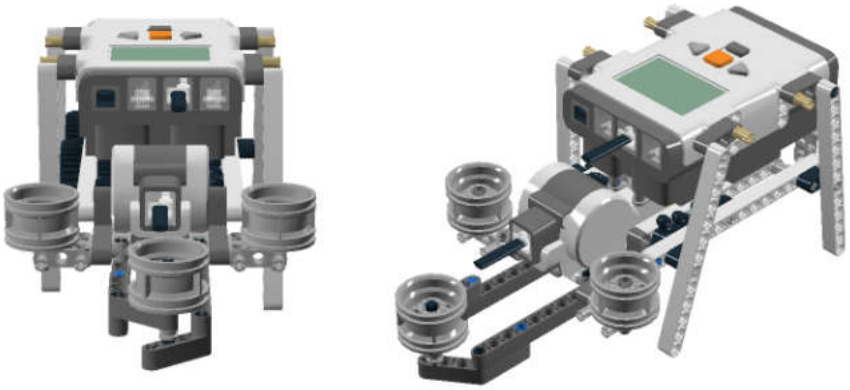


Рис. 3.68. Такой робот интересен тем, что может путешествовать, самостоятельно обходя стены.

Маятник Капицы

В 1940-х годах академик, будущий нобелевский лауреат по физике П. Л. Капица провел эксперимент, демонстрирующий, что верхнее неустойчивое положение равновесия маятника становится устойчивым, если ось подвеса маятника вибрирует в вертикальном направлении с достаточно большой частотой. Не вдаваясь в анализ уравнений математической модели маятника, попробуем подобрать оптимальное соотношение всех составляющих конструкции для достижения описанного эффекта (рис. 3.69).

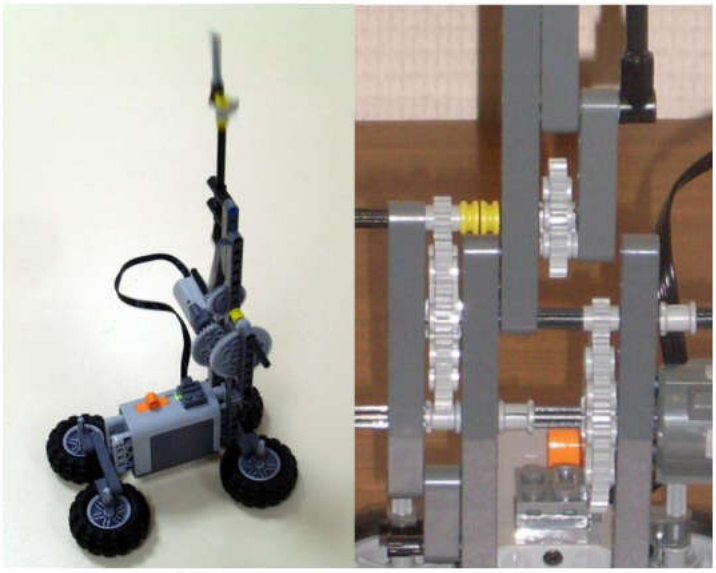


Рис. 3.69. Маятник Капицы из Lego Technic с передачей 1 : 25.

Главные принципы, из которых следует исходить при строительстве маятника, — это минимизация трения и оптимизация частоты вибрации. Поскольку передача будет явно повышающая, на выходе мы получим высокую скорость, но любая помеха сможет заблокировать движение, тем более что вибрация будет осуществляться с помощью уже знакомого нам кривошипно-шатунного механизма, преобразующего вращение в возвратно-поступательное движение.

Сам маятник не следует делать черезчур тяжелым: если установка заработает, его можно будет попробовать удлинить. На начальном этапе в качестве маятника подойдет 5—6-модульная ось.

Как видите, для построения маятника Капицы потребуется понимание основного материала, пройденного при прочтении всей главы о конструировании. А в награду нас ожидает наглядный, парадоксальный и почти чудесный эффект.

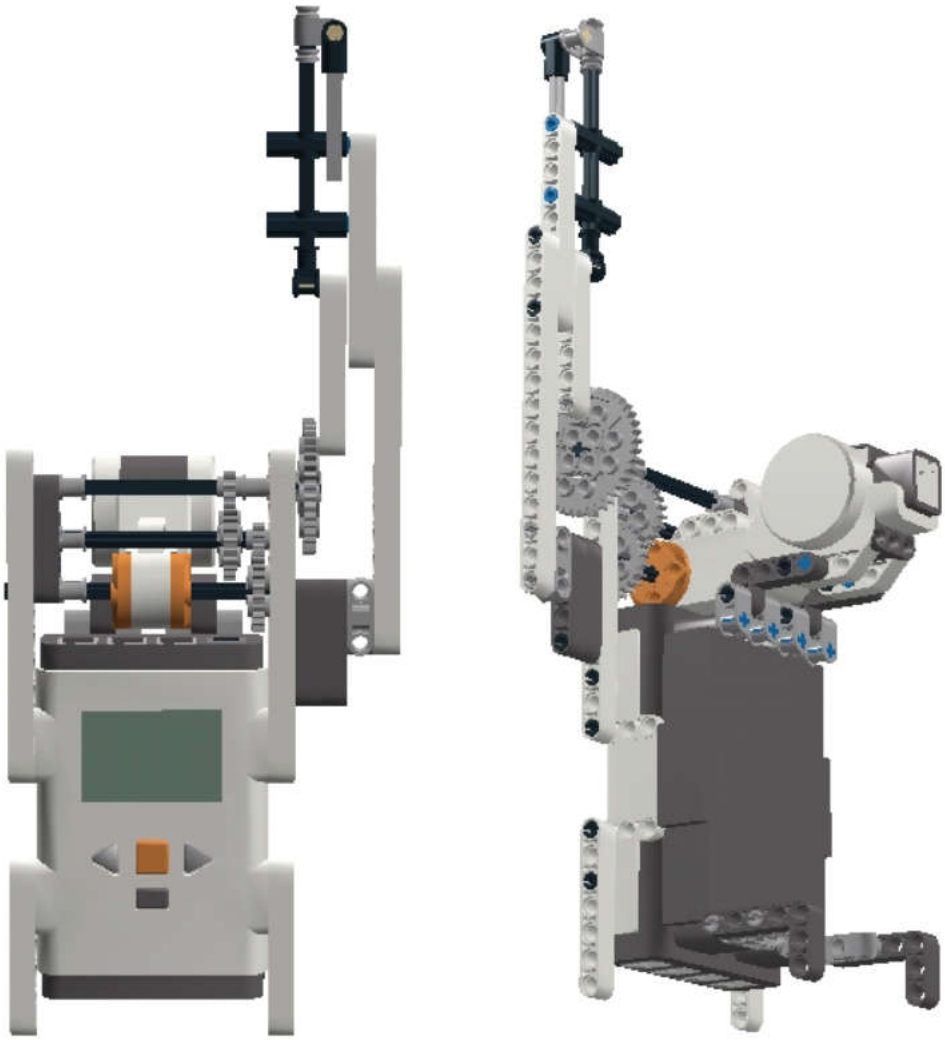


Рис. 3.70. Модель маятника Капицы, выполненная из деталей Mindstorms. Передача 1 : 9.

Двухмоторная тележка

Трехточечная схема

Это самая распространенная разновидность роботов. Тележка может быть с тремя точками опоры, две из которых — ведущие колеса, а третья — волокуша, или свободно вращающееся колесико (рис. 3.71). Такие модели являются базовыми для наборов 8527 и 9797. Инструкции по сборке прилагаеются. Если попытаетесь построить такую тележку самостоятельно, помните, что центр масс должен находиться не над волокушей, а ближе к ведущим колесам.

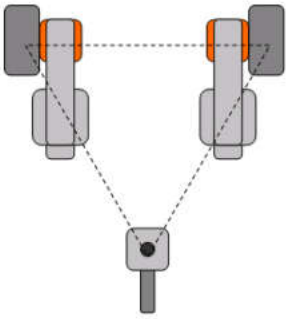


Рис. 3.71. Схема трехколесной тележки с подвижным третьим колесом.

Именно по этой схеме построена стандартная тележка из наборов 8527 и 9797 (рис. 3.72). В инструкциях этих наборов есть небольшие различия, но суть одна.

Рис. 3.72. Стандартная основа для робота из набора 9797.

Для тех, кто не хочет ограничиваться базовыми конструкциями, рассмотрим несколько примеров крепления моторов к NXT. От них можно отталкиваться при создании собственных роботов. Второй пример любезно предоставлен Центром инженерной поддержки образования на сайте *http://www.legoengineering.com* [4].

Простейшая тележка

Для придания устойчивости роботу имеет смысл поставить моторы по двум сторонам от NXT. Это несколько расширит корпус тележки (рис. 3.73).

Рис. 3.73. Широкая тележка — простейший вариант.

Рис. 3.74. Изогнутые балки для крепления моторов.

Предлагаемую конструкцию (рис. 3.74—3.81) можно делать вдвоем — бóльшая часть деталей устанавливается симметрично. А вот на подключение моторов следует обратить внимание. Для совместимости с алгоритмами, изложенными в этой книге, договоримся, что мотор B — слева, а мотор C — справа по курсу движения. На нашей тележке провода придется подсоединить накрест.

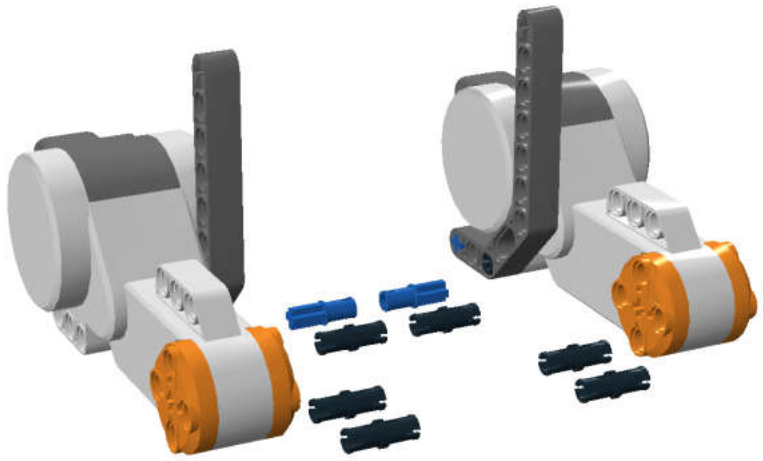


Рис. 3.75. В зависимости от расположения балок может быть смещен центр тяжести тележки.

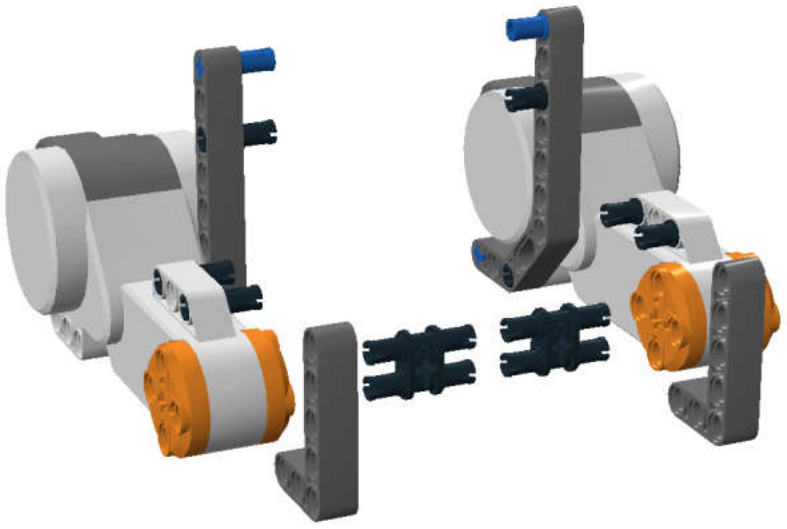


Рис. 3.76. Дополнительные крепления для придания устойчивости.

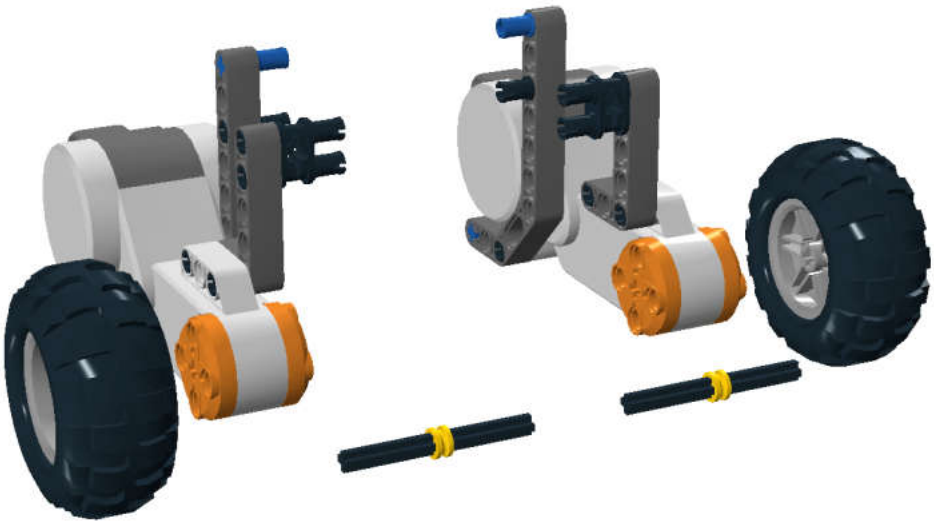


Рис. 3.77. Колеса устанавливаются на 6-модульные оси, втулки предохраняют от нежелательного трения шин о корпуса двигателей.

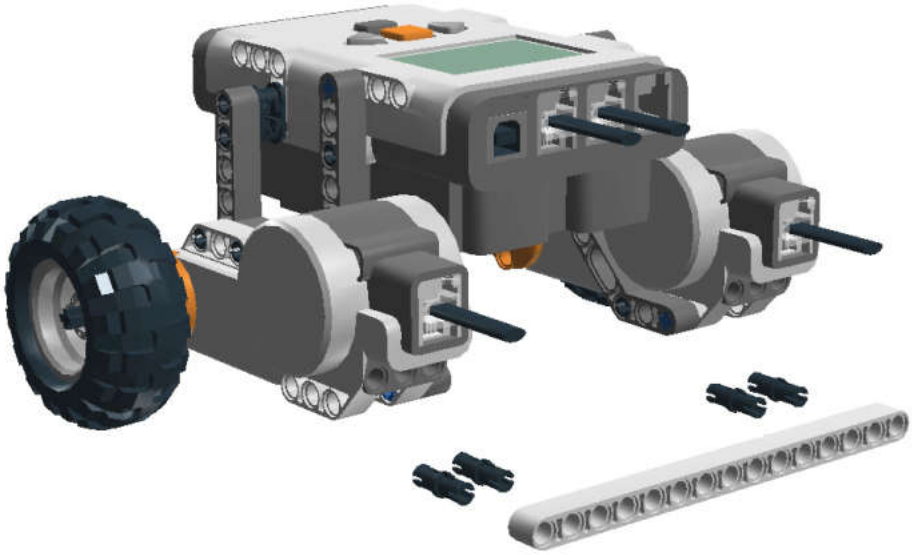


Рис. 3.78. Две половинки соединяются контроллером NXT и 15-модульной балкой, которая крепится к каждому мотору на 2 штифта.

Рис. 3.79. Элементы подвижного колеса. Длины осей — 3 и 5 модулей.

Рис. 3.80. Сборка заднего подвижного колеса. Обе оси должны вращаться свободно.

Простейшая конструкция тележки показана на рис. 3.81, однако в ней есть пара недостатков. Корпус тележки расположен с небольшим наклоном вперед. Если убрать одну втулку из вертикальной оси подвижного колеса, корпус выровняется, но тогда тележка потеряет возможность двигаться назад: колесико начнет цепляться за балку. Замена втулки на полувтулку решит проблему лишь отчасти.

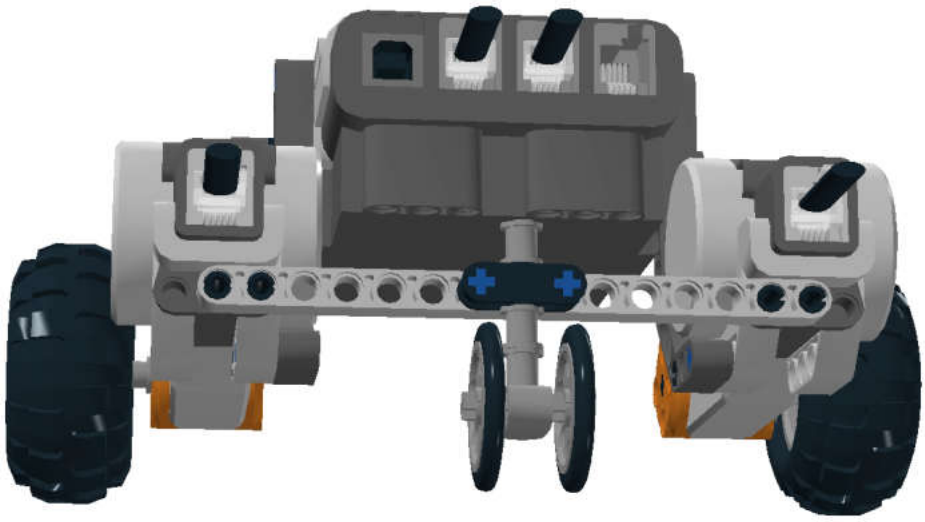


Рис. 3.81. Тележка готова. Она будет слегка наклонена вперед.

Для построения строго горизонтальной тележки воспользуйтесь инструкцией на рис. 3.82—3.85. Задняя 16-модульная балка будет заменена на конструкцию из угловых балок, которая обеспечит более высокий подъем подвижного третьего колеса.

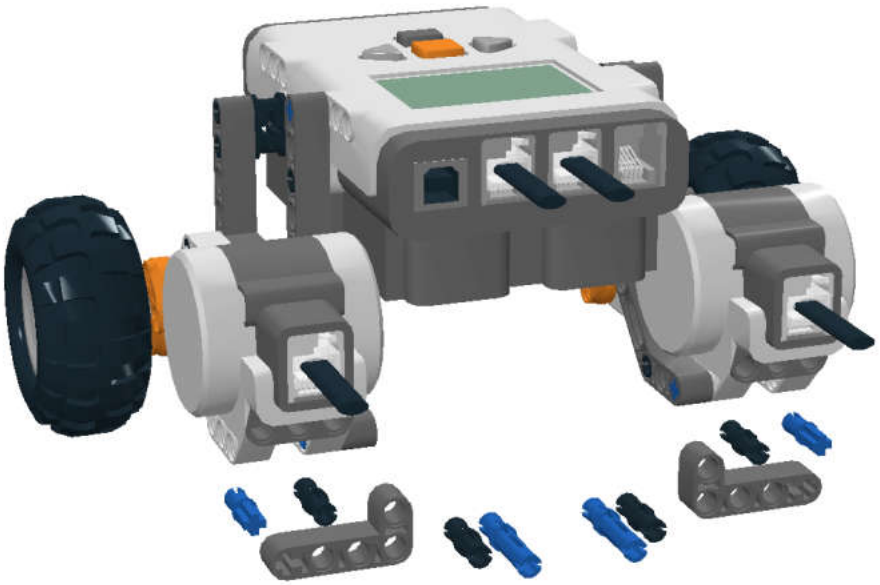


Рис. 3.82. Немного усложним конструкцию, подняв заднее крепление подвижного колеса.

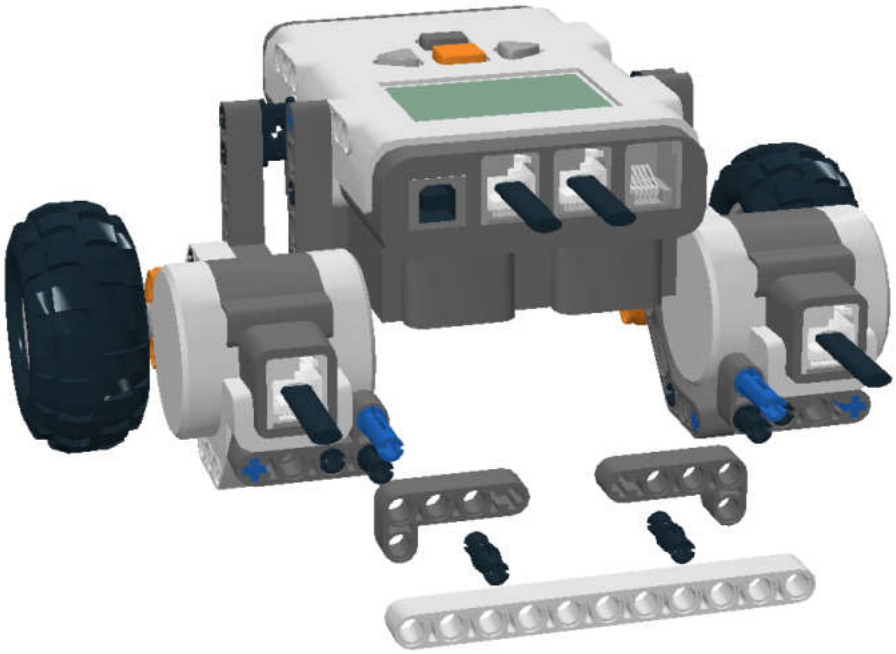


Рис. 3.83. Еще пара уголков, к которым крепится задняя балка.

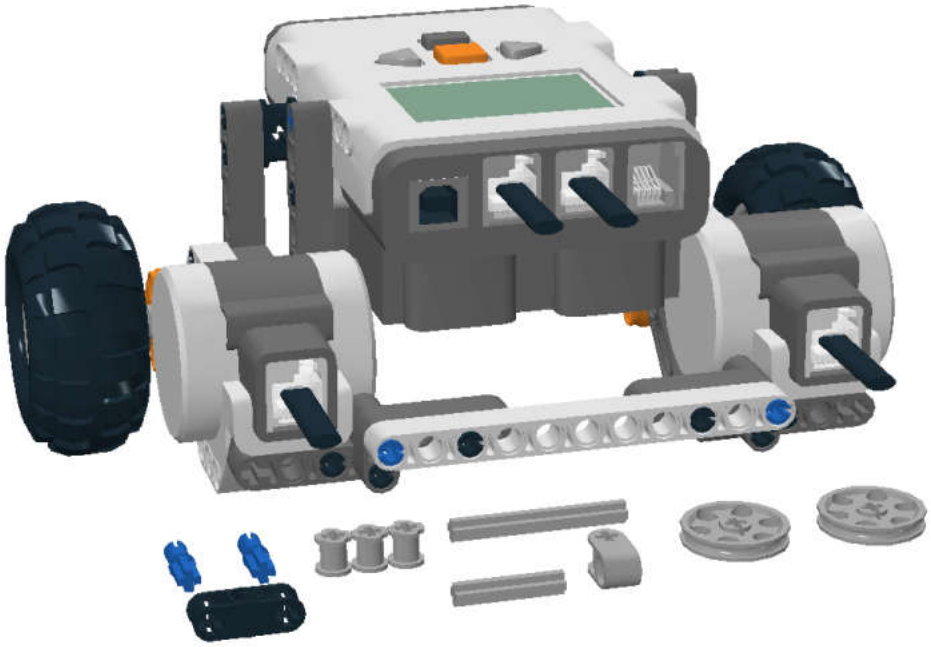


Рис. 3.84. Шины на диски подвижного колеса можно не надевать.

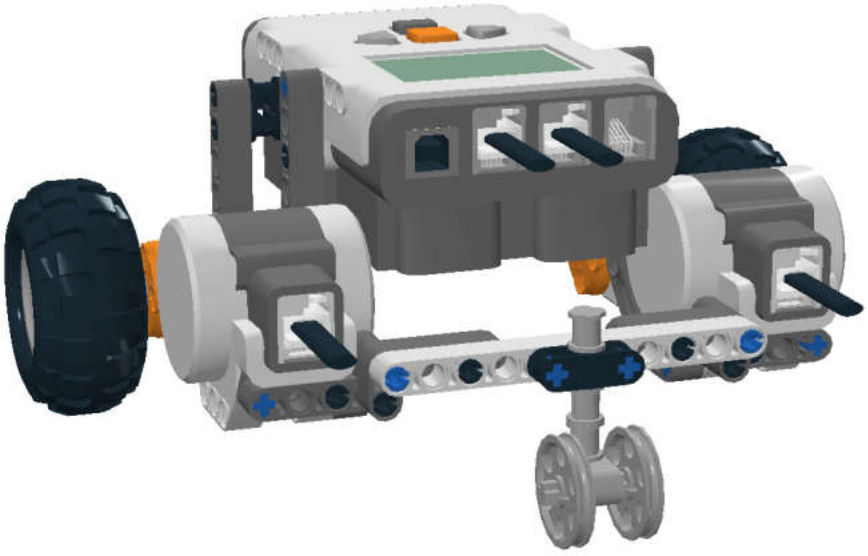


Рис. 3.85. Корпус такой тележки расположен горизонтально и колесико вращается вокруг вертикальной оси свободно.

Программирование без компьютера

Операционная система NXT содержит встроенную мини-среду программирования NXT Program (рис. 3.86, *а*). Эта среда настроена на конкретную конструкцию трехколесной тележки и позволяет составлять простейшие программы из пяти или менее команд (рис. 3.86, *в*). Понятно, что каких-либо сложных алгоритмов из пяти блоков создать не получится, но и этого достаточно для знакомства с основами управления роботом. Тем более что в некоторых из команд разработчиками заложены довольно сложные алгоритмы, скрытые от глаз пользователя.

Поскольку существует несколько сред программирования NXT, в каждой из них содержится своя операционная система для контроллера, которую будем называть «прошивкой». Внешне интерфейс большинства прошивок унифицирован, но исполнение некоторых команд может существенно различатся. За малым исключением при описании NXT Program будем опираться на прошивку от Robolab 2.9.4.

а *б* *в*

Рис. 3.86. Встроенная среда программирования NXT Program.

Чтобы минимизировать код программы, все моторы и датчики подключаются в конкретные порты (рис. 3.86, *б*), а в основных командах используются значения по умолчанию:

скорость моторов — 2 оборота в секунду,

расстояние до объекта — 25 см,

порог яркости света — 50 %,

порог громкости звука — 50 %.

Возьмите собранную тележку (например, рис. 3.49) и убедитесь, что левый мотор подключен к порту B, а правый к порту C. Зайдите в меню NXT Program и начните составлять программу. Для поиска блоков используйте стрелочки, для выбора — оранжевую кнопку, для отката на блок назад — темно-серую кнопку. Составьте программу по предложенному образцу (рис. 3.87).

Робот проедет две секунды вперед, столько же назад и остановится. Теперь разберемся, что же произошло.

Команды NXT Program делятся на 2 типа: «Делай» и «Жди». Команды типа «Делай» могут быть расположены в 1 и 3 ячейках, а команды типа «Жди» во 2 и 4 ячейках. Пятая ячейка может быть задействована для остановки (Stop) или повторного выполнения (Loop) алгоритма, содержащегося в первых четырех ячейках.

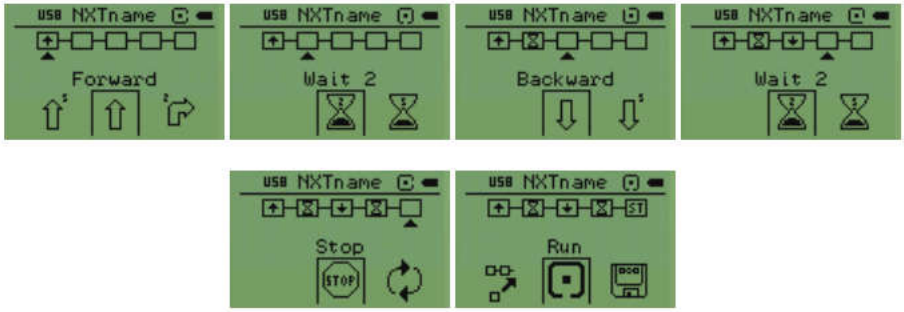


Рис. 3.87. Построение программы движения вперед-назад.

Попробуйте зациклить движение робота командой Loop. Для этого пару раз нажмите на темно-серую кнопку и вместо команды Stop выберите Loop, а затем снова Run (рис. 3.88).

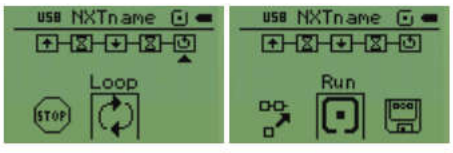


Рис. 3.88. Многократное повторение цепочки команд.

Если на контроллере стоит прошивка Robolab, при попытке остановить программу темно-серой кнопкой может произойти странное: моторы упрямо продолжат вращаться, не реагируя ни на что. Не стоит этого пугаться и немедленно выключать NXT. Можно решить проблему короче. Достаточно восстановить и запустить предыдущую программу с командой Stop в конце. Именно эта команда и остановит разбуянившегося робота. Почему так происходит? Давайте разберемся.

Таблица 3.1. Команды управления моторами в NXT Program

Блок	Назначение	Пример использования
Forward 	Включить моторы B и C вперед со скоростью около 2 об./с	 **Проехать две секунды вперед, затем 2 с назад и остановиться**
Backward 	То же назад	
Empty 	Остановить моторы B и C (в стандартной прошивке Empty не останавливает моторы, а просто пропускает ход)[1]	 **Двигаться с остановками: 2 с ехать, 2 с стоять**
Turn right 	Включить мотор B вперед со скоростью 2 об./с, мотор C вперед со скоростью 1 об./с	 **Двигаться по волнистой линии: 2 с направо, 2 с налево**
Back right 	То же назад	 **Проехать вперед 5 с, выполнить поворот назад направо за 2 с, остановиться**
Turn left 	Включить мотор B вперед со скоростью 1 об./с, мотор C вперед со скоростью 2 об./с	 **Двигаться по волнистой линии: 2 с направо, 2 с налево**
Back left 	То же назад	 **Двигаться по многоугольнику, проезжая стороны прямо 2 с, а углы с поворотом назад налево 2 с**

[1] Следует различать команду действия Empty и пустую команду ожидания Empty.

Команды управления моторами делятся на два вида: высоко- и низкоуровневые. Это деление условно — на самом деле низкоуровневым программированием занимаются только разработчики контроллеров и «прошивок» к ним. Однако и для начинающего робототехника в этих командах есть существенная разница. Низкоуровневые команды выполняются быстро, почти мгновенно. Зато последствия их выполнения программисту приходится контролировать самостоятельно. Например, включив мотор, надо обязательно выключить его через некоторое время, иначе он будет вращаться, пока не сядут батарейки. Команды высокого уровня сами по себе являются сложными алгоритмами: контроль за включением, синхронным вращением и выключением двигателей в них уже реализован разработчиками.

Как же отличить эти команды в NXT Program? Очень просто. Низкоуровневые команды обозначаются стрелкой без цифры, их всего шесть (табл. 3.1). Остальные стрелки с цифрами относятся к командам высокого уровня (табл. 3.2).

Надо признать, что даже в наших условно низкоуровневых командах реализована синхронизация вращения двух двигателей и поддержание заданной скорости. То есть достаточно придержать рукой один из них, второй немедленно затормозится. В главе об автоматическом управлении читатель узнает, как это происходит.

Для выполнения следующего упражнения с помощью темно-серой кнопки откатите курсор в начало программы и выберите команды, указанные на рис. 3.89.

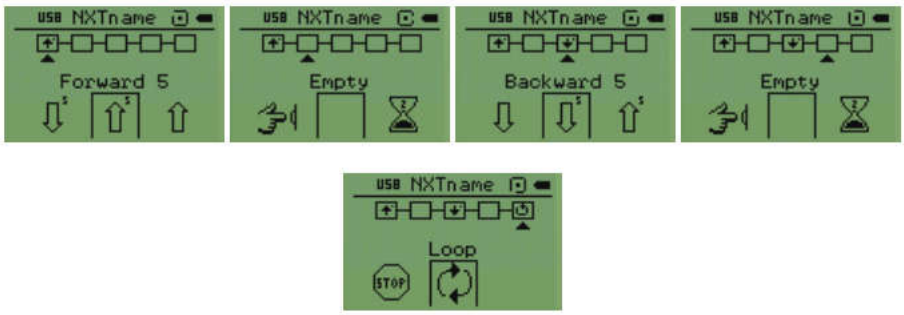


Рис. 3.89. Движение вперед-назад с помощью высокоуровневых команд.

После запуска программы робот начал выполнять практически те же действия: ездить вперед-назад. Однако для этого не потребовалось ставить песочные часы: ожидание встроено в команды управления моторами. Только ожидание чего? Почему над стрелкой стоит цифра 5? Предположение, что робот едет вперед 5 с, опытным путем быстро отметается. Быть может, это пять оборотов моторов? Давайте проверим. Если приложить к пройденному роботом пути рулетку или длинную

линейку, получится, что он проезжает немного меньше одного метра. От чего зависит пройденный путь? Понятно, что от размера колес. Диаметр колеса указан на шине: 56 мм. Воспользовавшись формулой расчета длины окружности, получаем: $L = \pi \cdot D \approx 3.14 \cdot 56 \approx 176 мм$. Таким образом, за один оборот колесо проходит примерно 176 мм, а за пять оборотов — 880 мм. В реальности это число может варьироваться из-за инерции робота и проскальзывания колеса. Однако оно очень подходит к результатам опыта, как раз немного меньше метра.

Что еще особенного произошло при использовании высокоуровневых команд? Во-первых, робот стал плавнее тормозить, во-вторых, после принудительного прерывания программы моторы послушно отключаются. Это связано с тем, что процесс торможения и остановки запрограммирован разработчиками внутри самих команд.

Таблица 3.2. Высокоуровневые команды управления моторами

Блок	Назначение	Пример использования
Forward 5 	Включить моторы B и C вперед со скоростью 2 об./с, синхронно вращать 5 оборотов и плавно затормозить	 **Проехать пять оборотов вперед, затем столько же назад**
Backward 5 	То же назад	
Turn right 2 	В течение 1 с вращать мотор B вперед два оборота, синхронно с ним мотор C вперед один оборот, плавно затормозить	 **Двигаться по волнистой линии по два оборота на внешнем колесе, останавливаясь при смене направления**
Back right 2 	То же назад	 **Двигаться по «звездочке»: поворот налево вперед на два оборота, затем столько же направо назад**
Turn left 2 	В течение 1 с вращать мотор B вперед один оборот, синхронно с ним мотор C вперед два оборота, плавно затормозить	
Back left 2 	То же назад	 **Проехать 2 с вперед, выполнить точный поворот налево назад и остановиться**

Высокоуровневые команды в чем-то освобождают программиста, но не достаточно гибки в использовании. Например, они никуда не годятся, если робот должен выполнять действия до наступления определенного события. Зато ограниченные наборы движений программмируются коротко и точно. Особенно хорошо эти команды работают со стандартной трехколесной тележкой Lego Mindstorms NXT, изображенной на рис. 3.72. О том, как использовать высокоуровневые команды в среде Robolab читатель узнает в разделе «Моторы NXT» главы «Программирование в Robolab».

В совокупности с командами низкого уровня замечательно работают команды типа «Жди». Они ничего не совершают, но останавливают ход выполнения программы, пока не произойдет определенное событие: пройдет заданное время, датчик ультразвука «увидит» объект, датчик света «увидит» изменение освещенности, будет нажата кнопка датчика касания или датчик звука «услышит» звуковой сигнал. Это весь набор возможных событий для простейшей среды NXT Program, и для каждого их них есть своя команда ожидания, которая может быть расположена на втором или четвертом блоке нашей маленькой программы (табл. 3.3).

Таблица 3.3. Команды ожидания

Блок	Назначение	Пример использования
Wait 2 Wait5 Wait10	Ждать соответствующее время: 2, 5 ил 10 с, после чего передать управление следующей команде	Повторять бесконечно: {поворачивать 2 с налево, затем 5 с направо}. Траектория движения робота будет напоминать восьмерку
Touch	Ждать нажатия кнопки датчика касания. Срабатывает мгновенно, для ожидания повторного нажатия требуется пауза	Ехать вперед до касания, затем отъехать 2 с назад и остановиться
Object	Ждать показания датчика ультразвука меньше 25 см.	Повторять бесконечно: {ехать вперед, пока впереди не появится объект, затем развернуться назад направо в течение 2 с}

Блок	Назначение	Пример использования
Sound 	Ждать громкость на датчике звука более 50 % (в прошивках Robolab и RobotC работает нестабильно)	 Повторять бесконечно: {ехать вперед до звукового сигнала, затем повернуть назад налево и остановиться в ожидании следующего сигнала}. Эта программа будет работать со стандартной прошивкой NXT
Light 	Ждать значение датчика освещенности выше 50 %	 Двигаться вдоль границы черного и белого. Этот алгоритм заработает только на очень белом поле[1]
Dark 	Ждать значение датчика освещенности ниже 50 %	
Empty 	Пропустить ожидание	 Ездить по многоугольнику, на углах поворачивая назад налево. Все команды высокого уровня, ожидание не требуется

Промежуточное положение между командами типа «Жди» и «Делай» занимают звуковые сигналы: Tone 1 и Tone 2 (табл. 3.4). С одной стороны, во время звучания сигнала происходит задержка, с другой стороны контроллер в это время выполняет конкретное действие: издает звук. И, хотя для робота эти команды могут представляться совершенно бесполезными, разработчики поместили их в одну группу с блоками управления моторами.

Забегая вперед, следует заметить, что для человека звуковые сигналы, издаваемые роботом, очень важны — ведь это зачатки общения. Поэтому мы еще не раз вернемся к ним и увидим, что именно звук помогает робототехнику понять свое творение в ситуациях, когда другие средства исчерпаны.

[1] Приведенный алгоритм является одним из ключевых в нашей книге. В NXT Program он может работать нестабильно, что с лихвой окупается возможностями Robolab 2.9 и RobotC.

Таблица 3.4. Звуковые сигналы

Блок	Назначение	Пример использования
Tone 1 Tone 2 	Остановить моторы, издать звуковой сигнал, по его окончании передать управление следующей команде	 Повторять бесконечно: {ехать вперед, пока освещенность не превышает 50 %, затем остановиться, издать звуковой сигнал и ждать понижения освещенности}

Компактная тележка

Вот другой пример, пожалуй, самого компактного расположения моторов (рис. 3.90). В предлагаемой конструкции контроллер повернут перпендикулярно направлению движения робота, но это ни на что не влияет: расположение деталей существенно только при смещении центра тяжести. Рассмотрим основу тележки без крепления колес (рис. 3.91—3.94).

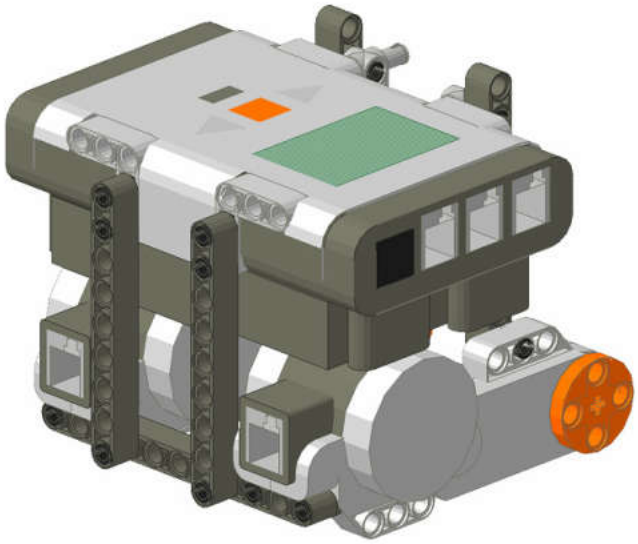


Рис. 3.90. Компактная основа тележки [4].

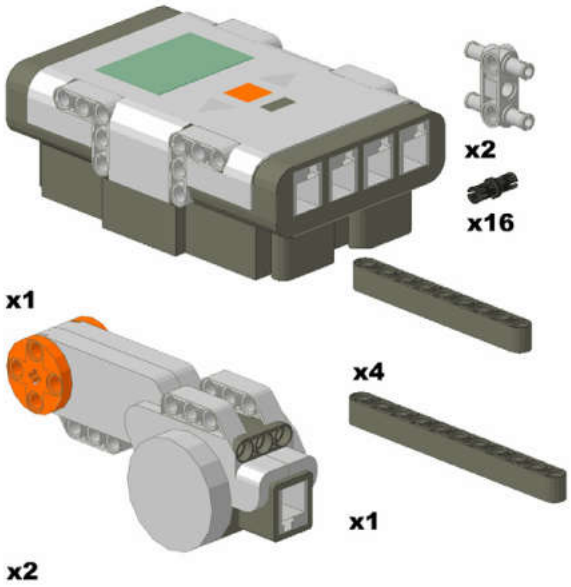


Рис. 3.91. Набор деталей для компактной основы [4].

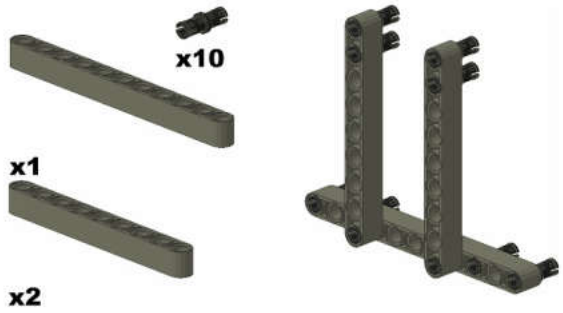


Рис. 3.92. Подготовка обвязки [4].

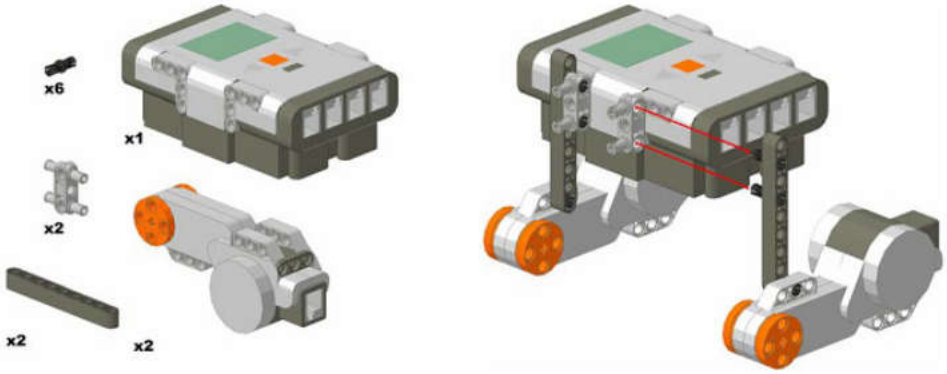


Рис. 3.93. Крепление моторов и NXT [4].

Собрав конструкцию обвязки из балок (рис. 3.92), отложите ее и соедините остальные блоки (рис. 3.93).

Далее остается прикрепить отложенную обвязку с противоположной стороны (рис. 3.94).

Компактная основа для тележки готова. Надо учесть, что при штатном Lego-аккумуляторе придется увеличить высоту вертикальных скрепляющих балок, поскольку он выступает на высоту одного модуля из корпуса NXT.

Если в наборе найдутся гусеницы, то лучшего варианта для гусеничной тележки не придумать.

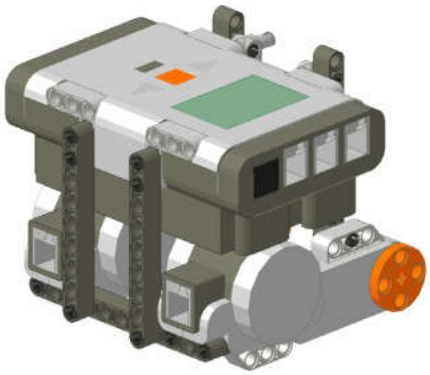


Рис. 3.94. Установка обвязки [4].

Полный привод

Можно сделать двухмоторную тележку, опирающуюся на четыре колеса, попарно соединенных с моторами. На улицах небольшого города встречаются подобные автомобили-погрузчики или электрокары. От своих собратьев с рулевым управлением они отличаются высокой маневренностью: способны выполнять поворот на месте. А от трехколесной тележки — высокой проходимостью.

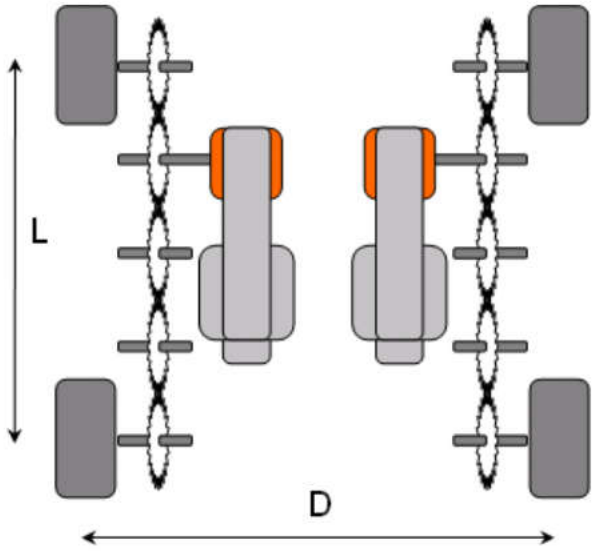


Рис. 3.95. Схема четырехколесной тележки [4].

Чтобы построить двухмоторную четырехколесную тележку с полным приводом, необходимо сконструировать механическую передачу с каждого мотора на оба боковых колеса (рис. 3.95). Если привод будет только на одно из колес с каждой стороны, то поворота, скорее всего, не будет из-за бокового трения второго колеса. Кроме того, расстояние между колесами по одной стороне L не дожно превышать расстояния между сторонами D: $L \leq D$.

Эта задача интересна, и читатель может попробовать решить ее самостоятельно, тем более что пример одномоторной тележки детально разобран. Остается расширить корпус добавлением второго мотора. Не забывайте только соблюдать четность и размеры шестеренок, чтобы колеса крутились в одну сторону с одинаковой скоростью.

Испробовав на готовой тележке небогатый арсенал NXT Program, читатель может приступить к серьезному программированию и выбрать задачу себе по вкусу: либо изучить какую-нибудь из предложенных сред (NXT-G в главе 4, Robolab в главе 5, RobotC в главе 6), либо, уже имея навыки программирования, освоить алгоритмы управления из главы 7, или сразу начать решать задачи для робота из главы 8.

Глава 4. Программирование в NXT-G

Введение

Способность NXT-робота выполнять любое задание, в чем бы оно ни заключалось, — следовать линии, бросить мяч или подметать пол, — не является интуитивной. Необходимо снабдить робота специальными инструкциями, которые будут диктовать ему, что делать; нужно запрограммировать робота. Программирование NXT включает в себя написание программы на компьютере и затем перенос в микроконтроллер, «мозг» робота, который запускает и выполняет программу. Программы должны сообщать NXT, как моторам работать, как датчикам получать информацию, как динамику воспроизводить звук и т. д.

Подходя к программированию NXT, первым делом обратим внимание на официальный язык NXT-G, включенный в пакет Lego Mindstorms NXT, поставляемый вместе с конструктором. NXT-G — это *графический язык программирования*, в котором программы можно создавать с помощью нажатия клавишей мыши и перетаскивания блоков кода на экране (G — graphical, графический). NXT-G довольно прост в использовании, но требует больших ресурсов компьютера и занимает много памяти. Далее рассмотрим еще две среды программирования NXT, которые работают быстрее, но могут оказаться менее доступны читателю.

В этой главе, во-первых, исследуем интерфейс NXT-G, который играет важную роль при создании программы. Во-вторых, обсудим некоторые базовые понятия языка NXT-G, которые необходимы для успешного программирования. В-третьих, рассмотрим готовые примеры из Robo Center.

Знакомство с NXT-G

При запуске Lego Mindstorms NXT появляется основной экран (рис. 4.1), откуда можно перемещаться к другим разделам. Начинающим настоятельно рекомендуем просмотреть краткие интерактивные руководства — Getting Started и Software Overview.

Чтобы увидеть интерфейс NXT-G и начать с ним работу, необходимо создать новую программу или открыть существующую.

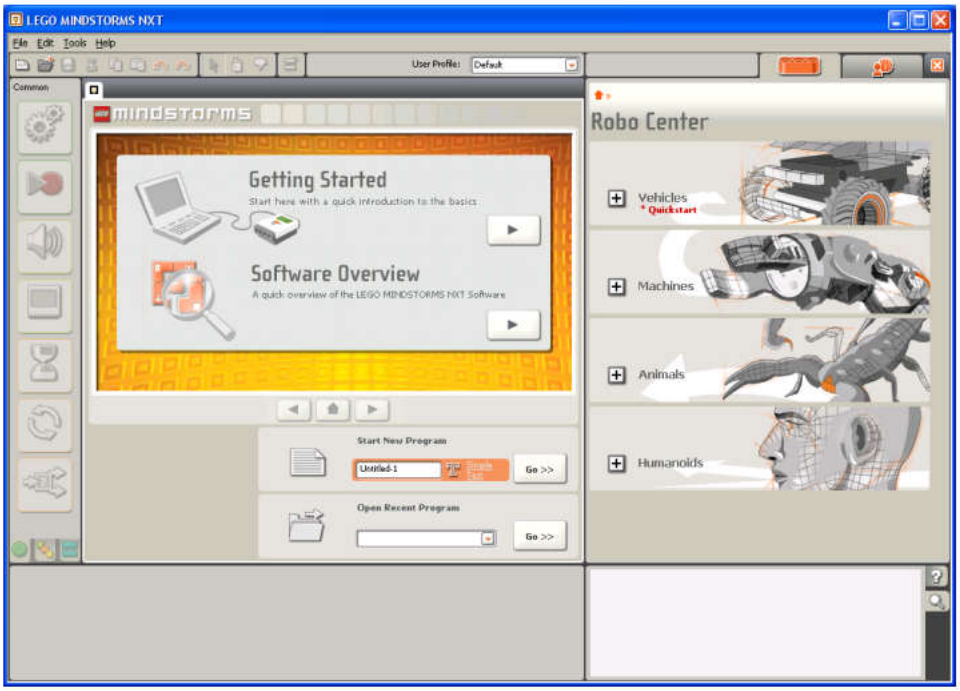


Рис. 4.1. Стартовое окно Lego Mindstorms NXT

Замечание. Предполагаем, что у читателя уже установлено программное обеспечение, поставляемое вместе с NXT-набором, и успешно проведено USB- или Bluetooth-соединение между компьютером и NXT. В случае несоответствия или сбоев операционной системы микроконтроллера следует воспользоваться разделом главного меню Tools → Update NXT Firmware.

Новая программа

Чтобы создать новую программу, напишите её название в рамке под словами *Start New Program* и затем нажмите кнопку **Go>>**. Чтобы открыть существующую программу, к которой вы недавно обращались, просто выберите программу из ниспадающего меню под словами *Open Recent Program* и затем нажмите кнопку **Go>>**.

Поскольку изначально не существует программы, напишите **First Program** в поле под словами *Start New Program* и нажмите кнопку **Go>>**, чтобы создать новую программу с именем *Test Program*.

Замечание. Можно создать новую программу, выбрав **File → New**, нажимая при этом сочетание клавиш Ctrl—N (для Windows) или CMD—N (для Mac).

Интерфейс NXT-G

Интерфейс NXT-G появляется, как только создана или открыта программа (Robo Center уменьшает обзор интерфейса, можно временно скрыть его). В результате получаем окно программы, состоящее из четырех основных прямоугольников: 1) рабочее поле программы, на котором расположен командный центр; 2) палитра команд; 3) окно настройки параметров команд; 4) окно просмотра общего вида программы (рис. 4.2). Кроме того, сверху находится главное меню и пикторраммы переключения интерфейса среды, в частности пиктограмма оранжевой трехмодульной балки, по которой можно вернуться в Robo Center. Стрелкой на рис. 4.2 показана пиктограмма перехода к полной палитре команд.

Перетаскивая блоки из палитры команд на рабочее поле, можно создавать программу. При этом близлежащие блоки автоматически связываются проводами, похожими на гладкие балки серии Technic, создавая последовательность выполнения команд. Этими же балками, придерживая клавишу Shift, можно строить ответвления для параллельных задач.

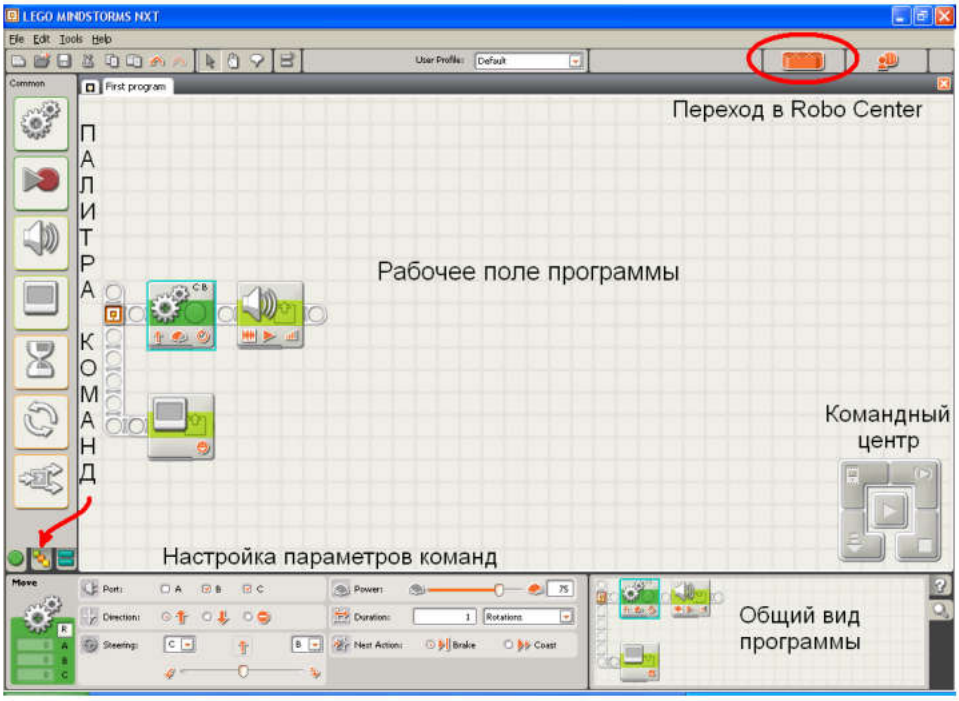


Рис. 4.2. Интерфейс NXT-G.

В окне настройки параметров команд указывается, к какому порту подсоединено устройство, а также в каком диапазоне значений и в ка-

ком режиме оно работает (для всех устройств по-разному). Внесенные параметры в виде миниатюрных пиктограмм отображаются на блоках команд в программе.

Рис. 4.3. Пиктограмма блока управления моторами.

Рассмотрим пиктограмму блока управления моторами «Move block» (рис. 4.3), параметры которого приведены в окне настройки (рис. 4.2).

Буквы в правом верхнем углу блока (1) показывают, какие из портов устройства NXT будут контролироваться. Пиктограмма «стрелка» (2) дает направление движения робота. Пиктограмма «индикатор мощности» (3) показывает уровень мощности. Скорость робота может также зависеть от прочих условий, например поверхности, по которой он движется, а также движения в гору или под гору. Пиктограмма (4) определяет значение, установленое для характеристики «Продолжительность» вращения: без ограничений, градусы, обороты или секунды.

У большинства блоков имеются концентраторы данных, которые по умолчанию спрятаны. Для извлечения концентратора достаточно щелкнуть мышкой по кнопке в левой нижней части блока, помещенного в рабочую область (рис. 4.4).

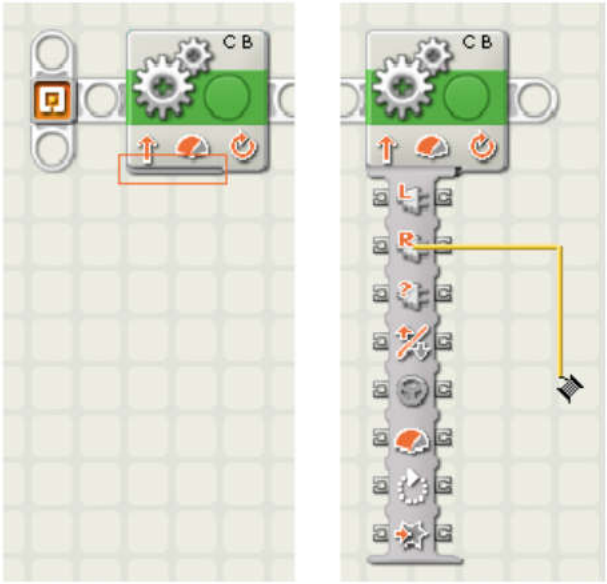


Рис. 4.4. Извлечение концентратора данных.

Как правило, шина данных создается между двумя концентраторами. На подключаемых разъемах один из них должен возвращать данные, а другой — принимать их. Если шину подключить неправильно, на несовместимые разъемы, то она будет выглядеть поврежденной, в виде пунктирной линии. Вот примеры подключении шин данных (рис. 4.5):
(A) — входной разъем,
(B) — выходной разъем,
(C) — числовая шина данных (желтая),
(D) — логическая шина данных (зеленая),
(E) — текстовая шина данных (оранжевая),
(F) — поврежденная шина данных (серая).

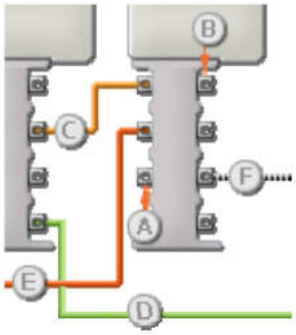


Рис. 4.5. Типы шин данных между концентраторами.

Ветвления

Есть еще несколько специфических блоков, которые стоит упомянуть. Первый из них «Блок принятия решений», или «Ветвление».

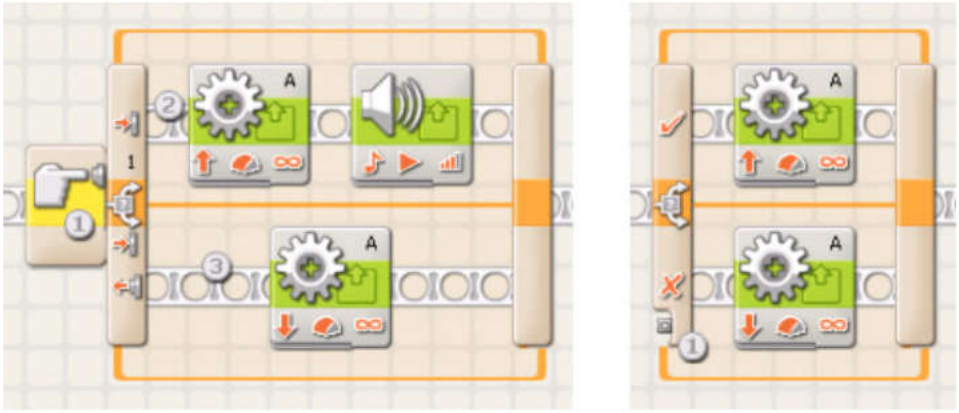


Рис. 4.6. Ветвления: по датчику нажатия (слева), по логическому или числовому значению (справа).

В приведенном примере (рис. 4.6, слева) в зависимости от состояния датчика нажатия (1) выполняется либо верхняя ветвь (2 — если нажат), либо нижняя (3 — если отпущен).

Любое ветвление с помощью окна настройки параметров можно перенастроить на другое условие. Например, можно принимать решение на основании некоторого значения (рис. 4.6, справа), подаваемого на специальный разъем. В этом случае значение должно поступить по логической или числовой шине данных от какого-то другого блока.

Ветвление может отображаться в двух режимах: 1) когда видны обе ветви (и «Да», и «Нет»), как на рис. 4.6; 2) с отображением только одной из выбранных ветвей, как на рис. 4.7. Это может быть полезно для экономии места на экране, которого в NXT-G катастрофически не хватает. Режим определяется установкой флажка «Flat view» в левой нижней части окна настройки параметров.

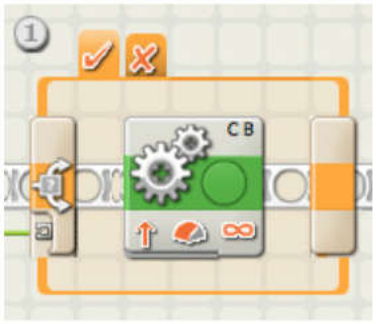


Рис. 4.7. Сокращенное отображение ветвления.

Циклы

Следующий блок, с которым стоит познакомиться, это цикл. Как правило, в NXT-G используются циклы либо с параметром, либо с постусловием.

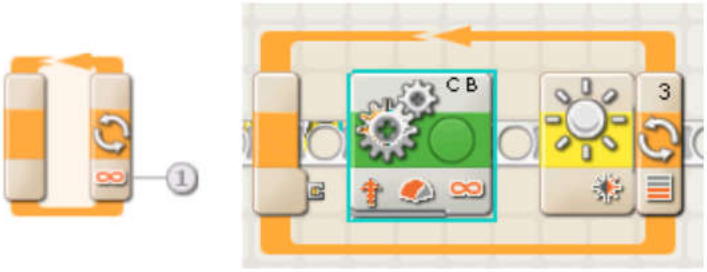


Рис. 4.8. Пиктограмма бесконечного цикла (слева), цикл по датчику освещенности (справа).

В качестве условия в правом нижнем углу пиктограммы (1) указывается режим повторений, в примере слева цикл работает бесконечно (рис. 4.8). Кроме того, циклы бывают с фиксированным числом повторений, по значению таймера, значению датчика, значению переменной.

В приведенном примере (рис. 4.8, справа) моторы B и C работают до тех пор, пока на датчике освещенности на 3-м порту не будет показано значение больше 50. При этом включен режим вывода значения счетчика повторений, и программист может подсоединить шину данных к разъему в левом нижнем углу пиктограммы цикла.

Переменные

Для использования переменных в NXT-G предусмотрен блок Variable, находящийся в палитре Data (рис. 4.9).

К единственному разъему (1) у него можно подключить шину данных, по которой будет передаваться значение (рис. 4.10).

Направление передачи на чтение или запись (2) и значение переменной (3) настраиваются в параметрах блока. По умолчанию присутствуют всего три переменных трех типов (1): логика, число и текст.

Программист может создать свою переменную, воспользовавшись пунктом главного меню Edit → Define Variables.

Рис. 4.9. Пиктограмма переменной.

Рис. 4.10. Настройка типа переменной.

Для того чтобы самостоятельно начать программировать, этого уже достаточно. Если же читатель заинтересуется глубже, он может заглянуть в полную палитру, добавить дополнительные блоки, создать свои блоки и многое другое. Среда языка NXT-G обладает массой интересных возможностей и представляется весьма полезным инструментом для изучения начинающими программирования NXT-роботов. Если же еще не появилась уверенность в своих силах, милости просим в Robo Center!

Robo Center

Раздел Robo Center содержит четыре модели, разработанных компанией Lego специально для пользователей NXT. Перейдя к любой из них, можно получить исчерпывающее руководство по сборке и составлению простейших программ для этих моделей. Эти разделы стоит пройти любому счастливому обладателю конструктора, поскольку в них не только можно найти неплохой самоучитель, но понять некоторые принципы, заложенные создателями Lego Mindstorms NXT в свое детище.

Собирая модель по инструкции, не забудьте раскрыть окно Robo Center на весь экран, чтобы лучше видеть мелкие детали и соединения.

Обладатели конструктора 9797 и программного обеспечения серии Education, к сожалению, не имеют возможности собрать эти модели по инструкции, поскольку ее там нет. Но зато прилагается несколько десятков неплохих уроков со стандартной учебной моделью, которые тоже можно пройти самостоятельно.

Сконструировав модели и разобравшись в принципе их функционирования, программы можно составить и в любой другой из сред, описанных в следующих главах.

TriBot

TriBot — это трехколесный движущийся робот (рис. 4.11). Он использует преимущества всех четырех датчиков, чтобы сделать то, на что он запрограммирован. TriBot может поднимать мяч, когда получает голосовую команду. Также он может быть запрограммирован на движение по линии. Он может ощущать объекты и имеет систему зрения.

Рис. 4.11. TriBot.

Датчики и элементы, которые использует TriBot:

— контроллер NXT,
— ультразвуковой датчик,
— датчик звука,
— датчик прикосновения,

— датчик света,
— три сервомотора.

Рассмотрим пример программы (рис. 4.12), которая просто и быстро заставит TriBot двигаться по черной линии. Пусть левый мотор подсоединен к порту C, правый к порту B, а датчик освещенности — на 3-й порт.

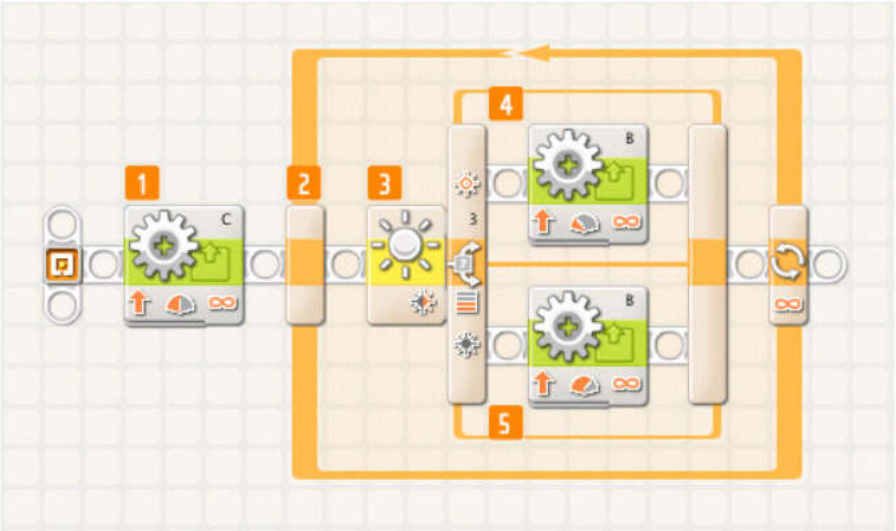


Рис. 4.12. Программа для Tribot.

В этой программе сначала (1) дается команда мотору C двигаться без ограничения вперед с мощностью 50 %. Мгновенно запускается цикл (2), в котором постоянно проверяются показания датчика освещенности (3) и корректируется мощность мотора B (4 и 5) с помощью ветвления. Если освещенность под датчиком выше 50 % (белый цвет), то выполняется верхняя ветвь (4): мотор B сбавляет мощность до 25 %. Если освещенность под датчиком ниже 50 % (черная линия), то выполняется верхняя ветвь (4): мотор B набирает мощность до 75 %. Таким образом, либо робот движется направо в поисках черной линии, либо налево, стараясь съехать с нее на белое поле. Очевидно, что для эффективного выполнения алгоритма робот должен стартовать, когда его датчик находится по левую сторону от линии, практически на границе с белым. Подобные устройства, называемые релейными регуляторами, подробно рассмотрены в главе 7, посвященной алгоритмам управления.

RoboArm

RoboArm T-56 (рис.4.13) — это сложная роботоподобная рука, которая может поднимать, поворачивать и захватывать объекты, исполь-

зуя для этого свои когти. Он может различать цвета и ощущать объекты. Для того чтобы робот двигался, используются три мотора — один управляет когтями захвата, а два других нужны для поворотов и движения вверх и вниз.

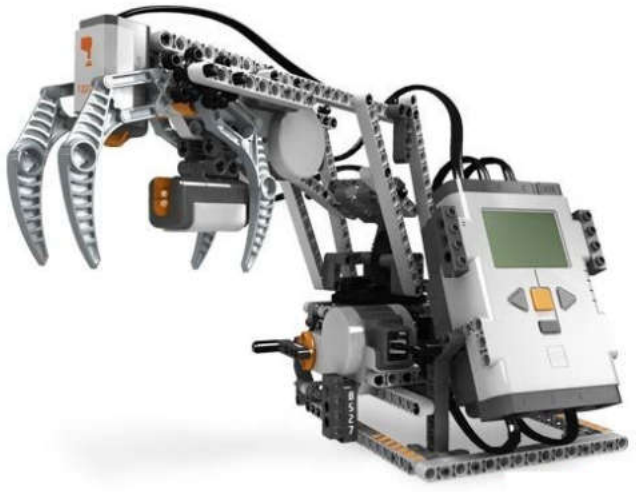


Рис. 4.13. RoboArm.

Датчики и элементы, которые использует RoboArm T-56:

— контроллер NXT,
— датчик прикосновения,
— датчик света,
— три сервомотора.

Пример программы, которая заставит робота распознавать и захватывать красный или синий шарик, дан на рис. 4.14.

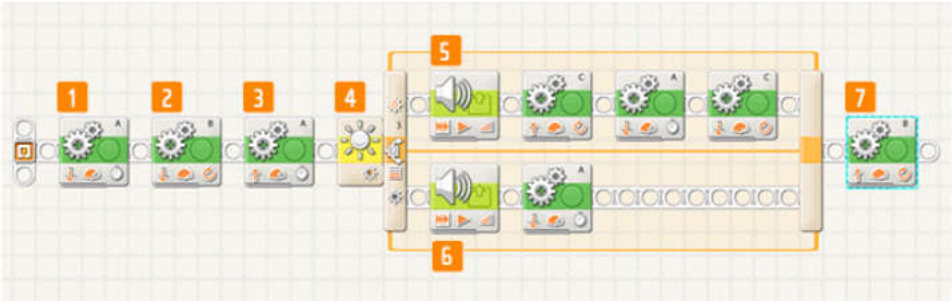


Рис. 4.14. Программа для RoboArm.

В этой программе мотор A (1) открывает когти, мотор B (2) опускает руку на шарик, после чего мотор A (3) захватывает шарик когтями. Далее производится проверка цвета шарика (4): для датчика освещен-

ности синий темнее красного. Если шарик оказался красным, выполняется верхняя ветвь алгоритма (5): NXT издает соответствующий красному сигнал, мотор C перемещает руку, мотор A раскрывает когти, роняя шарик, после чего мотор C возвращает руку. При синем шарике (6) издается сигнал «синего» и шарик сразу выбрасывается мотором A как негодный. После ветвления мотор B (7) возвращает руку в исходное положение.

Spike

Робот Spike (рис. 4.15) реагирует на все, как настоящий скорпион, крадется на шести ногах и имеет крепкие клешни. Он также может видеть и слышать, используя ультразвуковой датчик и датчик звука. Кроме этого, Spike умеет быстро и точно «парализовать» жертву, касаясь ее датчиком прикосновения.

Рис. 4.15. Spike.

Датчики и элементы, которые использует Spike:

— контроллер NXT,
— ультразвуковой датчик,

— датчик звука,
— датчик прикосновения,
— три сервомотора.

Рассмотрим программу, которая просто и быстро заставит Spike идти к цели, поразить ее хвостом, играя при этом музыку, и вернуться на место (рис. 4.16).

Рис. 4.16. Программа для Spike.

В этой программе скорпион движется вперед на 10 оборотов моторов C и B (1). Затем с помощью мотора A (2) он наносит удар хвостом до тех пор, пока не коснется жертвы «жалом», т. е. пока не сработает датчик касания на кончике хвоста (3). После успешного «укуса» NXT играет музыку (4), хвост возвращается назад (5) и скорпион пятится на исходную позицию (6). Если же кнопка не будет нажата во время удара, то робот «застрянет» в ожидании срабатывания датчика, а его создателю придется пожертвовать пальцем, чтобы продолжить выполнение программы.

Alpha Rex

Alpha Rex (рис. 4.17) умеет делать то, что под силу только самым развитым роботам-андроидам. Он ходит на двух ногах, почти как настоящий человек. В его ноги встроена пара сервомоторов, которые позволяют ему двигаться, а ультразвуковой датчик дает роботу возможность видеть препятствия. На руках у Alpha Rex находятся датчики звука и прикосновения.

Датчики и элементы, которые использует Alpha Rex:
— контроллер NXT,
— ультразвуковой датчик,
— датчик звука,
— датчик прикосновения,
— два сервомотора.

Рис. 4.17. Alpha Rex.

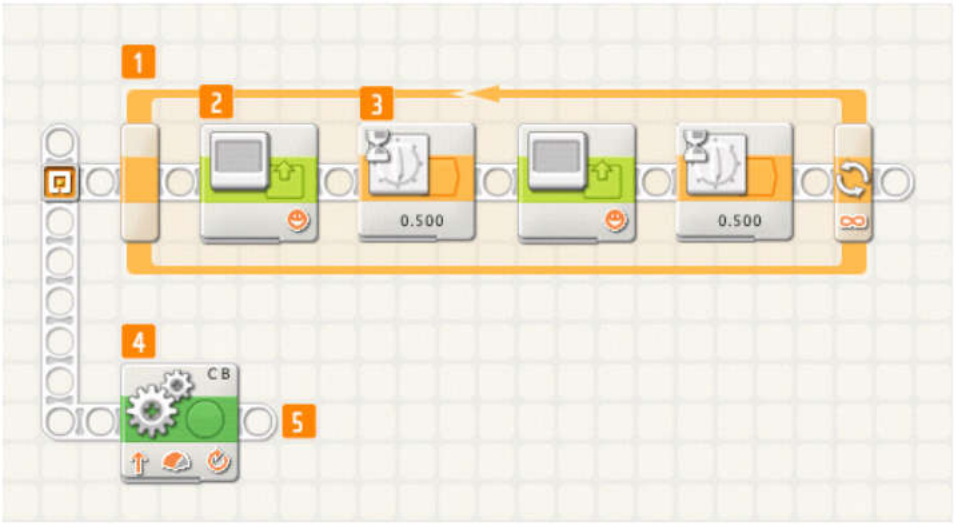


Рис. 4.18. Заготовка программы для Alpha Rex.

Пример стандартной программы, которая просто и быстро заставит Alfa Rex ходить, а его сердце — биться, дан на рис. 4.18.

В данной программе одновременно запускаются две параллельные задачи. В первой из них содержится бесконечный цикл (1), в котором на дисплее NXT поочередно отображаются две различные картинки бью-

щегося сердца (2) с промежутком 0.5 секунды (3). Во второй задаче, ветвь которой отходит вниз и вправо, осуществляется движение вперед с помощью заданного числа оборотов моторов C и B (4). Можно продолжить действия, добавив свои собственные блоки (5).

Более подробную информацию о среде NXT-G можно получить в руководстве, встроенном в программное обеспечение Lego Mindstorms NXT (раздел Robo Center) и на сайте Lego:
http://mindstorms.lego.com/en-us/Software/Default.aspx.

Глава 5. Программирование в Robolab

Введение

Robolab 2.9 — это многофункциональная графическая среда программирования, созданная на основе LabView 7.0 и ориентированная на самые разные возрасты — от дошкольников до студентов. Текущая версия Robolab позволяет программировать несколько типов микроконтроллеров — Control Lab, RCX, NXT, также проводить независимые расчеты на компьютере. Следует заметить для тех, кто работал с RCX, что новая прошивка (Firmware) для него работает в 100 раз быстрее, правда загружается также долго (4—5 мин). Загрузка прошивки в NXT длится не более минуты. Следует обратить внимание на дань прошлому со стороны разработчиков: некоторые пункты меню содержат название RCX, но подразумевают также и работу с NXT.

При запуске Robolab предлагает три уровня работы: «Администратор», «Программист» и «Исследователь» (рис. 5.1).

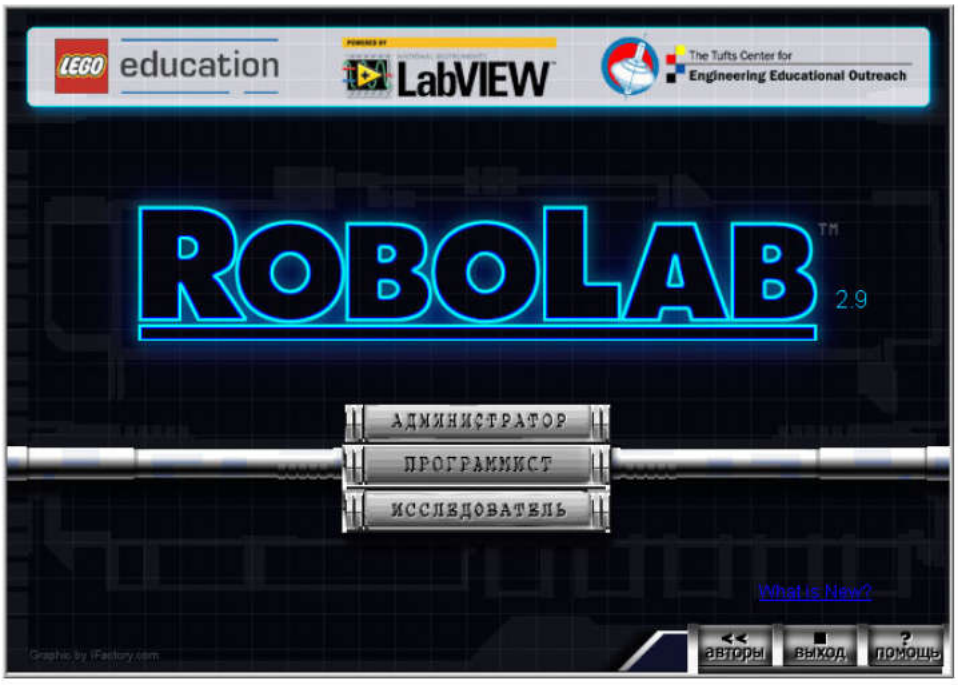


Рис. 5.1. Окно запуска Robolab 2.9.

Режим администратора позволяет настраивать контроллер на работу со средой.

Режим программиста позволяет непосредственно создавать программы и загружать их в микроконтроллер.

Режим исследователя позволяет осуществлять запись данных, поступающих с датчиков микроконтроллера, с их последующим анализом. Рассмотрим два предыдущих режима подробнее.

Режим «Администратор»

Первое, что следует сделать администратору — выполнить «Проверку связи с RCX» (подразумевается NXT). Есть три варианта исхода: 1) успешное завершение со звуковым сигналом на микроконтроллере; 2) сообщение о необходимости сменить операционную систему (ОС NXT или, иначе говоря, Firmware); 3) сообщение об ошибке связи.

Рис. 5.2. Соединение NXT с компьютером через USB.

Если связь компьютера и NXT не появилась сразу, следует воспользоваться меню «Select COM Port» для выбора USB-порта, выбрать автоопределение, выключить NXT, убедиться, что он хорошо соединен, и снова включить (рис. 5.2).

Рис. 5.3. Окно выбора устройства, с которым будет установлена связь.

Для поддержки новых возможностей NXT дополнительные опции добавлены в таблицу «Установка RCX/NXT». Здесь можно выбрать имя NXT, имя файла загружаемой в NXT программы. По умолчанию используется имя rbl. Допукается использовать до шести символов или можно задать его из программы с использованием расширенного NXT-светофора (блок NXT begin).

Поскольку Robolab поддерживает несколько устройств (RCX, NXT, Control Lab), диалоговое окно «Choose Hardware» добавлено в установки автоопределения. Оно может появляться в разных ситуациях при потере связи с NXT, и этого не надо бояться (рис. 5.3). Надеюсь, выбор будет очевиден.

Режим «Программист»

Раздел программиста делится на два: Pilot и Inventor, что не совсем точно переведено как «Управление» и «Конструирование» (рис. 5.4).

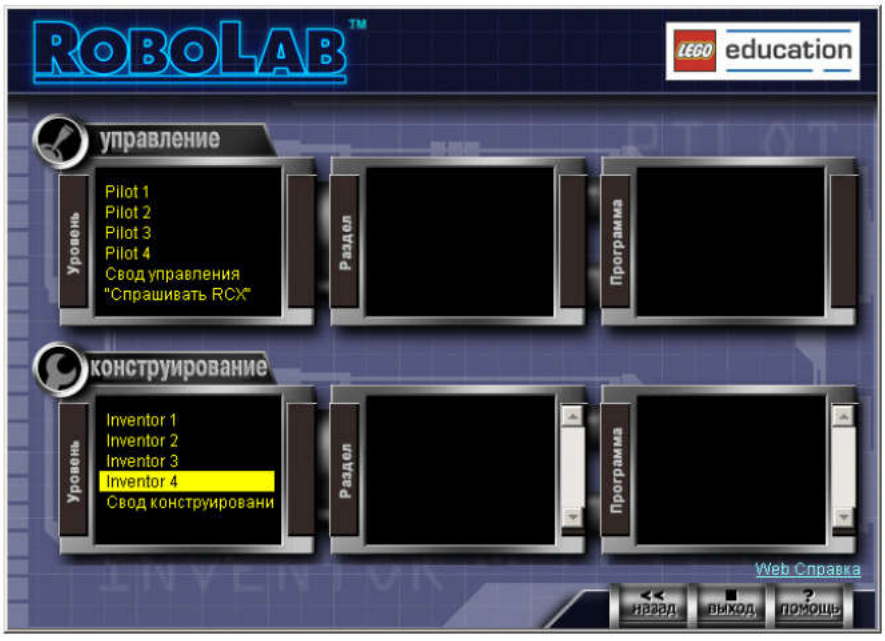


Рис. 5.4. Выбор режима программирования: двойной щелчок по Inventor 4.

Следовало бы назвать эти разделы «Новичок» и «Изобретатель». Можно было бы пройти их последовательно, постепенно окунаясь в среду графического программирования. Однако нет препятствий для того, чтобы начать сразу с последнего уровня Inventor 4, в котором

представлены все основные возможности программирования среды Robolab. Для перехода на этот уровень необходимо дважды щелкнуть по надписи Inventor 4, не касаясь подразделов.

Основные окна

Итак, мы зашли в среду программирования (рис. 5.5). На экране два основных окна, относящихся к одному проекту: Front Panel и Block Diagram. Первое (передняя панель) для программирования не пригодится, хотя его можно использовать в режиме исследователя. Второе, в котором уже расположены две пиктограммы светофоров (рабочее поле программы), предназначено для составления программы. Его стоит распахнуть на весь экран и приступить к работе.

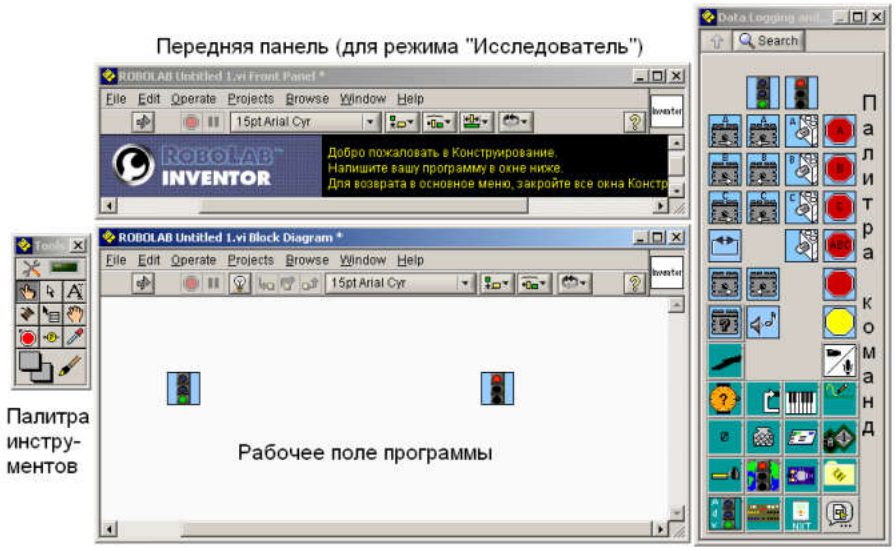


Рис. 5.5. Окна среды программирования Robolab.

Два вспомогательных окна — Tools Palette и Functions Palette — содержат все необходимое для составления программы. В случае закрытия их снова можно вывести на экран через пункт Windows верхнего меню.

Программа в Robolab похожа на блок-схему, положенную на левый бок. Она читается слева направо, хотя блоки располагать можно как угодно. Блоки команд находятся в окне Functions Palette (палитра команд). Они связываются между собой проводами (рис. 5.6), а также управляются инструментами, находящимися в меню Tools Palette (палитра инструментов).

Рис. 5.6. Соединение блоков в окне программы.

Название любого блока в Functions Palette можно прочитать в верхней части окна, подведя к нему курсор. Кроме того, под заголовком окна находится кнопка Search (поиск), с помощью которой можно найти пиктограмму по названию.

Некоторые из пиктограмм сами являются палитрами, и при переходе открывается новое окно (рис. 5.7). Вернуться в предыдущее можно по стрелочке, расположенной в левом верхнем углу палитры рядом с кнопкой Search.

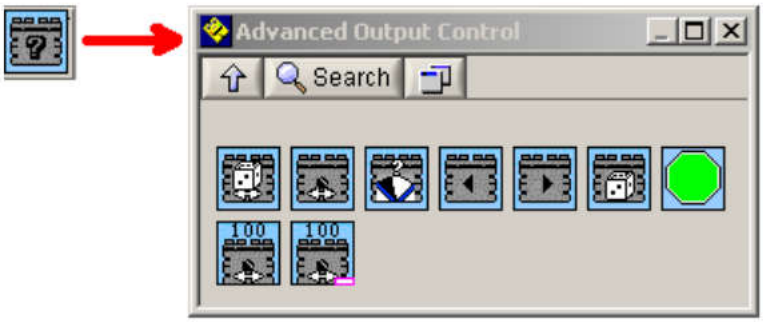


Рис. 5.7. Дополнительная палитра команд.

Готовые примеры программ

Одним из новшеств Robolab 2.9 является палитра примеров Behaviors (рис. 5.8). Это блоки, содержащие в себе готовые части программ под определенные задачи. Например, блок Go Straight (двигаться прямо) запускает моторы A и C вперед и через 1 секунду выключает их (рис. 5.9, слева). Однако для запуска этого фрагмента программы его надо обязательно поместить между светофорами, соединив их розовыми проводами (рис. 5.9, справа).

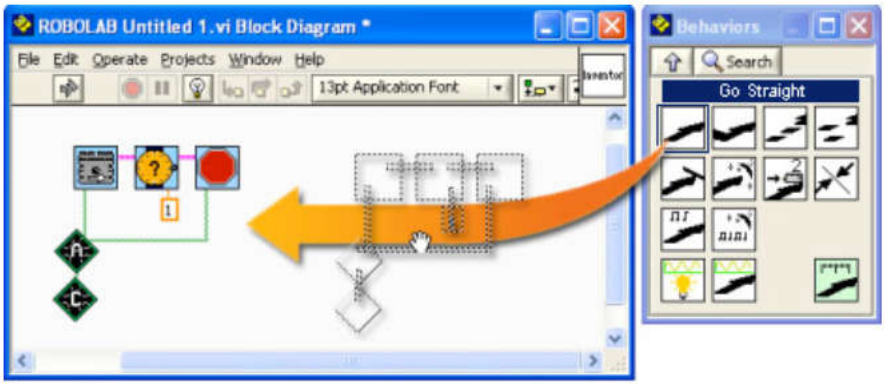


Рис. 5.8. Перетаскивание примера в поле программы.

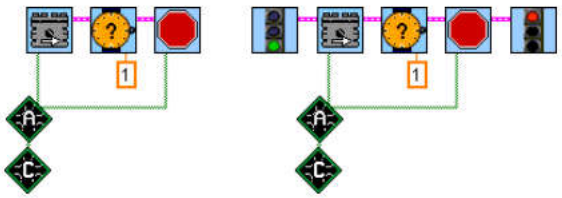


Рис. 5.9. Содержание примера и размещение его между светофорами.

Палитра Behaviors содержит ряд серьезных примеров программирования в Robolab, по которым можно самостоятельно многому научиться. Некоторые из таких примеров рассмотрены ниже.

Взаимодействие с NXT

Рис. 5.10. Загрузка ОС NXT через меню Project → Detective.

В режиме программиста могут потребоваться настройка USB-подключения NXT (соединение с компьютером через Bluetooth не поддерживается) и загрузка операционной системы. Это реализуется через меню Projects, которое для опытного робототехника содержит много интересного.

Рассмотрим некоторые возможности настройки.

Пункт меню Project → Select COM Port позволяет выбрать порт USB-подключения NXT с указанием имени устройства.

Пункт меню Project → Detective позволяет не только выбрать соответствующий порт подключения NXT, но и загрузить Firmware, если это не было сделано в режиме администратора (рис. 5.10). Во время за-

грузки операционной системы у NXT пропадает изображение на экране, но в течение минуты категорически нельзя предпринимать никаких действий над контроллером[1].

Когда операционная система загружена, стандартная программа, попавшая в память NXT, размещается в разделе My Files → Software Files и получает имя rbl. Для того чтобы дать другое имя, следует в программе Robolab заменить начальный блок — зеленый светофор — на аналогичный с надписью NXT из палитры NXT. На таком блоке в специальном розовом прямоугольнике можно задать латинскими буквами и цифрами любое имя программы, причем значимыми будут не более шести первых символов (рис. 5.11).

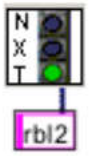


Рис. 5.11. Уникальное имя программы.

Не забывайте сохранять свои программы на диске компьютера, давая им запоминающиеся имена. Когда в другой раз попытаетесь открыть сохраненную программу и увидите черное окно, не пугайтесь: чтобы перейти к алгоритму, необходимо выполнить пункт меню Window → Show Block Diagram.

Типы команд

Блоки Functions Palette можно классифицировать следующим образом:

— команды действия,
— команды ожидания,
— управляющие структуры,
— модификаторы.

Начнем с простейших команд. Их можно разделить на два типа: 1) «Жди» и 2) «Делай».

Команды типа «Делай» посылают управляющий сигнал на одно из устройств управления микроконтроллера, например, «включить моторы», «остановить моторы», «издать звуковой сигнал», «отправить сообщение», «обнулить таймер» и т.п. Это действие, как правило, выполняется практически мгновенно (за исключением звуковых сигналов), после чего программа переходит к следующему блоку. Заметим, что

[1] NXT, вышедший из строя при несанкционированном отключении во время загрузки операционной системы, можно восстановить из другой системы. Например, временно загрузить Firmware из RobotC.

включенный мотор продолжает работать до тех пор, пока не выполнится команда выключения или программа не закончится.

Команды типа «Жди» ведут себя иначе. Они не выполняют никакого ощутимого действия, хотя и активно взаимодействуют с оборудованием NXT. Эти команды останавливают ход выполнения программы (точнее, задачи) в ожидании некоторого события. Как только событие происходит, управление переходит к следующей команде. Примеры таких команд: «жди громкого звука», «жди яркого света», «жди заданное время» и т. д. Во время выполнения команды «Жди» все запущенные ранее процессы (включенные моторы и др.) продолжают выполняться.

Следующий тип блоков — это управляющие структуры. Среди них представлен основной «джентльменский набор» программиста:

— ветвления,
— циклы и безусловные переходы,
— подпрограммы,
— параллельные задачи,
— обработчики событий.

И, наконец, вспомогательный тип блоков — модификаторы. Они по сути являются парамерами для выполнения различных команд и прикрепляются к блокам команд разноцветными проводами.

Практически все блоки, оставшиеся в Robolab со времен RCX, пригодны для NXT. Но при этом появилась часть новых блоков, относящихся преимущественно к датчикам NXT, а также расширяющих функциональные возможности Robolab для обеих систем. Заметим, что между датчиками, моторами и микроконтроллерами сохранена совместимость «снизу вверх», так что практически во всех случаях можно использовать старые датчики и моторы с новым микроконтроллером. Только для этого нужны соответствующие переходники, которые входят в состав набора 9797, а также продаются отдельно.

Команды действия

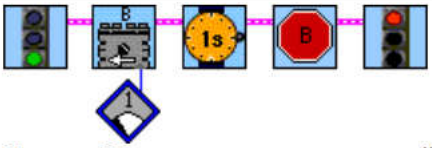
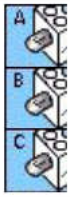
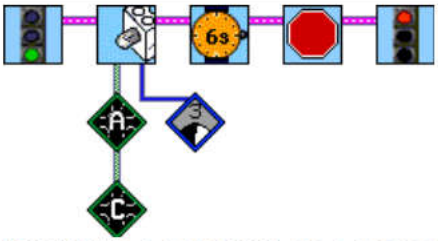
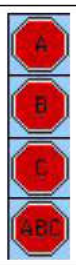
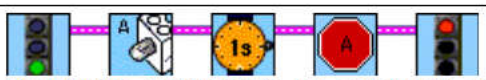
Команды действия применимы к управляющим устройствам NXT. К ним можно отнести:

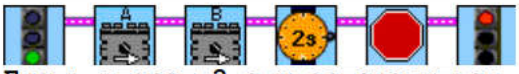
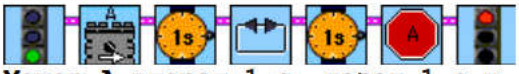
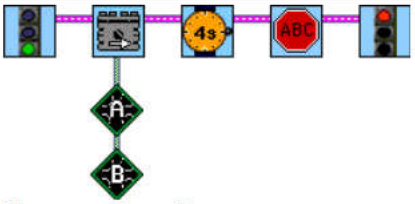
— моторы,
— динамик,
— таймеры,
— встроенная память (контейнеры),
— датчики при инициализации и др.

Базовые команды

Рассмотрим блоки действия, расположенные в главном окне палитры команд (табл. 5.1).

Таблица 5.1. Базовые команды действия

Блок	Назначение	Пример использования
	Мотор вперед на соответствующем порту (A, B или C) с заданной мощностью 1...5 (по умолчанию 5)	 Мотор A вперед в течение 1 с, затем остановиться
	Мотор назад на соответствующем порту (A, B или C) с заданной мощностью 1...5 (по умолчанию 5)	 Мотор B назад с минимальной мощностью в течение 1 с, затем остановиться
	Включить лампочку на соответствующем порту (A, B или C)	 Включить лампочку на порту A и через 1 с выключить
	Включить лампочку на указанном порту (по умолчанию все)	 Включить лампочки на портах A и C с мощностью 3, через 6 с выключить все
	Остановить мотор или выключить лампочку	 Включить лампочку на порту A и через 1 с выключить

Блок	Назначение	Пример использования
	Остановить мотор или выключить лампочку (по умолчанию — все)	 Ехать вперед 2 с и остановиться
	Снять напряжение с моторов (плавная остановка)	 Мотор A вперед в течение 1 с, затем плавно затормозить
	Сменить направление вращения (реверс)	 Мотор A вперед 1 с, назад 1 с и остановиться
	Моторы вперед или назад на указанном порту	 Моторы A и B вперед в течение 4 с, затем остановиться
	Издать звук (тип звука устанавливается модификатором)	 Издать звук по умолчанию и завершить программу

Команды управления моторами, приведенные в табл. 5.1 имеет смысл использовать в простейших экспериментах, в которых не требуется точный контроль мощности.

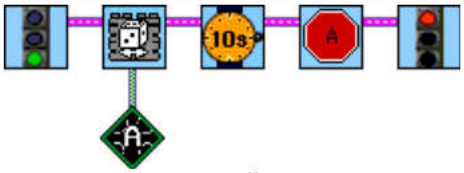
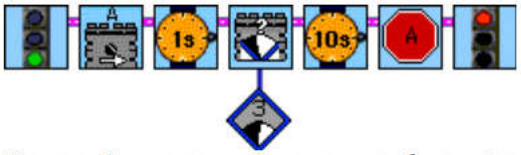
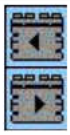
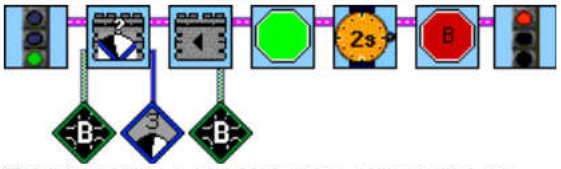
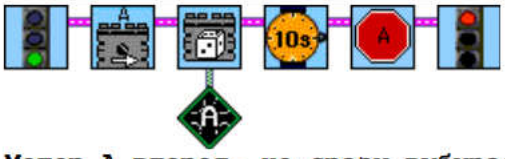
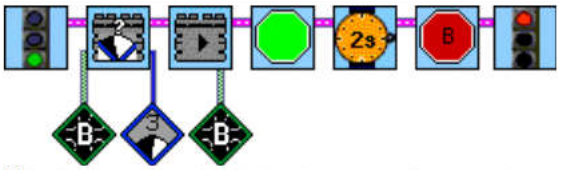
Для более совершенных результатов рекомендуется воспользоваться палитрой Advanced Output Control.

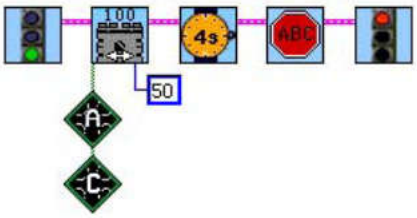
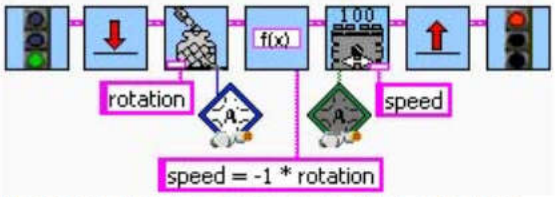
Продвинутое управление моторами

Блоки управления моторами с контролем мощности от –100 до 100 отличаются от стандартных режимом торможения широтно-импульсной модуляции (ШИМ), который имеет преимущество над плавающим режимом в точности движения. Подробнее об этом читайте в главе 6 «Программирование в RobotC».

В остальном дополнительная палитра команд управления моторами позволяет разнообразить и оптимизировать действия с ними (табл. 5.2).

Таблица 5.2. Дополнительные команды действия с моторами

Блок	Назначение	Пример использования
	Включить мотор в случайном направлении	 Мотор A в случайном направлении работает 10 с и выключается
	Задать мотору мощность (не изменяя направления движения)	 Мотор A вперед в течение 1 с, затем задать ему мощность 3 и через 10 с выключить
	Задать мотору направление (не изменяя мощности)	 Установить параметры движения мотора B: мощность 3, направление назад. Включить его с этими параметрами и через 2 с выключить
	Задать мотору случайное направление (не изменяя мощности)	 Мотор A вперед, но сразу выбирается случайное направление и через 10 с мотор выключается
	Включить моторы и лампочки с последними заданными значениями мощности и направления	 Установить параметры движения мотора B: мощность 3, направление назад. Включить его с этими параметрами и через 2 с выключить

Блок	Назначение	Пример использования
	Задать мощность и направление мотора одним числом –7...7 (положительное — вперед, отрицательное — назад, ноль — стоп) и включить мотор.	 **Положить в контейнер по умолчанию (red) число -5, затем 11 раз повторить: {включить мотор A с мощностью red, через 1 с издать звук, добавить 1 в контейнер (red)}. Затем выключить мотор A.** **В процессе движения мотор сперва крутится назад, затем плавно останавливается и вращается вперед**
	Включить мотор с мощностью –100...100 (в режиме торможения)	 **Моторы A и C вперед с мощностью 50%, через 4 с остановиться**
	Включить мотор с мощностью –100...100 (в режиме торможения) с помощью строковой формулы	 **Бесконечно повторять: {В переменную rotation записать показания датчика оборотов мотора A, в переменную speed записать противоположное значение, на мотор A подать мощность, равную значению speed}.** **По сути эта программа удерживает мотор в фиксированном положении, не позволяя вращать его извне**

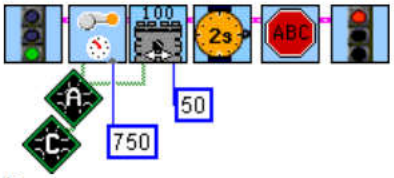
Последние две команды представляются наиболее интересными для использования в алгоритмах управления, о чем указано далее. Кроме того, они могут обращаться друг в друга автоматически при подключении соответствующего строкового (розового) или числового (синего) модификатора.

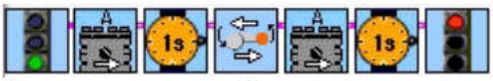
Моторы NXT

Кроме рассмотренной палитры интересные команды управления моторами содержатся в палитре NXT (табл. 5.3). С их помощью можно

— ехать с заданной скоростью,
— проезжать заданное расстояние и останавливаться,
— синхронизировать движение моторов,
— менять ориентацию моторов на противоположную.

Таблица 5.3. Моторы NXT

Блок	Назначение	Пример использования
	Синхронизация одного мотора с другим. Возможно изменение мощности (в процентах) и направления (знак)	 **Мотор C синхронизировать с мотором A в противоположном направлении, включить мотор A вперед (C назад), ждать 2 с, выключить синхронизацию, включить мотор C назад, ждать 2 с, выключить все моторы. Вращение моторов в разные стороны будет происходить в общей сложности 4 с**
	Контроль скорости моторов. Максимальная скорость при свежих аккумуляторах — 750°/с (на батарейках 1.5 В может достигнуть 1000)	 **Задать максимальную скорость моторов A и C на 750, включить их со скоростью 375 (т.е. 50%), ждать 2 с, выключить моторы**
	Контроль расстояния. Задает градусы, на которых после следующего запуска мотор должен остановиться	 **Задать точку останова моторов A и C на 750°, Включить моторы с мощностью 50, ждать 4 с. Моторы могут остановиться примерно через 2 с**

Блок	Назначение	Пример использования
	Изменить ориентацию мотора на противоположную	 **Включить мотор A вперед, ждать 1 с, сменить ориентацию мотора A, включить мотора A вперед (но уже в другую сторону), ждать 1 с**

Последняя команда хороша в том случае, когда на новой конструкции моторы оказались перевернуты. Чтобы не переписывать программу заново, достаточно изменить их ориентацию в самом начале.

Команды ожидания

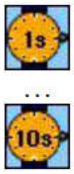
Команды ожидания можно классифицировать следующим образом:

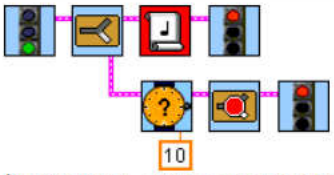
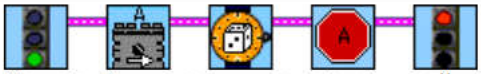
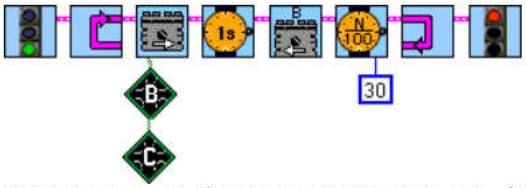
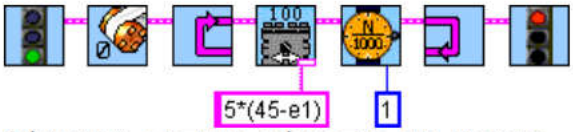
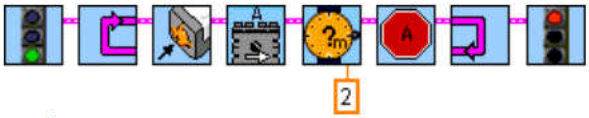
— ожидание интервала времени,
— ожидание показаний датчика,
— ожидание значения контейнера,
— ожидание показаний таймера и др.

Ожидание интервала времени

Эти команды следует разделить на фиксированные и переменные интервалы. Последние характеризуются возможностью подключения модификатора и наличием символов «?» или «N» на пиктограмме (табл. 5.4).

Таблица 5.4. Ожидание интервала времени

Блок	Назначение	Пример использования
	Ждать 1...10 секунд и перейти к следующей команде	 **Мотор A назад в течение 2 с, затем остановиться**

Блок	Назначение	Пример использования
	Ждать заданное число секунд (указать в модификаторе)	 **Запустив две параллельные задачи, в верхней проиграть мелодию, хранящуюся в красном свитке, а в нижней подождать 10 с и прервать выполнение всех задач, т.е. завершить программу, не дожидаясь окончания мелодии**
	Ждать случайное число секунд (до 5 по умолчанию)	 **Мотор A вперед, через случайное время от 0 до 5 с остановиться**
	Ждать N сотых долей секунды (указать в модификаторе)	 **Повторять в бесконечном цикле: {моторы B и C вперед на полную мощность, через 1 с развернуть мотор B в обратную сторону на 30 мс} Получается движение по квадрату или некоторой ломаной линии**
	Ждать N тысячных долей секунды (указать в модификаторе)	 **Обнулить датчик оборотов на моторе A (по умолчанию), затем в бесконечном цикле повторять: {на мотор A подать мощность, пропорциональную отклонению датчика оборотов от значения 45°, подождать 1 мс}.**
	Ждать заданное число минут	 **В бесконечном цикле повторять: {ждать нажатия датчика на 1-м порту, включить мотор A на 2 мин, затем выключить}**

Ожидание показаний датчика

Многим читателям так или иначе приходилось иметь дело со старой версией Mindstorms на базе контроллера RCX. Датчики из наборов этой серии компактны и, хотя уступают датчикам NXT, могут использоваться вместо них и вместе с ними.

Таблица 5.5. Сравнение блоков управления датчиками RCX и NXT

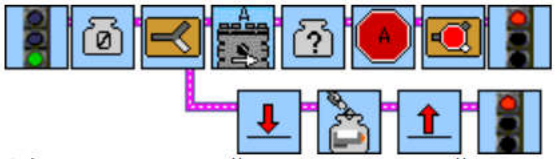
Блоки RCX	Блоки NXT	Назначение	Диапазон значений
		Жди нажато	$0...2^{31}-1$
		Жди отпущено	—
		Жди светлее, чем …	0...100
		Жди светлее на …	–100...100
		Жди темнее, чем …	0...100
		Жди темнее на …	–100...100
		Жди оборот на …	$-2^{31}...2^{31}-1$
—		Жди оборот на (с включением мотора) …	$-2^{31}...2^{31}-1$
		Жди оборот без обнуления	$-2^{31}...2^{31}-1$
—		Жди громче, чем …	0...100
—		Жди громче на …	–100...100
—		Жди тише, чем …	0...100
—		Жди тише на …	–100...100
—		Жди дальше, чем …	0...255
—		Жди ближе, чем …	0...255

Сравним основные команды ожидания, относящиеся к датчикам RCX и NXT (табл. 5.5). Диапазон значений в приведенной таблице в большинстве случаев указан из расчета реальных показаний датчиков. Между тем допустимой границей для всех целочисленных величин является диапазон $-2^{31}...2^{31}-1$, т. е. от –2147483648 до 2147483647.

Ожидание значения контейнера

Эту единственную команду следует выделить, поскольку она придает колоссальную мощь программе при использовании параллельных задач.

Таблица 5.6. Ожидание значения контейнера[1]

Блок	Назначение	Пример использования
	Жди значение контейнера	 **Обнулить контейнер, в верхней задаче включить мотор A, когда в контейнере появится 1 (по умолчанию), выключить мотор A. В нижней задаче в бесконечном цикле считывать в контейнер показания датчика нажатия.** **Таким образом, при нажатии на датчик в контейнер попадет логическая единица, которая и остановит мотор**

В данном примере (табл. 5.6) мотор будет крутиться, пока в контейнере (по умолчанию в красном) не появится значение (по умолчанию 1). Это произойдет в тот момент, когда в параллельной задаче, которая выполняется бесконечно, будет зафиксировано нажатие на кнопку датчика касания (и в контейнер будет положено значение 1).

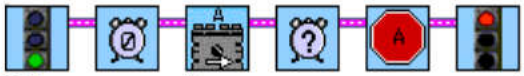
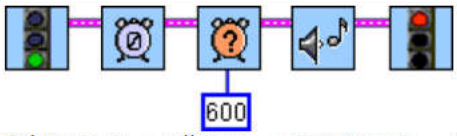
Ожидание значения таймера

Еще две команды, подобные предыдущей, могут быть использованы в системах защиты от сбоев, а также во многих других. Их можно объединить термином «ожидание значения таймера» (табл. 5.7).

[1] Если модификатор контейнера не указан, то по умолчанию он красный (red).

В системе допустимо использование четырех таймеров, пронумерованных от 1 до 4. Номер в качестве модификатора может быть прикреплен к блоку команды. По умолчанию используется первый таймер.

Таблица 5.7. Команды ожидания значения таймера

Блок	Назначение	Пример использования
	Жди значение таймера в 1/10 с (серая подсветка)	 Обнулить таймер, включить мотор A, ожидать значение таймера по умолчанию, выключить мотор.
	Жди значение таймера в 1/100 с (бежевая подсветка)	 600 Обнулить таймер, подождать, когда на таймере появится значение 6 с, издать звуковой сигнал

Управляющие структуры

Рассмотрим основные структуры, которые встретятся в этой книге. Все они доступны в палитре Structures (рис. 5.12). Структуры сами не производят действия, но определенным образом меняют их порядок выполнения в программе, что приводит к разнообразию возможностей поведения роботов.

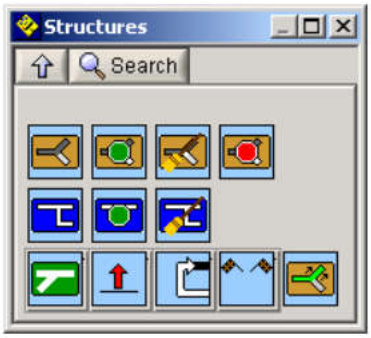


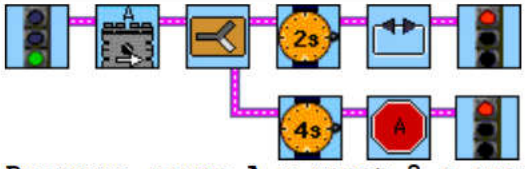
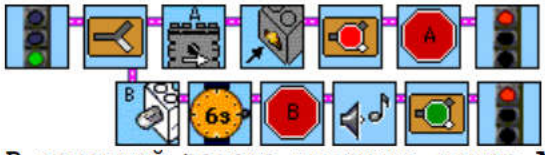
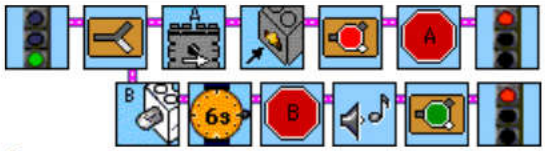
Рис. 5.12. Палитра управляющих структур Robolab.

Задачи и подпрограммы

Параллельные задачи

Один из основополагающих принципов управления роботом состоит в том, что программа разделена на задачи (Tasks), каждая из которых может выполняться одновременно с другими. Если разделения не видно, значит в программе всего одна задача, которая является главной. Первая строка пиктограмм палитры Structures содержит блоки управления параллельными задачами. Рассмотрим три из них (табл. 5.8).

Таблица 5.8. Управление параллельными задачами

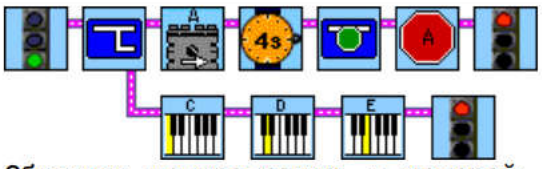
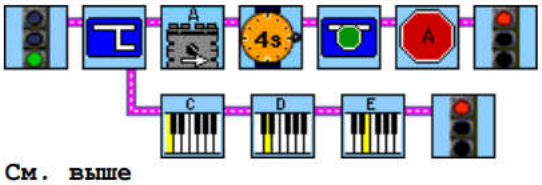
Блок	Назначение	Пример использования
	Начать новую параллельную задачу	 Включить мотор A и через 2 с сменить его направление из основной задачи, а через 4 секунды выключить из параллельной
	Перезапустить одну или все задачи, после того как они были выполнены или остановлены	 В основной задаче включить мотор A и ожидать нажатия датчика, после чего все отключить. В параллельной задаче включать лампочку, через 6 с выключать, издавать сигнал и повторять все заново
	Остановка одной или всех задач может означать окончание программы	 См. выше

Каждая задача должна иметь свой собственный конец — красный светофор. Всего в Robolab допустимо использовать до 10 параллельных задач.

Подпрограммы

Для сокращения размера программы за счет повторяемых кусков кода их можно заключать в подпрограммы (табл. 5.9). Всего допустимо использование восьми подпрограмм с номерами от 0 до 7. Как и параллельные задачи, подпрограммы должны иметь свои светофорные блоки конца.

Таблица 5.9. Создание и вызов подпрограмм

Блок	Назначение	Пример использования
	Создание новой подпрограммы. По умолчанию номер 0	 **Объявить подпрограмму, в которой проигрываются три ноты, и вызвать ее из основной программы через 4 с после включения мотора A. По окончании мелодии мотор A будет отключен**
	Вызов подпрограммы по ее номеру	 **См. выше**

В Robolab присутствует ограничение, по которому подпрограммы можно вызывать одну из другой только после того, как вызываемая подпрограмма была объявлена. В качестве номера подпрограммы можно использовать переменную, это может значительно упростить процедуру выбора варианта действия.

Ветвления

Среди ветвлений следует выделить несколько основополагающих. Остальные устроены по аналогичному принципу.

Во всех ветвлениях Robolab значение одной величины сравнивается со значением другой. На пиктограмме указывается, какое условие соответствует верхней ветви, а какое — нижней. Каждое ветвление строго должно сходится в одну развилку.

Условия можно разделить на три вида: 1) по состоянию, 2) по неравенству, 3) по равенству.

Условие состояния может относиться, например, к датчику касания: нажат/отпущен (рис. 5.13).

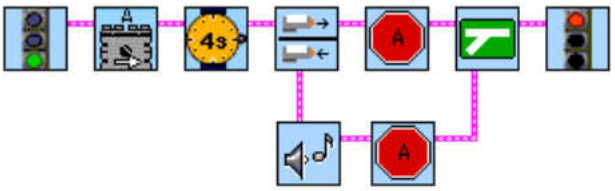


Рис. 5.13. Пример ветвления по состоянию датчика касания.

В условиях по неравенству рассматриваются два варианта: проверяемое значение больше/меньше или равно (рис. 5.14).

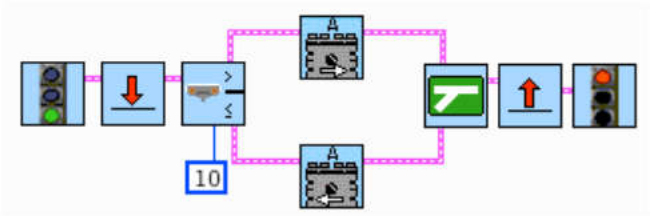


Рис. 5.14. Пример ветвления по показаниям датчика расстояния.

Условия по равенству допускают два варианта: равно/не равно (рис. 5.15).

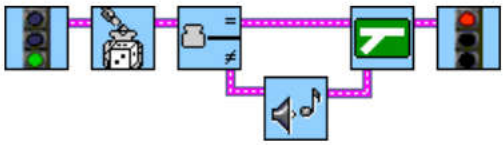
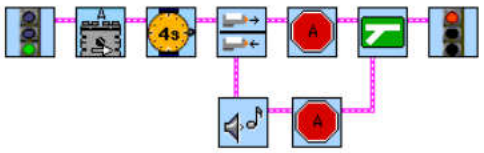
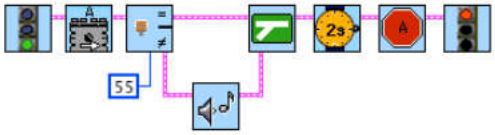
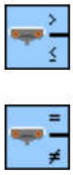
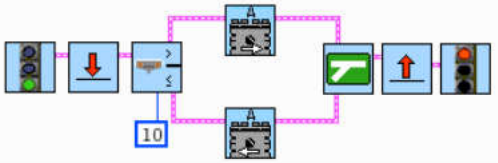
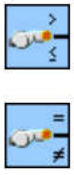
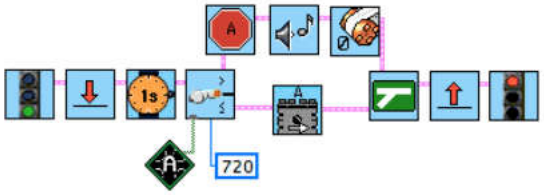


Рис. 5.15. Ветвление по значению контейнера.

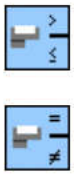
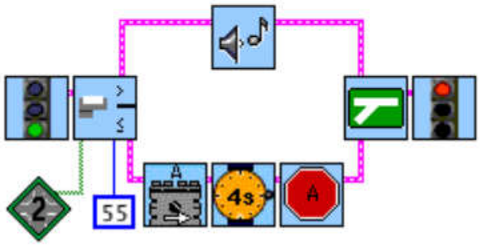
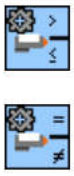
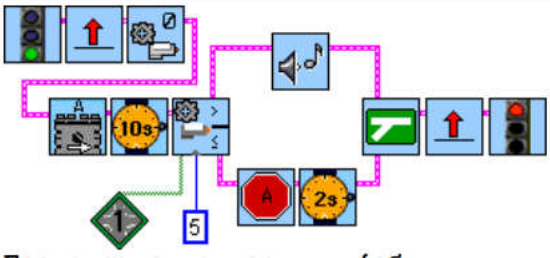
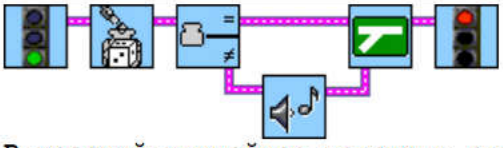
Следуя этим принципам, достаточно рассмотреть только некоторые из ветвлений (табл. 5.10). Остальные используются аналогично.

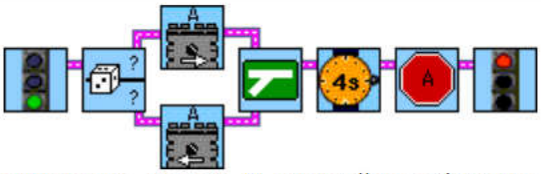
Проверка условий на равенство имеет смысл только в редких случаях, когда велика вероятность появления ожидаемого значения в момент проверки. Это относится в первую очередь к контейнерам, а также, например, к крайним значениям показаний датчиков. Для этих целей существует дополнительная палитра Equal Forks, пиктограмма которой находится в правом нижнем углу основной палитры ветвлений.

Таблица 5.10. Ветвления[1]

Блок	Назначение	Пример использования
	Датчик касания: отпущен или нажат	 **Включить мотор A и ждать 4 с. В зависимости от состояния датчика нажатия либо просто отключить мотор A, либо перед этим издать сигнал**
	Датчик звука: проверка уровня громкости	 **Включить мотор A и при уровне громкости, не равном 55%, издать звуковой сигнал. Через 2 с выключить мотор**
	Ультразвуковой датчик: проверка расстояния до объекта	 **Бесконечно повторять: {если расстояние до объекта больше 10 см, то включить мотор A вперед, иначе включить мотор A назад}**
	Датчик оборотов мотора: проверка текущего угла поворота	 **Бесконечно повторять: {ждать 1 с, если обороты мотора A превышают 720°, то {остановить мотор, издать сигнал, обнулить датчик оборотов}, иначе включить мотор A вперед}**

[1] Если не указан порт подключения датчика, то по умолчанию используется первый.

Блок	Назначение	Пример использования
	Датчик света: проверка уровня освещенности	 **Если показания датчика освещенности превышают 55%, издать сигнал, иначе {включить мотор A, ждать 4 с, выключить мотор}**
	Датчик касания: проверка числа нажатий	 **Бесконечно повторять: {обнулить счетчик нажатий на дачике касания, включить мотор A вперед, ждать 10 с, если датчик нажат более 5 раз, то издать звуковой сигнал, иначе {Выключить мотор A, ждать 2 с}}**
	Проверка показаний таймера	 **Обнулить таймер, включить лампочку, подождать случайное время от 0 до 5 с, если прошло меньше 3 с, издать сигнал, затем выключить мотор**
	Проверка значения контейнера	 **В красный контейнер положить случайное число от 0 до 8, если оно на равно 1, издать сигнал**

Блок	Назначение	Пример использования
	Случайный выбор	 Включить мотор A случайнм образом вперед или назад, ждать 4 с, выключить мотор

Прыжки

Эта команда схожа с оператором безусловного перехода goto, отвергаемым многими специалистами, как нарушающим структуру программы. Тем не менее в графическом программировании она может сыграть немаловажную роль. В каждом безусловном переходе присутствуют два блока: 1) собственно прыжок (переход) и 2) метка (область перехода), куда прыжок осуществляется (рис. 5.16).

Рис. 5.16. Метка.

Прыжок и его метка должны быть одного цвета, не могут повторяться и должны быть расположены в одной задаче (рис. 5.17).

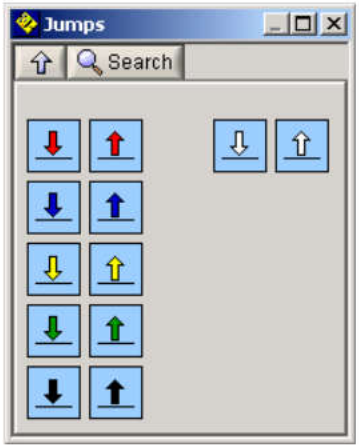


Рис. 5.17. Различные цветные пары меток и прыжков.

Таким образом, максимально возможное число цветных прыжков в программе — 5 (черный, красный, синий, зеленый, желтый) с номерами по умолчанию 0...4, а остальные имеет смысл нумеровать на основе «белого» прыжка, начиная с номера 5.

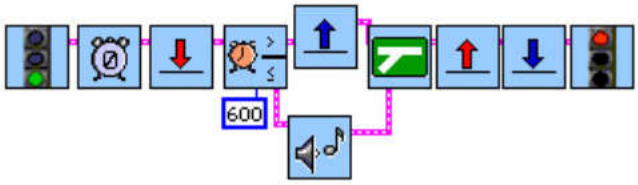


Рис. 5.18. Пример использования безусловных переходов.

С помощью прыжков можно создавать бесконечные циклы (рис. 5.18). Но для этой цели лучше воспользоваться классическими структурами.

Циклы

Циклы, как известно, можно разделить на три классических вида: 1) с предусловием, 2) с постусловием и 3) с параметром. В Robolab несколько иная классификация. Из трех перечисленных выше в чистом виде присутствует только цикл с предусловием (рис. 5.19).

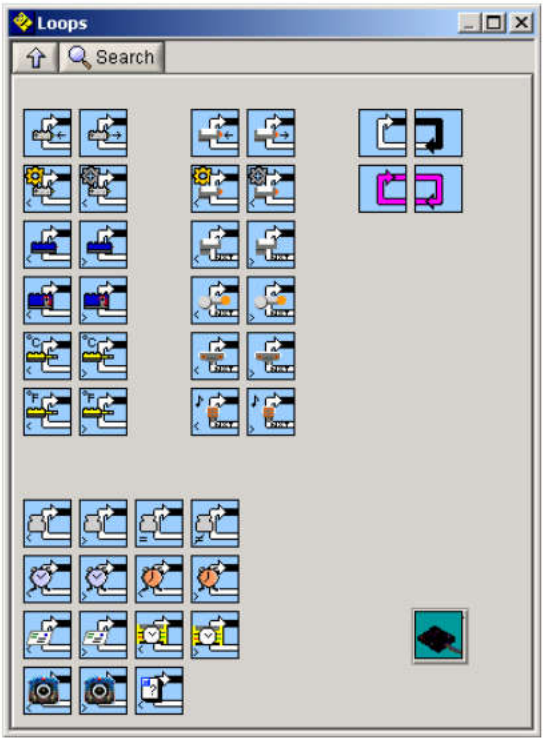


Рис. 5.19. Палитра «Циклы».

В целом же можно выделить следующие виды:

— бесконечный цикл;

— цикл с заданным числом повторений;

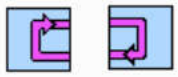
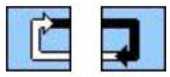
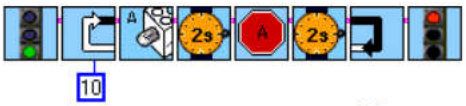
— цикл с предусловием.

В некоторых циклах возможно использование значения встроенного счетчика повторений.

Циклы без явных условий

У каждого цикла обязательно указываются блок начала и блок конца. Все команды, находящиеся между ними, составляют тело цикла (табл. 5.11).

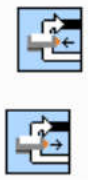
Таблица 5.11. Циклы без явных условий

Блок	Назначение	Пример использования
	Цикл с бесконечным числом повторений	 Бесконечно повторять: {включить лампочку A, ждать 2 с, выключить, ждать 2 с}. Короче говоря, мигалка.
	Цикл с заданным числом повторений	 То же самое повторить 10 раз

Циклы с предусловием

Условия в циклах весьма схожи с условиями ветвлений и задаются в блоке начала цикла. Блок конца цикла соответствует стандартному блоку конца цикла с заданным числом повторений (табл. 5.12).

Таблица 5.12. Циклы с предусловием по значению датчика

Блок	Назначение	Пример использования
	Цикл пока датчик касания нажат (отпущен)	 Пока не нажат датчик касания, повторять: {сыграть ноту, ждать паузу}, после нажатия издать сигнал. Если нажатие произойдет только во время звучания ноты или паузы, оно не будет зафиксировано

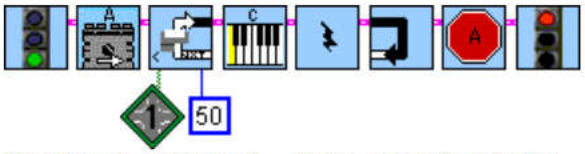
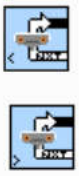
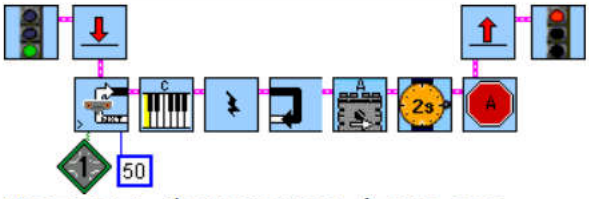
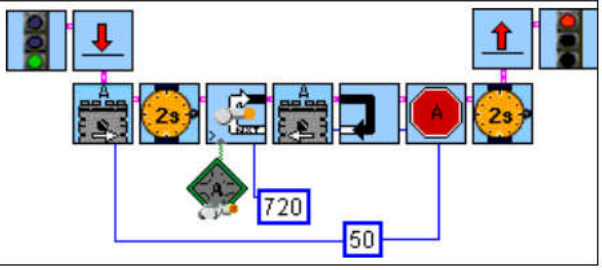
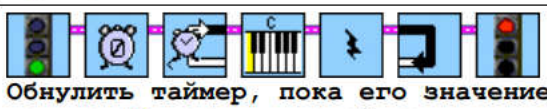
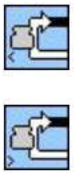
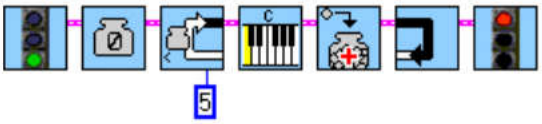
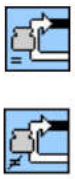
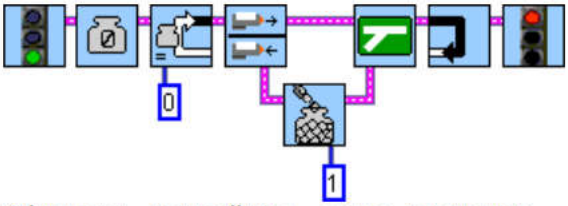
Блок	Назначение	Пример использования
	Цикл пока значение освещенности меньше (больше)	 **Включить мотор A, пока освещенность меньше 50%, повторять: {сыграть ноту, ждать паузу}, когда на момент проверки станет больше или равна 50%, выключить мотор A**
	Цикл пока расстояние меньше (больше)	 **Повторять бесконечно: {пока расстояние до объекта больше 50 см, повторять:{сыграть ноту, ждать паузу}, затем включить мотор A на 2 с и выключить}**
	Цикл пока показания датчика оборотов меньше (больше)	

Таблица 5.13. Циклы с предусловием по различным значениям[1]

Блок	Назначение	Пример использования
	Цикл пока значение таймера меньше (больше)	 **Обнулить таймер, пока его значение меньше 50 десятых долей секунды (5 с), повторять: {сыграть ноту, ждать паузу}**

[1] Если модификатор таймера не указан, по умолчанию используется красный (red).

Блок	Назначение	Пример использования
	Цикл пока значение таймера 0.01 с меньше (больше)	 **Обнулить таймер, ждать 6 с, пока значение таймера больше 600 сотых долей секунды (6 с), повторять: {издать сигнал, ждать 2 с, обнулить таймер}. Должно выполниться только один раз**
	Цикл пока значение контейнера меньше (больше)	 **Обнулить контейнер, пока значение контейнера меньше 5, повторять: {сыграть ноту, добавить в контейнер число 1}**
	Цикл пока значение контейнера равно (не равно)	 **Обнулить контейнер, пока значение контейнера равно 0, повторять: {если датчик касания нажат, добавить в контейнер число 1}**

Особое внимание среди циклов стоит уделить циклам по условию «меньше». Дело в том, что среди ветвлений отсутствует их аналог (там только «меньше или равно»). Это значит, что для выполнения действия в случае «меньше» требуется двойная проверка, что сильно загромождает программу. С помощью цикла, который выполнится только 1 раз благодаря прыжку, можно заменить отсутствующее ветвление (рис. 5.20).

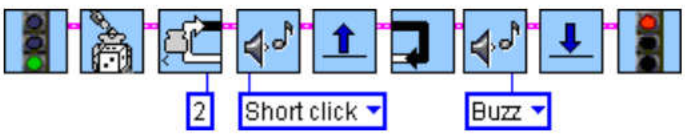


Рис. 5.20. Замена циклом ветвления по условию «меньше».

События

События — это, пожалуй, самая затейливая из управляющих структур Robolab. Она включает в себя и параллельные процессы, и ветвления, и прыжки. Только все это не на «поверхности», а встроено в обработчики событий. Начинающему робототехнику остается воспользоваться готовым решением.

Итак, порядок действий таков:

— объявляется событие,
— включается мониторинг события,
— указывается точка перехода по событию,
— выключается мониторинг события.

Рассмотрим основные команды, которые связаны с событиями (табл. 5.14). В основном блоки узнаваемы, вопрос только в модификаторах с изображением ключа, которые используются для обозначения события. Не вдаваясь в подробности, просто условимся, что в данном случае это модификатор события и его значения. А что это такое, узнаете в следующем разделе.

Таблица 5.14. Мониторинг событий.

Блок	Назначение	Пример использования
	Установить тип ожидаемого события	 **Установить на «красное» событие нажатие датчика касания, включить мониторинг красного события, повторять бесконечный сигнал; как только событие произойдет, перейти на точку перехода в конце программы**
	Начать мониторинг события	
	Закончить мониторинг события	
	Точка перехода по событию	

Модификаторы

Модификаторы — это параметры выполнения команд, которые определяют следующее:

— порт подключенного устройства,
— мощность мотора,
— длительность задержки,

— ожидаемое значение датчика,

— новое значение контейнера и др.

Например, для мотора или датчика — порт подключения, мощность или ожидаемое значение, для задержки — ее длительность и т.д. Если модификатор не указан, то он подставляется по умолчанию.

Прежде всего следует разделить два понятия для каждого объекта языка Robolab: 1) сам объект и 2) его значение. Возьмем, к примеру, кошелек с деньгами. Кошелек — это объект. Сумма денег, лежащая в нем, — это его значение. Примерно таким образом используются модификаторы: в одних ситуациях необходимо указать объект, в других — его значение. Кроме того, существуют модификаторы-константы, которые по статусу стоят рядом со значениями объектов.

Цвет модификаторов позволяет определить их тип. В среде Robolab модификаторы объектов обведены коричневой или зеленой рамкой, а значения — синей (табл. 5.15).

Таблица 5.15. Модификаторы: объекты и значения

Объект	Описание	Значение
	Порт подключения датчика (из стандартной палитры)	
	Порт подключения датчика из палитры NXT 	
	Порт подключения мотора (из стандартной палитры)	См. ниже
	Порт датчика оборотов мотора из палитры NXT 	
	Цветные контейнеры (красный — red, синий — blue, желтый — yellow). Все контейнеры имеют целочисленные значения типа int (–32767...32768).	
	Нумерованный контейнер. В качестве номера используется константа. Цветные контейнеры имеют номера 0, 1 и 2.	
	Контейнер контейнеров. По сути список состоит примерно из 30 элементов	

Объект	Описание	Значение
	Именованный контейнер. Переменная, представленная в виде контейнера (типы long и float приводятся к int)	
	Таймеры (красный — t1, синий — t2, желтый — t3). Значения в десятых долях секунды	
	Для того же красного таймера можно брать значение в тысячных долях секунды	
	Счетчик цикла. Содержит значение очередного повторения тела цикла	
—	Номер текущей задачи. Позволяет узнать, какой номер присвоен текущей задаче, чтобы воспользоваться им при остановке или удалении. Нумерация начинается с нуля.	

Модификаторы-константы

Модификаторы-константы для любой команды можно создать с помощью контекстного меню. Для этого надо выбрать инструмент «Провод», подвести его к тому месту на блоке команды, куда подключается модификатор, и щелкнуть правой кнопкой мыши. В контекстном меню выбрать: Create → Constant. Будет создана константа именно того типа, который требуется для данной команды (табл. 5.16).

Все модификаторы-константы можно найти в Палитре функций.

Таблица 5.16. Модификаторы-константы

Константа	Описание	Диапазон значений
	Значение мощности моторов	Используется для команд диапазона 1—5
	Целое или дробное число	В зависимости от блока команды

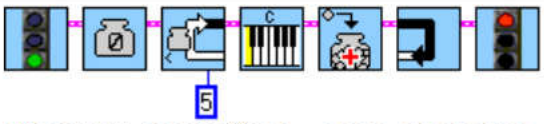
Константа	Описание	Диапазон значений
	Строковое выражение. Находится в палитре Internet 	Полное описание значений в справке к блоку 
	Выпадающий список доступных контейнеров. Список раскрывается инструментом «палец»: 	Для нужд пользователя доступны номера с 3 по 22 и с 29 по 31
	Случайное значение	От 0 до 8

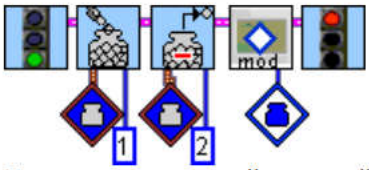
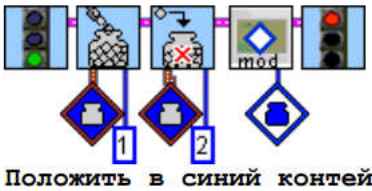
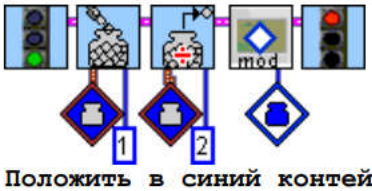
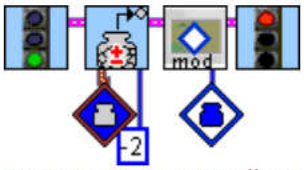
Контейнеры

Без переменных не обойдется ни одна программная среда. В Robolab они называются контейнерами. Всего в Robolab доступно 48 контейнеров, но пользователь может задействовать для своих целей около 25. Во всех командах, где модификатор контейнера требуется, но не указан, по умолчанию используется красный контейнер.

Самое главное достоинство контейнеров — возможность хранить данные и производить над ними арифметические и логические операции (табл. 5.17—5.19). Главный недостаток — все данные в контейнерах имеют целочисленный тип int, ограниченный диапазоном –32767...32768.

Таблица 5.17. Арифметические и логические операции

Блок	Назначение	Пример использования
	Инициализация. Обнулить контейнер	
	Добавить значение к текущему значению контейнера	**Обнулить контейнер, пока значение контейнера меньше 5, повторять: {сыграть ноту, добавить в контейнер число 1}**

Блок	Назначение	Пример использования
	Вычесть значение из текущего значения контейнера	 **Положить в синий контейнер число 1, вычесть из него число 2, результат вывести на экран NXT**
	Умножить текущее значение контейнера на заданное число	 **Положить в синий контейнер число 1, умножить его на 2, результат вывести на экран NXT**
	Разделить текущее значение контейнера на заданное число	 **Положить в синий контейнер число 1, разделить его на 2, результат вывести на экран NXT**
	Знак числа (число 1 или –1)	 **Положить в синий контейнер знак числа -2 (т.е. -1), результат вывести на экран NXT**
	Модуль числа	 **Положить в синий контейнер модуль числа -2 (т.е. 2), результат вывести на экран NXT**

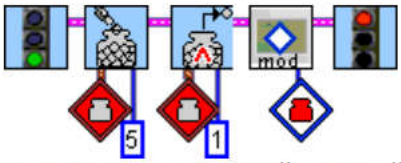
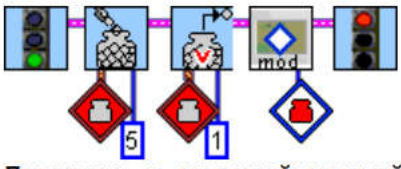
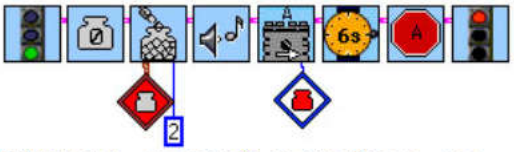
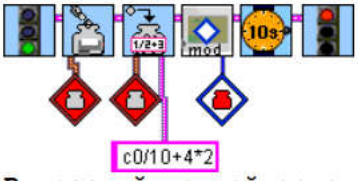
Блок	Назначение	Пример использования
	Логическое «И» 5 = 0101 1 = 0001 1 = 0001	 Положить в красный контейнер число 5, побитово умножить его на число 1, результат вывести на экран NXT
	Логическое «ИЛИ» 5 = 0101 1 = 0001 5 = 0101	 Положить в красный контейнер число 5, побитово сложить его с числом 1, результат вывести на экран NXT

Таблица 5.18. Операции присваивания

Блок	Назначение	Пример использования
	Присваивание. Положить новое значение в контейнер	 Обнулить красный контейнер, положить в него число 2, издать сигнал, включить мотор A с мощностью, равной значению красного контейнера, ждать 6 с, выключить мотор A
	Формула	 В красный контейнер считать показания датчика освещенности, заменить их на результат формулы, где c0 — значение красного контейнера, вывести на экран NXT

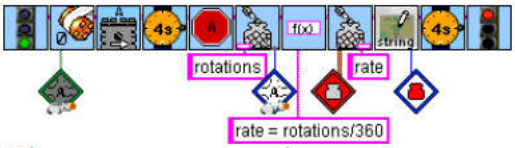
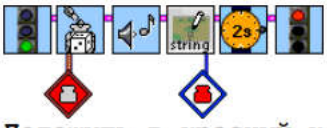
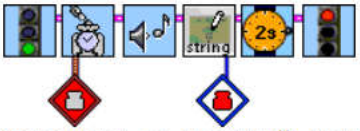
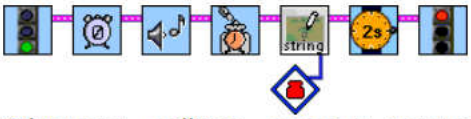
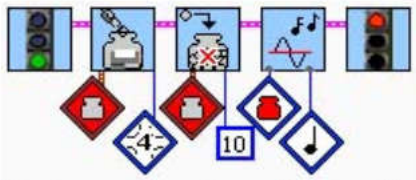
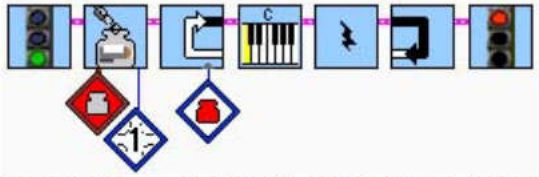
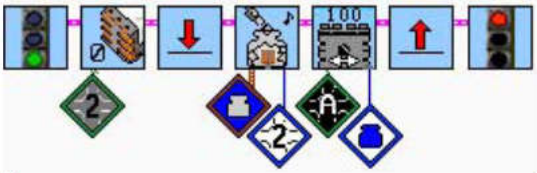
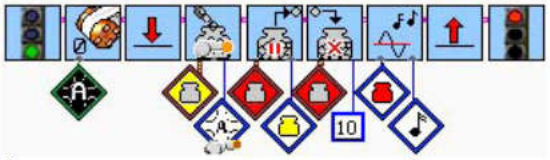
Блок	Назначение	Пример использования
	Вычисление выражения. Результат присвоить переменной типа int, long или float. Можно выполнить сразу несколько присваиваний в одном блоке	 **Обнулить датчик оборотов мотора A, включить мотор A, ждать 4 с, выключить мотор A, считать показания с порта A в переменную rotations (в градусах), в переменную rate положить значение в оборотах, в красный контейнер положить значение переменной rate, вывести на экран NXT**
	Положить значение выражения в контейнер	
	Положить значение в именованный контейнер	
	Положить в контейнер случайное число от 0 до 8	 **Положить в красный контейнер случайное число, издать сигнал, вывести на экран значение красного контейнера, ждать 2 с**

Таблица 5.19. Присваивание специальных значений

Блок	Назначение	Пример использования
	Положить в контейнер значение таймера 1/10 с	 **Положить в красный контейнер показания таймера, издать сигнал, вывести на экран значение красного контейнера, ждать 2 с**
	Положить в контейнер значение таймера 1/100 с	 **Обнулить таймер, издать сигнал, положить в красный контейнер показания таймера в сотых долях секунды, вывести на экран значение красного контейнера, ждать 2 с**

Блок	Назначение	Пример использования
	Положить в контейнер значение датчика освещенности	 **Положить в красный контейнер показания датчика освещенности на четвертом порту, умножить на 10, играть ноту с длительностью 1/4 и частотой, равной значению красного контейнера**
	Положить в контейнер значение датчика расстояния	 **Инициализировать датчик расстояния на 4 порту, бесконечно повторять: {положить в контейнер показания датчика расстояния на четвертом порту, умножить их на 5, включить мотор A с полученной мощностью}**
	Положить в контейнер значение датчика нажатия (0 или 1)	 **Положить в красный контейнер значение датчика нажатия на 1 порту, повторить заданное в контейнере число раз: {играть ноту, сделать паузу}. Цикл выполнится не более 1 раза.**
	Положить в контейнер значение датчика звука	 **Инициализировать датчик звука на 2 порту, бесконечно повторять: {положить в контейнер показания датчика звука на втором порту, включить мотор A с полученной мощностью}**

Блок	Назначение	Пример использования
	Положить в контейнер значение датчика оборотов	 **Инициализировать датчик оборотов на моторе A, бесконечно повторять: {положить в желтый контейнер показания датчика оборотов мотора на порту A, положить в красный контейнер модуль значения желтого контейнера, умножить значение красного на 10, играть ноту с длительностью 1/16 и частотой, равной значению красного контейнера}**

Операции с выражениями

В Robolab 2.9.4 появился целый набор возможностей операций с выражениями в текстовом режиме: от объявления типов переменных до обращения почти к каждому устройству по имени.

Подобные выражения заключаются в розовые прямоугольники и присоединяются к различным блокам, которые используют их результаты в качестве параметров.

Синтаксис аналогичен языку С.

Базовые операции над числами

Сложение: +

Вычитание: –

Умножение: *

Деление: /

Остаток от деления: %.

```
Пример: 3 % 2 = 1.
```

Битовое «И»: &.

```
Пример: 7 & 5 = 5.
```

Битовое «ИЛИ»: |.

```
Пример: 5 | 2 = 7.
```

Определение типов переменных: int, long, float.

```
Пример:
int b
long c=2147483647
float a=2.5
```

По умолчанию все переменные целые.

Разрешены круглые скобки.

Пример: 5+(red*2)/abs(sqrt(yellow/5.0)).

Указание числа 5.0 означает, что все математические операции будут производиться над числом с плавающей точкой.

Если к переменной целого типа присвоить дробное значение, оно автоматически будет приведено к целому, т. е. произойдет округление. Поэтому рекомендуется заранее определять тип переменных.

Допустимые функции

Тригонометрические (все параметры в радианах): sin, cos, asin, acos, atan. Пример: sin (1.57) = 1.

Стандартные: abs, sqrt, sgn. Пример: sgn(-2) = -1.

Логарифмические: ln10, ln, exp.

Предопределенные переменные среды Robolab

Все перечисленные далее идентификаторы зарезервированы в среде Robolab, и программист не может использовать их для обозначения своих переменных.

Контейнеры: red, blue, yellow, c0, c1, c2, c3,…,c47 (или C0...C47).

При этом c0 — красный контейнер, c1 — синий, c2 - желтый. Контейнеры с 0 по 31 являются глобальными целочисленными переменными. Контейнеры с 32 по 47 — тоже целочисленные переменные, но локальные для каждой задачи. Кроме того, частично они используются в циклах.

Счетчики циклов с *n* повторений: Loop_i, Loop_j, Loop_k (изменяются n-1 до 0). Им соответствуют контейнеры c33-c46 (максимум 14 вложенных циклов, причем счетчик внешнего цикла записан в контейнере c33).

Датчики оборотов моторов (энкодеры): e1, e2, e3 (соответствуют моторам A, B, C).

Случайное число: r. Максимальное значение 255.

Для примера, r8 возвращает случайное значение от 0 до 8.

Датчики: s1, s2, s3, s4 (или S1...S4 или Sensor1...Sensor4).

Необработанные «сырые» показания датчиков:
SensorRaw1...SensorRaw4. Возвращают значение от 0 до 1023.

Таймеры с интервалом 100 мс:
T1, T2, T3, T4 (или Timer100ms1...Timer100ms4, или t1...t4).

Таймеры с интервалом 10 мс: Timer10ms1...Timer10ms4.

Таймеры с интервалом 1 мс: Timer1ms1...Timer1ms4.

Текущее состояние моторов: MotorStatus1...MotorStatus3 (0=Свободное, 1=Удержание, 16=Разгон, 32=Движение, 64=Торможение).

Содержимое почтового ящика: m.

В блоке Evaluate Expression, с надписью f(x) на пиктограмме, возможно использование именованных переменных, которые размещаются контейнерах, начиная с c48, и могут быть двух типов: int и float. Программист не имеет прямого доступа к этим контейнерам, а только по имени, их общий объем ограничен 48 двухбайтовыми ячейками. Таким образом, декларируется, что всего в Robolab можно использовать до 96 целочисленных переменных: 48 нумерованных и 48 именованных. Следует помнить, что переменные типа float занимают 4 байта, а представленные с помощью контейнера преобразуются в тип int. На деле автору удалось воспользоваться 20 именованными переменными типа float или 42 именованными переменными типа int.

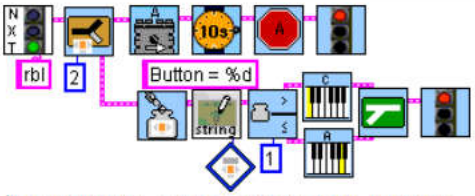
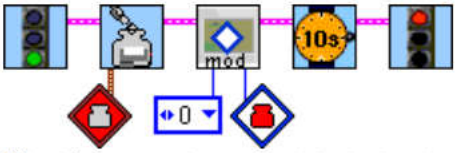
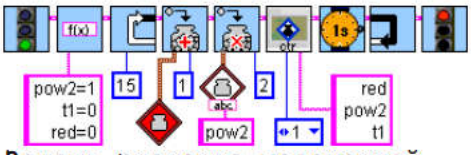
Исходя из перечисленных возможностей, опытный программист в небольших программах скорее всего вообще откажется от контейнеров в Robolab и будет пользоваться только блоком для записи выражений. Впрочем, эта замечательная возможность пригодится и начинающим, особенно в главе «Алгоритмы управления». А специалисты могут приступить к изучению главы 6, посвященной основам языка RobotC.

Интерфейс NXT

Используя палитру NXT, можно получить доступ к управлению дисплеем и кнопками контроллера. Во-первых, как уже было замечено в некоторых примерах, существует возможность выводить текстовую и числовую информацию на 8 строк экрана NXT. Во-вторых, возможен графический вывод на экран. И наконец, несколько необычным способом можно получить доступ к сбору данных о нажатии клавиш контроллера. Рассмотрим несколько примеров (табл. 5.20).

Таблица 5.20. Возможности палитры NXT

Блок	Назначение	Пример использования
	Возвращает в контейнер статус кнопок NXT: –1 — не нажато, 0 — нижняя кнопка (выход) 1 — стрелка вправо 2 — стрелка влево 3 — оранжевая кнопка (ввод)	**См. ниже**

Блок	Назначение	Пример использования
	Создает параллельную задачу контроля за кнопками NXT. Параметр — число нажатий на кнопку «Выход» для завершения программы. Возможна в единственном экземпляре	 **Запустить параллельную задачу контроля кнопок NXT, в которой получить их состояние в красный контейнер; при изменении выводить статус кнопок на экран (пример: Button = 1); если нажата стрелка влево или оранжевая кнопка, играть ноту До (C), иначе — ноту Ля (A); повторить выполнение задачи. В основной задаче ничего значимого не происходит: включить мотор A вперед, ждать 10 с, выключить мотор A**
	Возвращает статус кнопок NXT (см. выше)	
	Позволяет вывести значение контейнера в заданном формате. Строка формата может содержать %d для целых чисел и %f — для дробных (как в языке C). Параметр Row — номер строки на экране от 0 до 7	
	Вывод на экран значения модификатора. Параметр Row — номер строки на экране от 0 до 7, в которую выводится значение указанного модификатора. Если модификатор позволяет, значение может быть выведено в типе long	 **Показания датчика освещенности записать в красный контейнер и вывести его значение на экран в нулевой строке, подождать 10 с (чтобы прочитать значение)**
	Вывод на экран значений нескольких контейнеров. Параметр Row — номер строки на экране от 0 до 7. Параметр expressions содержит до восьми имен контейнеров в столбик. Разрешено использовать именованные и нумерованные (c1...c48) контейнеры, имена датчиков (s1...s4), таймеры (t1...t4), энкодеры (e1...e3) и другие предопределенные переменные среды Robolab	 **Задать значение переменной pow2, обнулить первый таймер и красный контейнер red. Повторить 15 раз: {добавить в красный контейнер 1, домножить переменную pow2 на 2, вывести на экран (с первой строки) значение красного контейнера, переменной pow2 и показания таймера, подож-**

Блок	Назначение	Пример использования
		дать 1 с}. На последнем шаге цикла произойдет переполнение и значение pow2 = 2^{15} будет выведено как -32678.
	Очистить экран NXT	-

Библиотеки пользователя

В Robolab существует много интересных команд и приемов программирования. Надеемся, что читатель найдет время, чтобы самому разобраться в замечательных возможностях этой среды. Если же повезет, и в добавление к стандартному набору появятся новые датчики от сторонних фирм-производителей, стоит заглянуть в палитру User libraries, в которой содержатся команды для управления некоторыми из них: Hitechnic, Mindsensors, Vernier (рис. 5.21).

Рис. 5.21. Расширенная Палитра User libraries.

В 2010 г. было выпущено очередное обновление для Robolab, в котором появилась поддержка новых датчиков сторонних производителей (см. приложение 3). Таким образом, популярность Robolab среди пользователей продлится еще на несколько лет, пока не появится более совершенная среда программирования для начинающих.

Глава 6. Программирование в RobotC

Введение

Язык программирования RobotC отличается от стандартного C расширенным набором команд по работе с устройствами микроконтроллера. Опытный программист найдет эту среду гораздо более удобной, чем пакеты графического программирования. Для тех же, кто не знаком с языком C, текстовое программирование микроконтроллеров может показаться недостаточно наглядным. Однако именно в текстовом режиме можно составлять наиболее сложные и эффективные программы.

Firmware

Операционная система, предназначенная для исполнения на NXT программ, написанных на RobotC, отличается от аналогичных прошивок микроконтроллера для Robolab или NXT-G. Хотя внешнее сходство присутствует. Однако при попытке исполнения программы, загруженной из другой среды, будет выдаваться сообщение об ошибке. Поэтому перед началом работы необходимо обеспечить загрузку соответствующего Firmware (рис. 6.1).

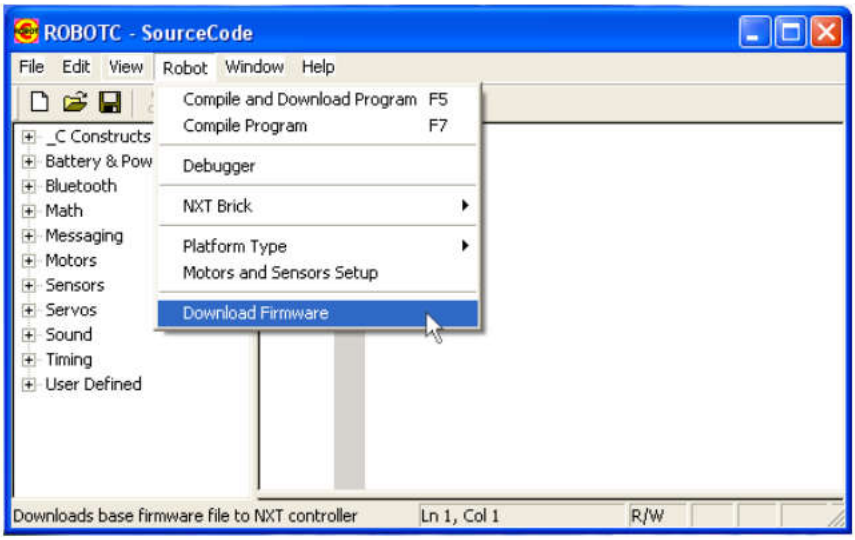


Рис. 6.1. Загрузка операционной системы на NXT.

Hello, world!

Традиционно первая программа на языке Си должна вывести на экран приветствие миру: «Hello, world!». Воспользовавшись возможностями NXT, исполним традицию:

```
task main()
{
  nxtDisplayTextLine(0, "Hello, world!");
  wait1Msec(5000);
}
```

Команда nxtDisplayTextLine(0, "Hello world!"); выведет в строку экрана NXT с номером ноль сообщение «Hello world!». По прошествии 5 с программа закончит выполнение и текст исчезнет. Сохраните программу на жесткий диск с именем "hello.c".

Для загрузки в микроконтроллер подключите его к компьютеру, включите NXT и нажмите на клавиатуре компьютера клавишу F5. Если программа загрузилась успешно, на экране компьютера появится окно отладки Program Debug (рис. 6.2).

Рис. 6.2. Окно отладки программы.

Если нажать в нем кнопку Start, программа запустится на NXT. Можно игнорировать это окно, найти в меню микроконтроллера файл hello и запустить его оранжевой кнопкой.

Усовершенствуем программу. То же сообщение можно вывести по центру экрана:

```
task main()
{
  nxtDisplayCenteredTextLine(3, "Hello world!");
  wait1Msec(5000);
}
```

На этот раз сообщение будет выведено в третьей строке дисплея NXT с выравниванием посередине.

Используя набор команд по работе с экраном NXT, в том числе и графических, можно написать собственные миниигры для микроконтроллера. Но это тема отдельной главы или даже книжки.

Структура программы

Теперь обратим внимание на структуру исходного кода. Главная задача, с которой начинается выполнение программы в RobotC, называется task main(). Тело задачи располагается между двумя фигурными скобками: открывающей и закрывающей. Все команды должны быть размещены между ними.

Более подробно о задачах написано в разделе «Параллельные задачи».

Управление моторами

Состояние моторов

Простейшая задача: вращать мотор C вперед 3 с:

```
task main()
{
  motor[motorC] = 100;
  wait1Msec(3000);
}
```

Команда motor[] представляет собой массив из трех элементов, каждый из которых определяет текущее состояние вращения моторов A, B и C. Действие «полный вперед» задается числом 100, «полный назад» — числом –100, остановка — числом 0. Действие этой команды можно считать мгновенным. После ее выполнения мотор включается и продолжает работать до тех пор, пока не будет остановлен другой аналогичной командой.

Команда wait1Msec() задает время ожидания в миллисекундах (1 мс = 1/1000 с). Таким образом, wait1Msec(3000) означает «ждать 3 секунды».

Теперь сформулируем задачу для робота более четко: проехать вперед 2 секунды со средней скоростью и остановиться. В программе последовательно включаются моторы C и B с мощностью 50, работают в течение 2 с, после чего последовательно выключаются. По сути вклю-

чение обоих моторов произойдет практически одновременно, как и их выключение:

```
task main()
{
  motor[motorC] = 50;
  motor[motorB] = 50;
  wait1Msec(2000);
  motor[motorC] = 0;
  motor[motorB] = 0;
}
```

Следующий пример демонстрирует возожность бесконечного вращения мотора с помощью цикла while. Остановить его можно, прервав выполнение программы на NXT:

```
task main()
{
  while (true)
    motor[motorA]=100;
}
```

Следующий пример работает аналогично:

```
task main()
{
  motor[motorA]=100;
  while (true)
    wait1Msec(1);
}
```

Миллисекундная задержка полезна для разгрузки контроллера при работе бесконечного цикла.

Встроенный датчик оборотов

Команда nMotorEncoder[] — это обращение к датчику оборотов мотора, который возвращает значение в градусах. Каждый из трех элементов этого массива имеет 16-битное значение, что дает возможность хранить число в диапазоне –32768...32767, это соответствует 95 полным (360 градусов) оборотам моторов. В длительно работающих программах следует время от времени обнулять значения массива во избежание переполнения.

```
nMotorEncoder[motorA] = 0;
```

Этим же действием датчик оборотов инициализируется для начала измерений.

Следующая программа обеспечит вращение мотора A на 1000 градусов.

```
task main()
{
  motor[motorA]=50;
  while (nMotorEncoder[motorA] < 1000)
  {
  // никаких действий можно не производить
  }
}
```

Команда nMotorEncoderTarget[] обеспечивает остановку мотора при повороте на заданное число градусов. Команда обладает собственным «интеллектом» и, учитывая инерцию движения, может откорректировать окончательную позицию мотора.

```
task main()
{
  nMotorEncoderTarget[motorA] = 1000;
  motor[motorA] = 50;
  wait1Msec(10000);
}
```

Синхронизация моторов

На практике часто можно заметить, что движение колесного робота непрямолинейно. Это может быть связано с различным трением приводов, разницей нагрузок на колеса и т. п. Команда nSyncedMotors позволяет синхронизовать моторы, объявив один из них главным, второй — подчиненным (один ведущим, второй — ведомым):

```
nSyncedMotors = synchNone; // Отключить синхронизацию
nSyncedMotors = synchAC; // Мотор C подчинен мотору A
```

В режиме синхронизации достаточно управлять только одним мотором: второй будет в точности повторять его действия.

Переменная nSyncedTurnRatio позволяет изменить соотношение движения синхронизированных моторов. Ее значения изменяются от –100 до 100 %. Отрицательный знак указывает на противоположное направление движения подчиненного мотора. Абсолютное значение показывает отношение скорости ведомого мотора к скорости ведущего. Если оно меньше 100, робот поворачивает.

В следующем фрагменте программы робот проедет по прямой, а затем повернется:

```
nSyncedMotors = synchAC; // Мотор C подчинен мотору A

// Движение по прямой
nSyncedTurnRatio = +100; // Режим движения прямо
nMotorEncoder[motorA] = 0;
// остановиться через 1000 градусов
nMotorEncoderTarget[motorA] = 1000;
motor[motorA] = 100;

while (nMotorEncoder[motorA] < 1000) // дождаться остановки
{}

// Повернуться на месте

nSyncedTurnRatio = -100; // Режим поворота
// остановиться через 200 градусов
nMotorEncoderTarget[motorA] = 200;
motor[motorA] = 50;
wait1Msec(3000);
```

Режим импульсной модуляции

Скорость моторов контролируется технологией широтно-импульсной модуляции (ШИМ, pulse width modulation — PWM). Импульсы тока подаются на моторы тысячи раз в секунду. Каждый импульс представляет из себя волну, содержащую период наличия напряжения (on-time) и период отсутствия напряжения (off-time). Отношение между этими двумя периодами определяет мощность, подаваемую на мотор. В периоды on-time используется полный ресурс батареи, и эта технология более эффективна, чем регуляция скорости с помощью изменения напряжения.

В периоды off-time, когда напряжение на моторы не подается, цепь может находиться в двух состояниях: разомкнутом и замкнутом. Разомкнутая цепь получается в режиме плавающего напряжения, замкнутая — в режиме торможения. RobotC позволяет выбрать режим как в установках среды, так и с помощью специальной переменной bFloatDuringInactiveMotorPWM. При компиляции программы будет установлен выбранный режим:

```
bFloatDuringInactiveMotorPWM = false; // режим торможения
bFloatDuringInactiveMotorPWM = true; // плавающий режим
```

Для моторов NXT предпочтительным является режим торможения, он и выставляется по умолчанию.

Для примера в среде Robolab базовые команды управления моторами включают их в режиме плавающего напряжения, а продвинутые команды (со значениями мощности от –100 до 100) в режиме торможения.

Зеркальное направление

В некоторых конструкциях роботов движение моторов вперед противоположно тому, которое представляется логичным. Чтобы не менять в программе все знаки на противоположные, достаточно установить значение соответствующего элемента массива bMotorReflected[] для конкретного мотора:

```
bMotorReflected[motorA] = true;
// меняет направление мотора
```

Датчики

Настройка моторов и датчиков

Motors and sensors setup — один из ключевых пунктов среды RobotC. В этом подразделе меню Robot осуществляется конфигурирование всех устройств, подключенных к микроконтроллеру (рис. 6.3).

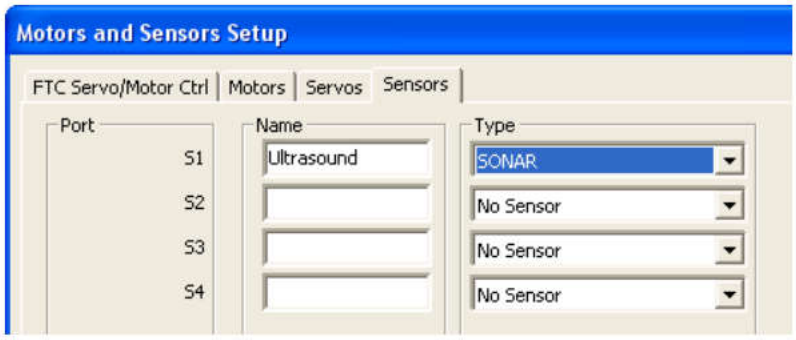


Рис. 6.3. Окно настройки датчиков.

Сделанные изменения сохраняются в программе в виде автоматически сгенерированных строк кода. Благодаря этому при переносе программы на другой компьютер или при подключении другого робота повторять настройку не требуется:

```
#pragma config(Sensor, S1, Ultrasound, sensorSONAR)
//*!!Code automatically generated by 'ROBOTC' configuration
wizard!!*//
```

В данном примере Ultrasound — это имя, присвоенное ультразвуковому датчику пользователем. Далее оно может использоваться в программе вместо указания порта S1 подключения датчика расстояния. Такая возможность дана потому, что при модификации робота конкретный порт может измениться. Чтобы не переписывать всю программу, достаточно поправить начальные установки. Вот пример программы с определенными названиями моторов и датчиков:

```
#pragma config(Sensor, S1, Ultrasound, sensorSONAR)
#pragma config(Motor,motorA,LMotor,tmotorNormal,PIDControl,
)
#pragma config(Motor,motorB,RMotor,tmotorNormal,PIDControl,
)
//*!!Code automatically generated by 'ROBOTC' configuration
wizard!!*//

task main()
{
  while(SensorValue[Ultrasound]>20)
  {
    motor[LMotor]=50;
    motor[RMotor]=50;
  }
}
```

Программа выглядит длиннее, но в ней четко указано, какой мотор на какой порт следует подключать: левый LMotor — на порт A, правый RMotor — на порт B. Впоследствии при построении сложных программ это может пригодиться. На рис. 6.4 показано, как были сконфигурированы моторы.

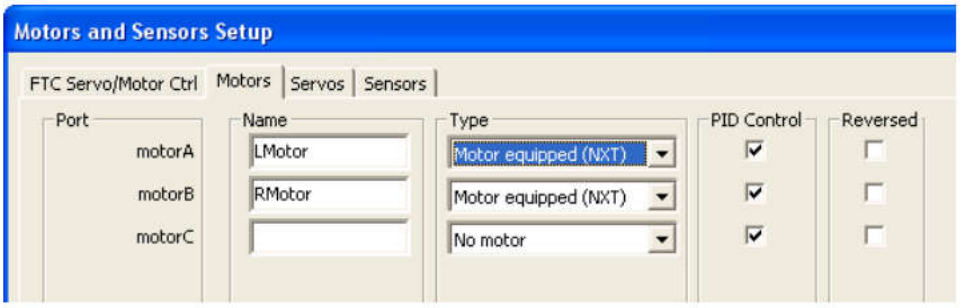


Рис. 6.4. Окно настройки моторов.

Другой способ конфигурации, предлагаемый в примерах RobotC, выглядит как комментарий. Однако это системный работающий код, заключенный в скобки из специальных символов: //*!! !!*//.

```
//*!!Sensor, S4, sonarSensor, sensorSONAR, , !!*//
//*!! !!*//
//*!!Start automatically generated configuration code.!!*//
const tSensors sonarSensor = (tSensors) S4;
```

В данном примере ультразвуковой датчик конфигурируется на 4-й порт и ему сопоставляется имя sonarSensor.

В списке поддерживаемых датчиков есть базовый набор, а также некоторые датчики сторонних производителей, например, Hitechnic. При отсутствии нужного датчика в списке можно попробовать сконфигурировать его как один из уже имеющихся. Как правило, это описывается в прилагаемой к нему документации. Например, Hitechic IrSeeker может быть сконфигурирован как Sonar — ультразвуковой датчик, а показания выдает в своем рабочем диапазоне. Правда, при этом используются не все его возможности. Существуют другие более сложные способы работы с датчиками.

Тип датчика

С помощью управляющего массива SensorType[] можно указать тип датчика в теле функции.

```
SensorType[sonarSensor]=sensorSONAR;
```

Указание типа датчика важно для его правильного функционирования. В прошивке NXT содержатся драйверы определенного набора датчиков. В идеале система сама определяет, какой датчик подключен, но не во всех случаях это возможно. Вообще говоря, в стандартном виде на порт датчика подается необработанное значение от 0 до 1023, и благодаря драйверу оно преобразуется в удобочитаемый диапазон. Опытные программисты могут воспользоваться документацией к датчику и анализировать его показания в формате RAW.

Задержки и таймеры

Задержки

Управление роботом с использованием задержек времени довольно просто, но таит в себе «подводные камни». Во-первых, при исполнении задержки в задаче почти полностью теряется контроль за показаниями датчиков. Во-вторых, неизбежное изменение заряда батарей ведет к различным эффектам на одних и тех же промежутках времени. И, наконец, немаловажную роль играет окружающая среда. Даже простейший поворот робота на месте в зависимости от материала поверхности (например, линолеум пола или ламинированный ДСП столешницы) может носит различный характер. Поэтому при необходимости воспроизведения точных движений робота вместо временны*х* интервалов стоит использовать датчик оборотов двигателей.

Но есть ситуации, в которых задержки необходимы. Например, если робот должен войти в плотное соприкосновение с предметом, то ожидание оборотов моторов может привести к зависанию системы: мотор просто застопорится, не довернувшись на несколько градусов, а программа остановится. В этом случае лучше использовать задержку.

В RobotC всего два типа задержек: в тысячных и сотых долях секунды:

```
wait1Msec(N); // Жди N миллисекунд (N тысячных секунды)
wait10Msec(N); // Жди N сотых долей секунды
```

Поскольку обращение к датчикам и моторам в RobotC может осуществляться тысячи раз в секунду, для очень тонкого управления может потребоваться первая команда. Вторую команду удобнее использовать, когда высокая точность не требуется.

Таймеры

В системе допустимо использование четырех таймеров: T1, T2, T3, T4. Каждый из них может быть задействован с определенной точностью: от 1 до 100 мс. Команда ClearTimer(Timer) включает конкретный таймер и обнуляет его значение:

```
ClearTimer(T1);
```

Три массива хранят значения всех четырех таймеров, соответственно в тысячных, сотых и десятых долях секунды:

```
time1[]
time10[]
time100[]
```

Следующий пример в течение 10 с выводит на экран показания таймера T1 с различной точностью. В нем же можно увидеть, как таймеры можно использовать для контроля за временем работы циклических алгоритмов:

```
task main()
{
  ClearTimer(T1);
  while(time100[T1]<100)
  {
    nxtDisplayTextLine(0, "%d", time1[T1]);
    nxtDisplayTextLine(1, "%d", time10[T1]);
    nxtDisplayTextLine(2, "%d", time100[T1]);
    wait1Msec(1);
  }
}
```

Параллельные задачи

Управление задачами

Программа в RobotC может состоять из нескольких задач и подпрограмм, главная из которых называется task main(). Эта задача и вызывается при запуске программы, а из нее при необходимости могут быть вызваны остальные с помощью команды StartTask:

```
task first()
{
  motor[motorA] = 100; // Мотор А вперед
  wait1Msec(1000);
}
task second()
{
  motor[motorA] = -100; // Мотор А назад
  wait1Msec(1000);
}
task main() {
  StartTask(first);
  StartTask(second);
}
```

Допускается одновременное использование до 10 параллельных задач. Остановить конкретную задачу можно командой StopTask. Все задачи останавливаются командой StopAllTasks:

```
task first()
{
  while(true)
    motor[motorA] = 100; // Мотор А вперед в бесконечном
цикле
}
task second()
{
  wait1Msec(1000);
  StopTask(first); // Остановить первую задачу
  motor[motorA] = -50; // Мотор А назад
  wait1Msec(10000); // Задержка на 10 секунд
}
task main()
{
  StartTask(first);
  StartTask(second);
  wait1Msec(2000);
  StopAllTasks(); // Остановить все задачи
}
```

Время процессора распределяется между задачами таким образом, что каждой достается небольшой отрезок времени, за который она выполняет несколько инструкций. Поскольку задачам присваивается приоритет от 0 до 255, первыми выполяются те, чей приоритет более высокий. Задачи с низшим приоритетом находятся в ожидании, пока в очереди присутствуют задачи с высшим приоритетом.

Команда hogCPU() передает все процессорное время текущей задаче до тех пор, пока не встретится команда releaseCPU(). Таким образом, на время можно полностью остановить задачи даже с очень агрессивным поведением:

```
task first()
{
  while(true)
    motor[motorA] = 100; // Мотор А вперед бесконечно
}
task second() {
  wait1Msec(1000);
  hogCPU(); // Захватить управление
  motor[motorA] = -100; // Мотор А назад
  wait1Msec(1000); // Ждать 1 секунду
  releaseCPU(); // Вернуть управление
}
```

```
task main()
{
  StartTask(first);
  StartTask(second);
}
```

Работа с датчиком в параллельных задачах

В параллельном программировании важно соблюдать одно правило: не обращаться к одному устройству единовременно из разных задач. Такое обращение может привести к сбоям в его работе.

Для использования показаний датчиков, например, можно в одной задаче постоянно считывать их в отдельную переменную, а в других задачах использовать значение этой переменной:

```
int Light1;
task first()
{
  while(Light1>40) // Пока датчик на светлом
    motor[motorA] = 100; // Мотор А вперед
  StopAllTasks(); // Увидев темное, остановить все
}
task second()
{
  while(true)
    Light1=SensorValue[S1]; // Считывание показаний датчика
}
task main()
{
  SensorType[S1]=sensorLightActive; // Объявляем датчик
  StartTask(second);
  wait1Msec[100]; // Задержка, чтобы датчик начал работу
  StartTask(first);
  while(true)
    if (Light1>50) // При высокой освещенности пиликать
      PlaySound(soundBeepBeep);
}
```

Параллельное управление моторами

При управлении моторами ситуация несколько иная. Вот, к примеру, описанная ранее программа из раздела «Моторы вперед!»:

```
task main()
{
  while(true)
    motor[motorA] = 100;
}
```

Если требуется в какой-то момент перехватить управление из параллельной задачи, то такой цикл быстро смешает все карты, поскольку на каждом витке цикла обновляется мощность мотора. Поэтому лучше использовать другой вариант этой же структуры:

```
task main()
{
  motor[motorA] = 100; // Мотор А вперед
  while(true); // Бесконечный пустой цикл
}
```

Команда управления мотором поступает только 1 раз, но с этого момента подача напряжения осуществляется постоянно в течение работы пустого бесконечного цикла.

Вот пример, в котором направление вращения мотора изменяется из параллельной задачи через 1 секунду после начала работы:

```
task second()
{
  wait1Msec(1000);
  motor[motorA] = -100; // Мотор А назад
}
task main()
{
  StartTask(second); // Вызов параллельной задачи
  motor[motorA] = 100; // Мотор А вперед
  while(true);
}
```

В следующем примере рассматривается управление мотором А из двух параллельных задач с использованием флага:

```
int flag=1;
task first()
{
  motor[motorA] = 100; // Мотор А вперед
  wait1Msec(1000);
  flag=0;
}
task second() {
  while(flag=1); // Пустой цикл
  motor[motorA] = -100; // Мотор А назад
  wait1Msec(1000);
}
task main() {
  StartTask(second);
  StartTask(first);
}
```

Даже тот факт, что задача *second* вызвана первой, не позволит ей начать управление мотором, пока задача *first* не обнулит *flag*, а это произойдет ровно через 1 секунду. Таким образом, сперва мотор будет крутиться в течение 1 секунды вперед, затем столько же времени назад.

Графика на экране NXT

Графические возможности NXT сравнимы с возможностями сотового телефона с черно-белым дисплеем. Этого достаточно, чтобы вывести необходимые данные, нарисовать график или запрограммировать небольшую игру. Размер экрана NXT составляет 100 × 64 точки, начало координат расположено в левом нижнем углу, ось *OX* — горизонтальная, *OY* — вертикальная. Базовые графические команды приведены в табл. 6.1.

Таблица 6.1. Основные графические команды языка RobotC

Команда	Назначение	Параметры
eraseDisplay()	Очистка экрана	—
nxtSetPixel(xPos, yPos)	Нарисовать точку	xPos, yPos — координаты точки
nxtClearPixel(xPos, yPos)	Стереть точку	То же
nxtDrawLine(xPos, yPos, xPosTo, yPosTo)	Нарисовать линию	xPos, yPos — координаты начальной точки, xPosTo, yPosTo — координаты конечной точки
nxtEraseLine(xPos, yPos, xPosTo, yPosTo)	Стереть линию	То же
nxtDrawRect(Left, Top, Right, Bottom)	Нарисовать прямоугольник	xPos, yPos — координаты левого верхнего угла, Right, Bottom — координаты правого нижнего угла
nxtFillRect(Left, Top, Right, Bottom)	Нарисовать закрашенный прямоугольник	То же
nxtEraseRect(Left, Top, Right, Bottom)	Стереть прямоугольник	То же
nxtDrawCircle(Left, Top, Diameter)	Нарисовать окружность	Left, Top — координаты левого верхнего угла прямоугольника, в который вписана окружность, Diameter — диаметр

Команда	Назначение	Параметры
nxtDrawEllipse(Left, Top, Right, Bottom)	Нарисовать эллипс	Left, Top и Right, Bottom — координаты диагонально противоположных углов прямоугольника, в который вписан эллипс
nxtFillEllipse(Left, Top, Right, Bottom)	Нарисовать закрашенный эллипс	То же
nxtEraseEllipse(Left, Top, Right, Bottom)	Стереть эллипс	То же
nxtDisplayStringAt(xPos, yPos, sFormatString, parm1, parm2)	Вывести строку в заданную позицию	xPos, yPos — координаты левого нижнего угла, sFormatString — строка форматирования, parm1, parm2 — значения
nxtDisplayBigStringAt(xPos, yPos, sString)	Вывести крупную строку	xPos, yPos — координаты, sString — строка

Рассмотрим пример вывода графика показаний датчика (например, звука или освещенности). Данные считываются в течение 10 с, задержка составляет 0.1 с. Таким образом производится 100 измерений, что соответствует 100 точкам горизонтальной оси времени. Значения вертикальной оси будут соответствовать показаниям датчика.

```
task main()
{
  for(int x=0; x<100; x++)
  {
    int y=SensorValue[S1];
    nxtSetPixel(x,y);
    wait1Msec(100) ;
  }
  while(true) wait1Msec(1);
}
```

Если показания датчика выйдут за пределы высоты экрана, точки просто не будут отображаться. Чтобы увидеть весь график, следует произвести масштабирование по оси *OY*. Коэффициент масштабирования 64.0 / 100 следует записать в дробных числах, чтобы, с одной стороны, избежать переполнения, а с другой — нежелательных последствий целочисленного деления. Подумайте, каким должен быть коэффициент для отображения показаний ультразвукового датчика во весь экран NXT? И что делать, если необходимо отобразить отрицательные значения энкодера?

При резких перепадах значений возникают разрывы в графике, связанные с рисованием по точкам. Использование команды рисования линии сделает график более наглядным. Рассмотрим усовершенствованную программу:

```
task main()
{
  int y=SensorValue[S1]*(64.0/100);
  wait1Msec(100) ;
  for(int x=1; x<100; x++)
  {
    int ynew=SensorValue[S1]*(64.0/100);
    nxtDrawLine(x-1,y,x,ynew);
    y=ynew;
    wait1Msec(100) ;
  }
  while(true) wait1Msec(1);
}
```

Следующая интересная задача — изображение громкости в виде круга, расположенного в центре экрана. Диаметр круга соответствует уровню громкости на датчике звука. Можно было бы воспользоваться командой nxtDrawCircle(), однако у нее нет версии с заливкой. Поэтому остановим свой выбор на nxtFillEllipse(). Обеспечив смещение центра круга в точку с координатами (50, 32), добьемся также стирания каждого предыдущего рисунка. Использование команды eraseDisplay() сильно замедляет программу. Поэтому стирать будем только ту фигуру, которая была нарисована:

```
task main()
{
  int d=0;
  while(true)
  {
    nxtEraseEllipse(50-d,32+d,50+d,32-d);
    d=SensorValue[S1];
    nxtFillEllipse(50-d,32+d,50+d,32-d);
    wait1Msec(50);
  }
}
```

Попробуйте запрограммировать отображение показаний датчика в виде прямоугольника, увеличивающегося вверх или вправо.

Еще одна интересная задача — расположить два графика, две столбчатые диаграммы или два пульсирующих круга на одном экране так, чтобы они не пересекались.

Массивы

Возможности использования массивов выгодно отличают RobotC от Robolab и NTX-G. По опыту автора, объем доступной памяти NXT позволяет использовать до 7000 целочисленных элементов, что соответствует примерно 14 килобайтам. Этого достаточно для хранения данных небольшой карты размером с экран NXT или траектории длиной 20 м, по которой проехал робот.

Рассмотрим пример, в котором показания датчика сперва записываются в массив, а затем после сигнала выводятся на экран (при этом запись данных сопровождается мигающей надписью):

```
task main()
{
  byte mas[100];
  for(int i=0; i<100; i++)
  {
    if (i%10<5)
      nxtDisplayBigStringAt(1, 30, "Record…");
    else
      eraseDisplay();
    mas[i]=SensorValue[S1]*(64.0/100);
    wait1Msec(100);
  }
  PlaySound(soundBeepBeep);
  for(int i=1; i<100; i++)
  {
    nxtDrawLine(i-1, mas[i-1], i, mas[i]);
    wait1Msec(100);
  }
  while(true) wait1Msec(1);
}
```

В следующем примере рассматривается запись в массив показаний энкодера мотора, который можно поворачивать (например, рукой) до очередного нажатия оранжевой кнопки NXT. После остановки повторяется последовательность движений. Для этого используется пропорциональный регулятор, алгоритм которого приведено в главе 7. Значения, записанные в массив, подаются в качестве изменяющейся уставки регулятора.

Следует обратить внимание на важность переменной *flag*, которая используется для отключения цикла П-регулятора. Если остановить программу, оставив подобный цикл незавершенным, то регулятор «повисает» в памяти и не позволяет поворачивать мотор рукой даже в плавающем режиме. Ситуация восстанавливается после повторного включения NXT. Далее приведен алгоритм повторения движений мотора:

```
int alpha=0, k=2;
bool flag=true;
task preg()
{
  while(flag) // Регулятор положения alpha для мотора A
  {
    int e=alpha-nMotorEncoder[motorA];
    motor[motorA]=e*k;
    wait1Msec(1);
  }
}
task main()
{
  int mas[1000];
  int i=0;
  while(nNxtButtonPressed!=-1);
  bFloatDuringInactiveMotorPWM=true; // Плавающий режим
  nMotorEncoder[motorA]=0;
  while(nNxtButtonPressed!=3) // Запись положений мотора
  {
    mas[i]=nMotorEncoder[motorA];
    i++;
    wait1Msec(100);
  }
  PlaySound(soundBeepBeep);
  wait1Msec(1000);
  bFloatDuringInactiveMotorPWM=false;
  StartTask(preg);             // Запуск П-регулятора
  for(int j=0; j<i; j++) // Цикл повторения положений
  {
    alpha=mas[j]; // Передача значений через alpha
    wait1Msec(100);
  }
  flag=false;  // Отключение регулятора
  wait1Msec(100);
}
```

Применив приведенный алгоритм к нескольким моторам, можно добиться повторения движения робота по пройденному пути или задать последовательность положений манипулятора. В этой ситуации, очевидно, придется использовать несколько массивов. Помните, что совокупное число элементов не должно превышать 7000. Это связано с ограничением оперативной памяти NXT (всего 64 килобайта).

Одной из частных ошибок при работе с массивами является обращение к несуществующему индексу (ошибка 7). В языке Си элементы нумеруются с нуля, а при объявлении массива в квадратных скобках указывается общее число элементов, а не номер последнего.

Операции с файлами

Операции с файлами в NXT расширяют возможности управления роботом с использованием памяти. Правда, и они ограничены. В контроллере установлено 256 килобайт флеш-памяти, часть из которой занята операционной системой, примерами программ и медиафайлами. Файлы NXT доступны через меню Robot → NXT Brick → File Management. С помощью этого раздела возможен обмен файлами с компьютером, а также удаление ненужных файлов из памяти контроллера.

Особенность хранения файлов в NXT состоит в том, что запись и чтение чисел в большинстве случаев производится по байтам. То есть целое число типа int представляется не как набор от 1 до 5 цифр и минуса, что принято в текстовых файлах в обычных языках программирования, а как всего два символа, несущих 16 бит информации. Такая форма экономит память, но делает файл неудобочитаемым для пользователя. Исключение составляет запись данных типа string.

В остальном операции с файлами в RobotC похожи на аналогичные оперции в других языках. Последовательность операций обращения к файлу такова:

— объявление дескриптора файла,
— открытие файла на чтение/запись,
— операции чтения/записи,
— закрытие файла.

Дескриптор файла представляет собой переменную специфического типа TFileHandle, которая используется в программе для привязки к конкретному файлу. При открытии файла в переменную автоматически заносится значение и устанавливается привязка к файлу, а при закрытии связь освобождается. Очень важно не оставлять открытых связей, иначе потребуется перезагрузка контроллера. Все остальные команды чтения и записи обращаются к файлу через дескриптор.

Есть еще два параметра, через которые передается информация о выполнении команды: 1) nIoResult — результат выполнения операции (не ноль, если ошибка); 2) nFileSize — размер файла в байтах (сообщается при открытии файла на чтение). Длинные имена для этих переменных не обязательны, подойдут любые. При записи важно не превышать указанный размер файла в байтах, а при чтении — строго придерживаться последовательности расположения данных: после каждой операции чтения «указатель» смещается на такое число байт, которое соответствует типу использованной команды чтения.

Кроме того, существуют команды, не требующие объявления дескриптора: удаление и переименование. Перечень основных операций с файлами приведен в табл. 6.2.

Таблица 6.2. Основные операции с файлами в RobotC

Команда	Назначение	Параметры
OpenRead (hFileHandle, nIoResult, sFileName, nFileSize)	Открытие файла на чтение	hFileHandle — дескриптор файла, nIoResult — результат выполнения операции, sFileName — имя файла, nFileSize — размер файла в байтах (сообщается автоматически)
OpenWrite (hFileHandle, nIoResult, sFileName, nFileSize)	Открытие файла на запись	hFileHandle, nIoResult, sFileName см. OpenRead, nFileSize — максимальный размер файла в байтах (задается программистом)
ReadByte (hFileHandle, nIoResult, nParm)	Чтение байта из файла	hFileHandle, nIoResult см. выше, nParm — переменная типа byte
WriteByte (hFileHandle, nIoResult, nParm)	Запись байта в файл	hFileHandle, nIoResult см. выше, nParm — любое значение размером один байт
ReadShort (hFileHandle, nIoResult, nParm)	Чтение целого 16-битного числа из файла	hFileHandle, nIoResult см. выше, nParm — переменная типа int
WriteShort (hFileHandle, nIoResult, nParm)	Запись целого 16-битного числа в файл	hFileHandle, nIoResult см. выше, nParm — числовое значение размером до двух байтов (тип int)
ReadFloat (hFileHandle, nIoResult, fParm)	Чтение дробного числа из файла, 4 байта	hFileHandle, nIoResult см. выше, nParm — переменная типа float
WriteFloat (hFileHandle, nIoResult, fParm)	Запись дробного числа в файл, 4 байта	hFileHandle, nIoResult см. выше, nParm — целое или дробное числовое значение (тип float)
Close (hFileHandle, nIoResult)	Закрытие файла	hFileHandle, nIoResult см. выше
Rename (sFileName, nIoResult, sOriginalFileName)	Переименование файла	sFileName — новое имя файла, nIoResult см. выше, sOriginalFileName — старое имя
Delete (sFileName, nIoResult)	Удаление файла	sFileName, nIoResult см. выше.
nAvailFlash	Количество свободной флеш-памяти	Без параметров. Каждая единица возвращаемого значения соответствует 100 байтам

Остальные команды для работы с файлами можно найти в справочной системе или в библиотеке функций RobotC. Хороший пример находится в папке Sample Programs\NXT\NXT Feature Samples и называется Nxt File IO Test.c.

Рассмотрим пример записи в файл серии значений нажатых кнопок NXT: 3 — оранжевая, 1 — стрелка вправо, 2 — стрелка влево. Запись закончится, когда будет нажата клавиша с номером 0 или превышен лимит числа нажатий:

```
task main()
{
  TFileIOResult res;
  TFileHandle h;
  int size=100, i=0;
  byte k=3;
  Delete ("buttons.dat", res);
  nNxtExitClicks=2; // Программная обработка кнопки выхода
  OpenWrite(h, res, "buttons.dat", size);
  if (res==0) {     // В случае успешного открытия файла
    while((k!=0)&&(i++<100)) { // Цикл записи
      while(nNxtButtonPressed!=-1); // Ждать отпускания
      while(nNxtButtonPressed==-1); // Ждать нажатия
      k=nNxtButtonPressed;
      if (k!=0)
        WriteByte(h, res, k);
    }
    Close(h, res); // Закрытие файла
  }
}
```

Вторая программа считывает из полученнного файла номера нажатых кнопок и выводит на экран первые восемь из них:

```
task main()
{
  TFileIOResult res;
  TFileHandle h;
  int size;
  byte k;
  OpenRead(h, res, "buttons.dat", size);
  if (res==0) {
    for(int i=0; i<size; i++) {
      ReadByte(h, res, k);
      nxtDisplayTextLine(i, "%d", k);
    }
    Close(h, res);
    while(true) wait1Msec(1);
  }
}
```

Глава 7. Алгоритмы управления

Релейный регулятор

Одной из главных задач теории автоматического управления является управление с помощью обратной связи. В таких задачах можно выделить четыре основных компонента [1]:

— управляемую систему (или как говорят специалисты, объект управления) — то, чем мы хотим управлять;

— цель управления — то, чего мы хотим достичь при помощи управления, т.е. желаемое поведение объекта управления;

— список измеряемых переменных (или выходов) — то, что мы можем измерять;

— список управляющих переменных (или входов) — то, что мы можем менять для того, чтобы воздействовать на объект управления.

Еще один важный компонент — регулятор — устройство, вырабатывающее входные величины, необходимые для достижения заданной цели. Этот пятый элемент обычно появляется после того, как теоретическое решение задачи найдено. Под решением проблемы управления будем понимать нахождение закона управления (алгоритма управления), обеспечивающего достижение цели. Как только искомый закон найден, он может быть использован для вычисления управляющих входов по измеренным значениям выходов объекта управления. Полученные значения входов в виде некоторых сигналов подаются на исполнительные устройства.

В формировании этих сигналов может принимать участие микропроцессор, производящий достаточно сложные вычисления в соответствии с заданным алгоритмом.

Будем рассматривать простейший случай, когда задача состоит в поддержании системы в определенном состоянии. Тогда можно говорить о задаче регулирования как частном случае задачи автоматического управления. Для осуществления автоматического регулирования к объекту подключается регулятор. Под регулятором будем понимать устройство, которое с помощью чувствительного элемента (датчика) измеряет регулируемую величину и в соответствии с законом регулирования вырабатывает воздействие на регулирующий орган объекта. Система, состоящая из объекта и регулятора, называется системой управления [1].

Исполнительное устройство, осуществляющее механическое перемещение регулирующего органа, обычно называется сервоприводом.

Релейным двухпозиционным регулятором называется регулятор, у которого регулирующий орган под действием сигнала от датчика может принимать одно из двух крайних положений: «открыт» или «закрыт». При этом управляющее воздействие на регулируемый объект может быть только максимальным или минимальным.

Управление мотором

«Робот поднимает меч и бросается на противника!» Здорово. Как в сказке. А что если меч наткнется на что-нибудь и потеряет свое заданное положение? Нехорошо бросаться на противника с опущенным мечом.

Для примера прикрепите к мотору длинную балку с помощью сдвоенного трехмодульного штифта. Это будет меч. По сценарию, в момент запуска его надо поднять на 45 градусов и удерживать в этом положении, что бы ни случилось (рис. 7.1).

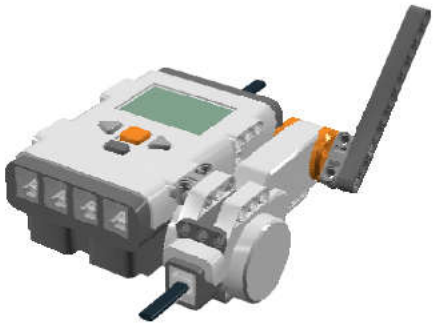


Рис. 7.1. Стабилизация мотора в положении 45 градусов.

Регулируемой величиной будет угол поворота мотора, определяемый через показания энкодера, регулирующим органом — сам мотор.

Алгоритм таков: обнулить показания датчика оборотов, задать желаемое положение в 45 градусов и в цикле поддерживать это положение, учитывая возможные отклонения. Если показания энкодера превышают 45, мотор вращается назад. В противном случае вращается вперед. Задержка в 1 мс предназначена для разгрузки контроллера.

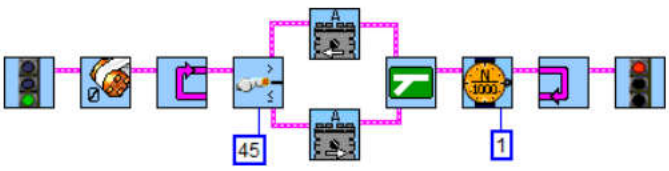


Рис. 7.2. Релейное управление одним мотором.

```
task main()
{
  int alpha=45;
  nMotorEncoder[motorA]=0;
  while(true)
  {
    if(nMotorEncoder[motorA]>alpha)
      motor[motorA]=-100;
    else
      motor[motorA]=100;
    wait1Msec(1);
  }
}
```

Что же мы увидим? Робот ведет себя как новобранец с мечом. Мотор удерживает балку, но как-то неуверенно: происходят постоянные колебания. Уменьшить их можно, разве что понизив мощность мотора. Попробуйте сделать это.

Особенность релейного регулятора в том, что он в принципе не может стабилизироваться в нужном положении и вызывает колебания с той или иной амплитудой. Как этого избежать, описано чуть дальше.

Движение с одним датчиком освещенности

Рассмотрим пример трехколесного Lego-робота с одним датчиком освещенности, который должен двигаться по плоской поверхности вдоль границы черного и белого (рис. 7.3).

Рис. 7.3. Движение вдоль границы черного и белого.

Исполнительным органом объекта в данном случае будут два колеса, подключенные к сервоприводам. Регулируемая величина — положение датчика света на границе черного и белого, зависящее от уровня освещенности под ним. Возмущающее воздействие — это движение робота вперед, которое приводит к отклонению от границы. Поскольку линия может быть кривой, а также вследствие других факторов при отсутствии регулирующего воздействия робот непременно съедет с линии.

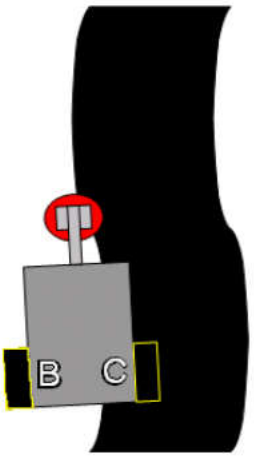


Рис. 7.4. Расположение моторов и датчика освещенности.

Подключим левый мотор на порт B, правый — на порт C (рис. 7.4). Стартовая позиция робота — датчик на белом. Построим программный регулятор, который обеспечит движение по дуге в сторону черного, пока робот на белом, и движение по дуге в сторону белого, пока робот на черном. Для этого выключается или резко понижается мощность одного из моторов. Вот реализация на RobotC:

```
task main()
{
  int grey=45;
  while (true)
  { // grey - значение серого
    if (SensorValue[S1]>grey)
    {
      motor[motorB]=100;
      motor[motorC]=0;
    }
    else
    {
      motor[motorB]=0;
      motor[motorC]=100;
    }
    wait1Msec(1);
  }
}
```

Значение «серого» (grey) может быть константой для данной системы или вырабатываться в результате предварительной калибровки как среднее арифметическое черного и белого.

Под управлением такого регулятора робот движется вдоль линии границы по ломаной кривой, периодически наезжая то на черное, то на белое. Скорость его невысока, стабильность тоже, а траектория движе-

ния оставляет желать лучшего. Все это издержки использования двухпозиционного релейного регулятора.

Программа для движения по линии с помощью релейного регулятора в Robolab показана на рис. 7.5.

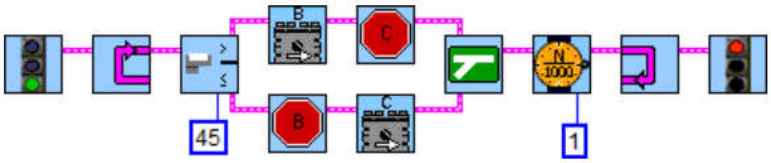


Рис. 7.5. Алгоритм движения вдоль границы черного и белого на релейном регуляторе.

Движение с двумя датчиками освещенности

Правильно проехать перекресток с одним датчиком освещенности довольно сложно. Если требуется сделать это с достаточно высокой скоростью, нужны хотя бы два датчика, поставленные на расстоянии в две-три ширины линии (или шире). Для начала используем релейный регулятор, с помощью которого можно обработать четыре возможных состояния датчиков (рис. 7.6):

— оба на белом — движение прямо;

— левый (S1) на черном, правый (S2) на белом — движение налево;

— левый на белом, правый на черном — движение направо;

— оба на черном — движение прямо.

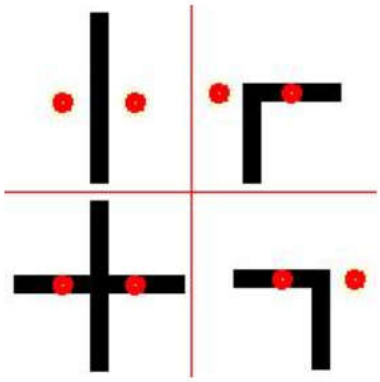


Рис. 7.6. Варианты расположения двух датчиков освещенности над черной линией.

Этот регулятор реализуется программно с помощью вложенных ветвлений:

```
task main()
{
  int grey1=45, grey2=45;
  while(true) {
    if(SensorValue[S1]>grey1) {
      if(SensorValue[S2]>grey2) { // Оба на белом
        motor[motorB]=100;
        motor[motorC]=100;
```

```
      }
      else {  // Правый на черном
        motor[motorB]=100;
        motor[motorC]=-100;
      }
    }
    else {
      if(SensorValue[S2]>grey2) {    // Левый на черном
        motor[motorB]=-100;
        motor[motorC]=100;
      }
      else {  // Оба на черном
        motor[motorB]=100;
        motor[motorC]=100;
      }
    }
  wait1Msec(1);
  }
}
```

То же самое в Robolab (рис. 7.7):

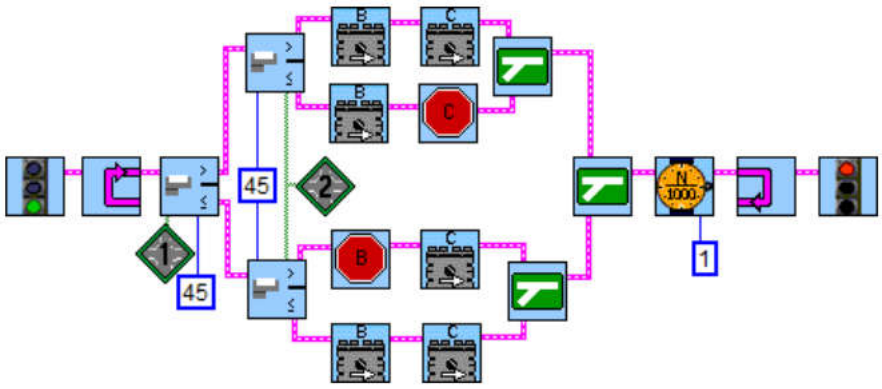


Рис. 7.7. Алгоритм движения вдоль черной линии с двумя датчиками на релейном регуляторе.

Вот такой, довольно длинный и малоэффективный алгоритм автор и многие его коллеги по состязаниям Lego-роботов использовали для решения классической задачи движения по траектории. Чтобы не сбиться с линии, приходилось значительно усложнять его, вводить дополнительные проверки и, конечно, калибровку датчиков под конкретную освещенность. Однако достаточно было в окнах появиться солнышку, как робот терял линию, поскольку значения были жестко фиксированными. Так продолжалось до тех пор, пока мы не познакомились с базовыми принципами автоматического регулирования.

Пропорциональный регулятор

Описание

При автоматическом регулировании управляющее воздействие *u(t)* обычно является функцией динамической ошибки — отклонения *e(t)* регулируемой величины *x(t)* от ее заданного значения *x0(t)*:

$$e(t) = x0(t) - x(t).$$

Это принцип Ползунова-Уатта регулирования по отклонению, или принцип обратной связи. Математическое выражение функциональной зависимости желаемого управляющего воздействия *u0(t)* от измеряемых регулятором величин называется законом или алгоритмом регулирования, о котором говорилось выше.

Пропорциональный регулятор — это устройство, оказывающее управляющее воздействие на объект пропорционально его отклонению от заданного состояния:

$$u0(t) = ke.$$

Здесь *k* — это коэффициент усиления регулятора.

Заданное состояние *x0* принято называть *уставкой*, а отклонение от него *e* — *невязкой*. Далее для определенности будем обозначать невязку сокращением *err* (от английского слова «error» — ошибка).

Управление мотором

Опытный воин не станет размахивать мечом, как это делает робот на релейном регуляторе. Надо придумать алгоритм, который задержит мотор, удерживающий меч, в строго фиксированном положении (рис. 7.1). В этом поможет П-регулятор.

Пусть *e*1 — показания датчика оборотов[1] на моторе A — является регулируемой величиной. Уставка *x0* = 45, а невязка *e* = 45 – *e*1. Тогда управляющее воздействие на мотор задается формулой

$$u = k \cdot (45 - e1).$$

Здесь k — коэффициент усиления, например 5, который позволит усилить реакцию мотора даже при небольших отклонениях от уставки.

[1] Не стоит путать математическое обозначение невязки *e* (от error) с показаниями энкодера *e*1 (от encoder), предопределенной переменной среды Robolab.

При отклонении в положительную сторону на мотор будет подаваться отрицательное управляющее воздействие, и наоборот. Это управление можно применять к мотору в цикле с небольшой задержкой в 1-10 мс, чтобы разгрузить контроллер (рис. 7.8).

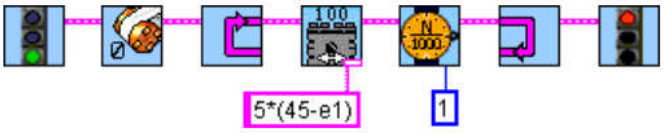


Рис. 7.8. Алгоритм управления мотором на пропорциональном регуляторе.

Если усиливающий коэффициент увеличить с 5 до 100, наш пропорциональный регулятор начнет работать как релейный, вызывая сильные колебания из-за возникновения эффекта перерегулирования.

В языке RobotC нет такого удобного обозначения показаний энкодера как в Robolab, поэтому программа выглядит немного длиннее:

```
task main()
{
  int k=5, u;
  nMotorEncoder[motorA]=0;
  while(true)
  {
    u=k*(45-nMotorEncoder[motorA]);
    motor[motorA]=u;
    wait1Msec(1);
  }
}
```

Далее, чтобы нанести «удар мечом», достаточно, имея вместо числа 45 переменную, изменить ее значение извне, например, из параллельной задачи. Об этом написано в разделе, посвященном роботам-барабанщикам в главе 8.

А сейчас построим регулятор, управляющий не только статическим положением мотора, но и скоростью его движения. Следуя логике алгоритма, уставка, которая до сих пор была константой и не менялась, должна начать движение в сторону увеличения или уменьшения. Повинуясь регулятору, мотор неизбежно будет следовать за ней. Самый простой инструмент для постоянного приращения значения уставки — это таймер.

В контроллере NXT есть четыре встроенных таймера, каждый из которых может отмерять время в десятых, сотых и тысячных долях секунды. Освоим первый таймер, который за секунду совершает 10 «тиков». В Robolab он обозначается T1 или Timer100ms1, а в RobotC — timer100[T1].

Угол *alpha* отклонения мотора, заданный в предыдущем примере значением 45, поставим в зависимость от показаний таймера с ускоряющим коэффициентом $k2$:

$$alpha = k2 \cdot T1.$$

Управляющее воздействие останется прежним с усиливающим коэффициентом $k1$:

$$u = k1 \cdot (alpha - e1).$$

Кратко в программе на языке Robolab управляющее воздействие подадим сразу на мотор, предварительно инициализировав таймер (рис. 7.9).

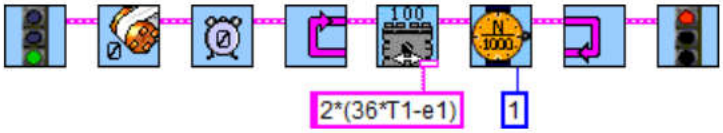


Рис. 7.9. Управление скоростью мотора — один оборот в секунду.

Коэффициент $k2 = 36$ определяет, что за секунду значение *alpha* набегает до 360, что соответствует одному полному обороту двигателя:

```
task main()
{
  int k1=2, k2=36, u, alpha;
  nMotorEncoder[motorA]=0;
  ClearTimer(T1);
  while(true)
  {
    alpha=timer100[T1]*k2;
    u=k1*(alpha-nMotorEncoder[motorA]);
    motor[motorA]=u;
    wait1Msec(1);
  }
}
```

Используя целочисленное деление, принятое в языке C (и в Robolab) для переменных целого типа, можно достичь дискретного изменения угла, т.е. приращения его раз в секунду:

$$alpha = T1 / 10 \cdot k2.$$

При коэффициенте $k2 = 60$ перемещения балки будут соответствовать движению секундной стрелки на циферблате часов. Но это мало

заметно. Для наглядности можно задать *k2* = 30, тогда стрелка сделает полный оборот за 12 «тиков» по 30 градусов каждый. Будьте внимательны с последовательностью операций целочисленного деления и умножения, при изменении их порядка или «сокращении» непременно изменится результат (рис. 7.10).

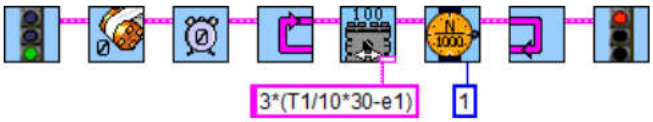


Рис. 7.10. Ускоренная имитация движения стрелки часов.

И, наконец, пример математического барабанщика. Вместо постоянного движения вперед стрелка будет совершать колебания вперед-назад под управлением П-регулятора. В этом поможет операция деления с остатком, которая в языке С обозначается знаком %. Остаток от деления неотрицательного целого числа на 2 всегда будет 0 или 1:

$$alpha = T1 \ \% \ 2 \cdot k2.$$

Усилив отклонение в *k2*=15 раз, получим колеблющуюся уставку *alpha*, что вынудит регулятор 5 раз в секунду перемещать мотор то в 0°, то в 15 градусов. Изменения в программе невелики. Рассмотрим пример на RobotC:

```
task main()
{
  int k1=3, k2=15, u, alpha;
  nMotorEncoder[motorA]=0;
  ClearTimer(T1);
  while(true)
  {
    alpha=timer100[T1]%2*k2;
    u=k1*(alpha-nMotorEncoder[motorA]);
    motor[motorA]=u;
    wait1Msec(1);
  }
}
```

Этот прототип барабанщика наносит удары по столу через одинаковые промежутки времени. Главное, стартовать в нужной позиции. Используя целочисленную математику, можно задать и более сложный ритмический рисунок, например (табл. 7.1):

$$alpha = T1 \ \% \ 5 \ \% \ 2 \cdot k2.$$

В остатках от деления на 5 получаем цепочку «0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, …». Применяя деление на 2, получаем цепочку остатков «0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, …». Таким образом, каждая вторая пауза имеет удвоенную длительность, что и задает рисунок ритма (табл. 7.1). Надеюсь, читатель сможет придумать свой собственный зажигательный ритм.

Таблица 7.1. Задание последовательности ударов и пауз

T1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
T1 % 5	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
T1 % 5 % 2	0	1	0	1	**0**	**0**	1	0	1	**0**	**0**	1	0	1	0

Мы выяснили, что изменяя уставку, можно вынудить регулятор не стоять на месте, а совершать порой довольно замысловатые движения. Этот принцип может быть положен в основу управления роботом-манипулятором или, например, конечностями робота, имитирующего живое существо.

Напоследок предлагаем читателю самостоятельно построить барабанщика с двумя моторами, каждый из который будет играть свою партию. Разница в алгоритме будет заключаться в добавлении двух команд: инициализации энкодера B и управления мотором B с аналогичной зависимостью от таймера T1.

Синхронизация моторов

Довольно часто мы наблюдаем, как тележка, которой дана команда двигаться прямо, медленно съезжает с заданного курса. Причин может быть множество: и трение на осях или шестернях, и цепляющаяся за корпус шина, и капризы моторов.

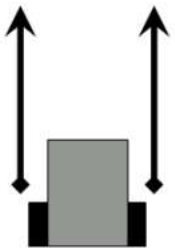


Рис. 7.11. Задача двигаться прямолинейно.

Однако можно построить регулятор, который обеспечит синхронное движение моторов с точностью до 1 градуса. В принципе, практически во всех средах есть соответствующие управляющие команды высокого уровня. Давайте разберемся в общих чертах, как они устроены.

Итак, задача — двигаться прямолинейно (рис. 7.11). Пусть e2 и e3 — показания датчиков оборотов моторов B и C. Их надо будет обнулить перед началом движения (рис. 7.12). Управляющее воздействие определим следующим образом:

$$u = k \cdot (e3 - e2).$$

Управление моторами уже знакомо из предыдущих примеров:

```
motor[motorB]=v+u;
motor[motorC]=v-u;
```

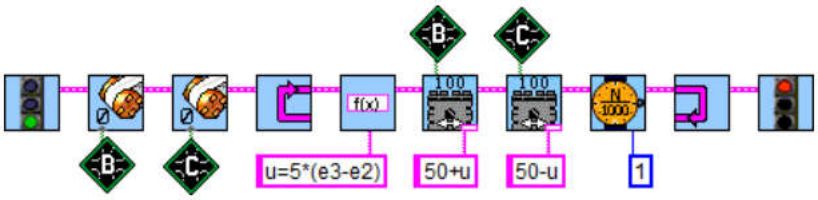


Рис. 7.12. Алгоритм синхронизации моторов для прямолинейного движения с использованием П-регулятора.

```
task main()
{
  int k=5, v=50, u;
  nMotorEncoder[motorB]=0;
  nMotorEncoder[motorC]=0;
  while(true)
  {
    u=k*(nMotorEncoder[motorC]-nMotorEncoder[motorB]);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

Если мотор B обгоняет мотор C, то управляющее воздействие станет отрицательным и мотор B начнет притормаживать, а C — ускоряться. И наоборот. При этом можно указать и максимальную мощность, поскольку, как правило, отклонения в этом случае довольно малы и достаточно ограничиться понижением мощности обгоняющего мотора.

Взять азимут

Если читателю удалось обзавестись компасом от компании Hitechnic, стоит попробовать и его использовать вместе с П-регулятором.

Задачка проста. Запомнить азимут и ехать в заданном направлении, никуда не сворачивая. Пусть *alpha* — азимут, т.е. угол, полученный с компаса на момент старта, а *s*1 — показания датчика компаса в каждый момент времени.

Попробуйте руками повернуть робота, который едет по командам этого алгоритма (рис. 7.13). Он окажется на удивление настырным:

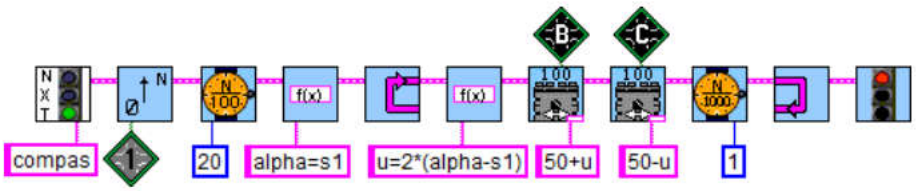


Рис. 7.13. Алгоритм движения по азимуту с использованием П-регулятора.

```
task main()
{
  float u, k=2;
  int alpha=SensorValue[s1];
  while (true)
  {
    u=k*(alpha-SensorValue[s1]);
    motor[motorB]=50+u;
    motor[motorC]=50-u;
    wait1Msec(1);
  }
}
```

Напомним, что для надежности работы компас должен быть отодвинут от моторов и NXT не менее чем на 15 см. Кроме того, желательно, чтобы поблизости не было сильных магнитных полей.

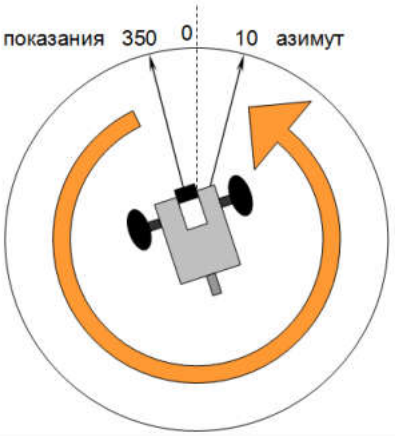


Рис. 7.14. Разворот по компасу в неправильном направлении.

В силу особенностей работы П-регулятора и компаса при азимуте, близком к направлению на север, робот будет делать развороты в сторону, противоположную той, которая представляется разумной (рис. 7.14). Дело в том, что компас всегда вырабатывает неотрицательное значение, а отклонение может иметь любой знак. Так, например, при азимуте *alpha* = 10 и текущем направлении *S*1 = 350 отклонение будет равно *err* = 10 – 350= –340. При том, что оба значения были близки к нулю и находились на расстоянии всего 20 градусов, от регулятора поступит команда сделать оборот на –340 градусов. Очевидно, что решить эту поблему можно было бы добавлением периода, т. е. –340+360=20 градусов. Однако в другом случае период придется вычитать. Обобщая проблему, можно сказать, что при отклонении, превышающем по модулю 180 градусов, требуется приведение его к значению от –180 до 180 градусов. В этом опять поможет целочисленная математика.

Поскольку показания компаса $S1$ лежат в промежутке 0...359, отклонение $err = alpha - S1$ умещается в промежуток –359...359. Деление отклонения на 180 без остатка даст значение 0, 1 или –1:

$$err/180 = \begin{cases} 1, \text{ при } err \geq 180 \\ 0, \text{ при } -180 < err < 180 \\ -1, \text{ при } err \leq -180 \end{cases}.$$

Это дает возможность вычесть период с нужным знаком при достижении и превышении модулем невязки err значения 180. Получаем новую формулу для отклонения:

$$err\_new = err - err / 180 \cdot 360.$$

В общем виде управляющее воздействие будет записано следующим образом:

$$u=k \cdot (alpha - S1 - (alpha - S1) / 180 \cdot 360).$$

Программа для Robolab изображена на рис. 7.15:

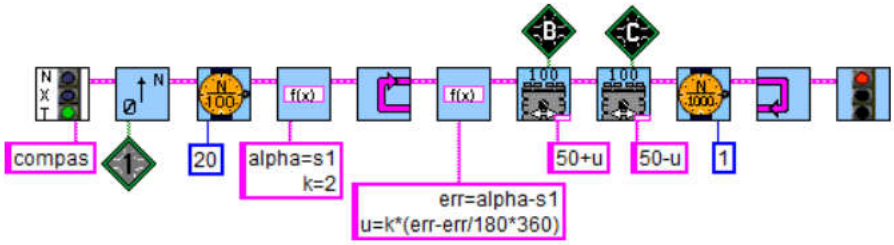


Рис. 7.15. Алгоритм движении по компасу с ограничением диапазона отклонения.

```
task main()
{
  float u, k=3;
  int err, alpha=SensorValue[s1];
  while (true) {
    err=alpha-SensorValue[s1];
    u=k*(err-err / 180*360);
    motor[motorB]=50+u;
    motor[motorC]=50-u;
    wait1Msec(1);
  }
}
```

Теперь робот при любых отклонениях будет выравниваться в нужном направлении, если только его не собьют с пути внешние магнитные

излучения. Полученные результаты можно использовать как в навигации, так и ряде увлекательных соревнований, таких как кегельринг, футбол, теннис и др.

Следование за инфракрасным мячом

Вместе с компасом компания Hitechnic выпустила набор устройств (рис. 7.16), совместимых с NXT, с помощью которых можно организовать настоящую игру в футбол, что и происходит уже много лет в лиге юниоров турнира Robocup, а с 2010 г. в рамках World Robot Olympiad.

Инфракрасный мяч (IRBall) обладает 20 излучателями и выполнен из прочного прозрачного пластика. Инфракрасный поисковик (IR-Seeker[1]) позволяет определить направление на мяч в одном из 9 секторов с точностью до 36 градусов (нулевой сектор означает, что мяч вне поля видимости), а также интенсивность его излучения.

Рис. 7.16. Инфракрасный мяч (слева), компас (в центре сверху), инфракрасный поисковик (в центре снизу) и его секторы обзора (справа).

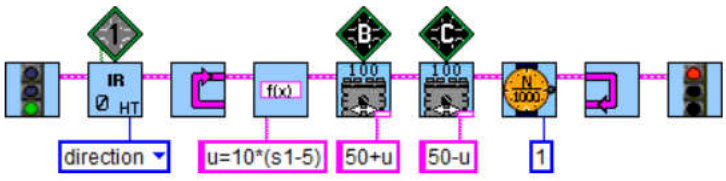


Рис. 7.17. Движение за инфракрасным мячиком с помощью датчика IRSeeker.

Построим алгоритм на П-регуляторе, по которому робот будет следовать за мячиком (рис. 7.17). В качестве уставки используем число 5 — номер центрального сектора. Инициализируем датчик в режиме direction для определения сектора.

[1] Поддержка в Robolab новых датчиков IRSeeker V2 появилась в 2010 г. с очередным обновлением, ссылка приведена в приложении 3.

Следующей задачей объединим движение по азимуту и поиск инфракрасного мяча. Поскольку IRSeeker может измерять интенсивность излучения, установим порог 160, который будет означать, что датчик находится непосредственно перед роботом. В этом случае робот едет по компасу. Если мяч далеко, робот едет за ним. И так в бесконечном цикле. Чтобы для переключения режимов работы датчика каждый раз не инициализировать его заново, используем предопределенную переменную *dir*, которая соответствует текущему номеру сектора (рис. 7.18).

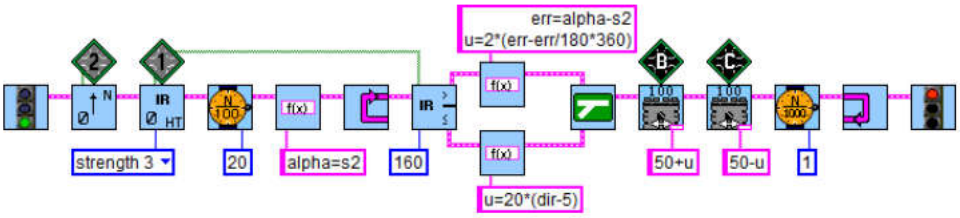


Рис. 7.18. Ведение мяча по азимуту.

Используя этот простой алгоритм, можно организовать увлекательные соревнования «Пенальти», в которых робот на пустом поле с разных позиций должен завести мяч в ворота. Полноценный футбол[1] потребует, конечно, более сложного программирования. Однако П-регулятор может быть основой для любого алгоритма движения.

Движение по линии

Поначалу это казалось странным, но движение по границе черного и белого тоже можно построить на П-регуляторе. Хотя внешне задача представляется решаемой только с помощью релейного регулятора, поскольку в системе присутствует всего два видимых человеческому глазу состояния: черное и белое. Но робот все видит иначе, для него отсутствует резкая граница между этими цветами (рис. 7.19). Можно сказать, он близорук и видит градиентный переход оттенков серого. Причина в том, что датчик освещенности улавливает отраженный свет всего одним фотоэлементом, поэтому наличие в пятне от фонарика-светодиода сегмента черного

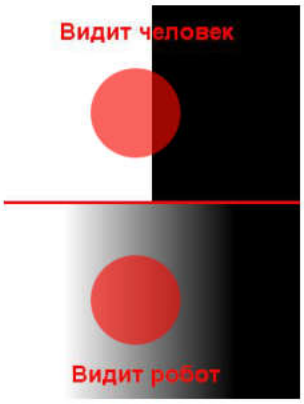


Рис. 7.19. Разница в восприятии человека и робота.

[1] Правила всероссийских соревнований по версии Robocup Junior Gen II находятся по адресу: http://wroboto.ru/rules/football/.

поля просто понижает совокупную освещенность. Вот это нам и поможет построить П-регулятор.

Так же как и в релейном регуляторе, необходимо определить среднее значение между черным и белым, обозначим его grey. Это будет то состояние датчика освещенности s1, к которому должна стремиться система.

Коэффициент k может быть достаточно мал (от 1 до 3) для маневренного робота.

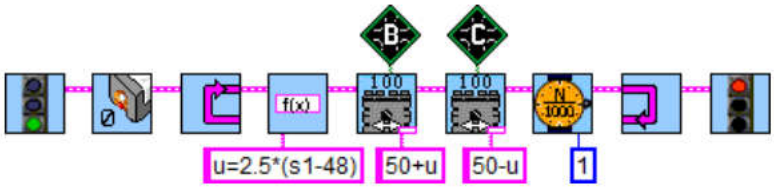


Рис. 7.20. Алгоритм движения вдоль черной линии, основанный на П-регуляторе.

```
task main()
{
  float u, k=2.5;
  int grey=48;
  while(true)
  {
    u=k*(SensorValue[s1]-grey);
    motor[motorB]=50+u;
    motor[motorC]=50-u;
    wait1Msec(1);
  }
}
```

Такой регулятор (рис. 7.20) эффективно работает только для малых углов отклонения, поэтому робота надо ставить в направлении движения так, чтобы датчик оказался по левую сторону от черной линии. Если датчик заедет на черную линию, управляющее воздействие станет отрицательным и левый мотор будет вращаться медленнее правого, что выровняет робота. Нетрудно заметить, что движение по линии на П-регуляторе отличается плавностью и на некоторых участках робот движется практически прямолинейно или точно повторяя изгибы линии.

В палитре Behaviors среды Robolab 2.9 есть пример использования П-регулятора для движения с одним датчиком вдоль черной линии, который называется Follow Line. Этот пример адаптирован для RCX, но нетрудно его переделать и под NXT.

Движение по линии с двумя датчиками

Этот регулятор интересен тем, что на него практически не влияют колебания освещенности в помещении, которые обычно доставляют немало проблем начинающим робототехникам.

Как же использовать то, что мы имеем? Давайте размышлять. Релейный регулятор на двух датчиках работает известным образом:

— при двух белых едет прямо,
— при одном черном поворачивает в сторону черного,
— при двух черных едет прямо.

Из первого и третьего утверждений можно заметить, что при равных показаниях датчиков робот едет прямо. Учитывая наши знания о зрении робота, можно утверждать, что при любых равных значениях показаний датчиков (в том числе и на сером) робот должен ехать прямо. Предположим, что наши датчики s1 и s2 откалиброваны абсолютно одинаково, и построим регулятор для идеальной системы, который будет равен нулю при равных показаниях:

```
u=k*(SensorValue[s1]-SensorValue[s2]);
motor[motorB]=50+u;
motor[motorC]=50-u;
```

Если на левом датчике s1 потемнеет, управление *u* станет отрицательным, мотор B станет медленнее, чем C, и робот начнет подруливать влево. И наоборот. Все сходится.

Теперь спустимся с небес на землю. Может оказаться, что датчики откалиброваны по-разному. Для этого следует приспособиться под их показания, т. е. произвести автокалибровку перед запуском. И сравниваться в регуляторе будут не абсолютные показания, а их отклонения.

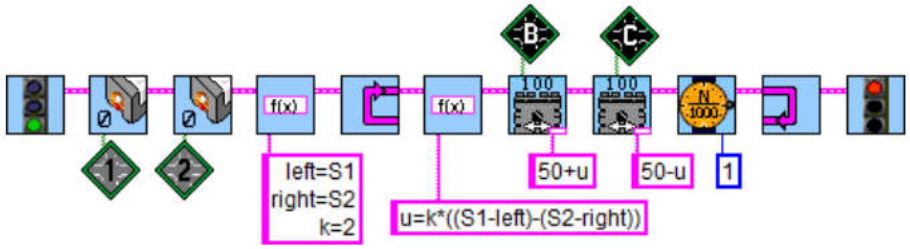


Рис. 7.21. Алгоритм движения по линии с двумя датчиками, основанный на П-регуляторе.

```
task main()
{
  int left=SensorValue[s1];
  int right=SensorValue[s2];
  float k=2, u;
```

```
  while(true)
  {
    u=k*((SensorValue[s1]-left)-(SensorValue[s2]-right));
    motor[motorB]=50+u;
     motor[motorC]=50-u;
    wait1Msec(1);
  }
}
```

Коэффициент k может изменяться в достаточно широком диапазоне (1—20 и более) в зависимости от кривизны линии, маневренности робота и разницы между черным и белым на поле (рис. 7.21).

Важное условие. Автокалибровка должна производиться на одноцветной поверхности и желательно при той освещенности, которая будет занимать наибольшую часть пути. Например, если робот едет вдоль черной линии на белом поле, то калиброваться надо на белом (рис. 7.22).

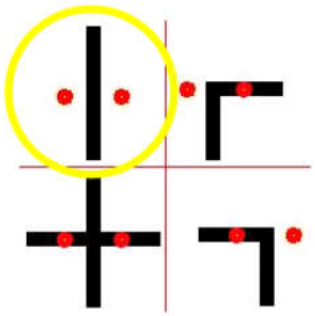


Рис. 7.22. Положение для старта робота с предварительной калибровкой.

И еще замечание. Встречаются датчики, показания которых расходятся на 10—20 %. Желательно их не ставить в паре на регулятор с большим коэффициентом, поскольку при резком изменении общей освещенности даже на однотонном белом поле отклонения могут оказаться различны, что приведет к неожиданным последствиям.

Движение вдоль стенки

Решим такую задачу. Робот должен двигаться вдоль стенки на заданном расстоянии L (рис. 7.23). Предположим, что левое колесо робота управляется мотором B, правое — мотором C, а датчик расстояния, подключенный к порту 1, закреплен несколько впереди корпуса тележки (это важно!) и направлен на стенку справа по ходу движения (рис. 7.27—7.29).

Расстояние до стенки, которое показывает датчик в настоящий момент времени, обозначим S1.

Моторы двигаются с средней скоростью v=50, но при отклонении от заданного курса на них подается управляющее воздействие u. Снова обозначим это следующим образом:

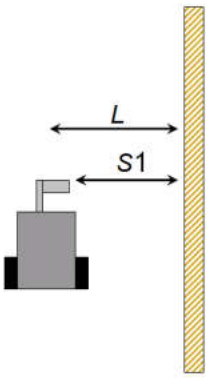


Рис. 7.23. Задача движения вдоль стены на расстоянии *L*.

```
motor[motorB]=v+u;
motor[motorC]=v-u;
```

Осталось определить, чему будет равно управляющее воздействие. Это нетрудно:

```
u=k*(SensorValue[s1]-L);
```

Таким образом, при $s1 = L$ робот не меняет курса и едет прямо. В случае отклонения его курс корректируется. Здесь k — это некоторый усиливающий коэффициент, определяющий воздействие регулятора на систему. Для робота NXT средних размеров коэффициент k может колебаться от 1 до 10 в зависимости от многих факторов. Предлагаем подобрать его самостоятельно (рис. 7.24).

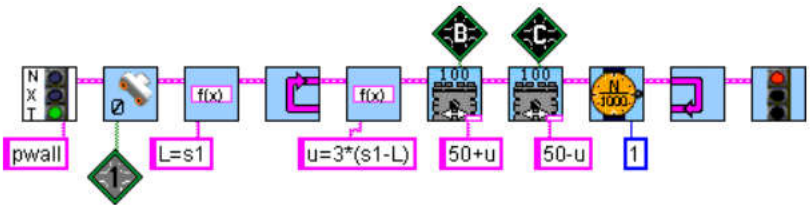


Рис. 7.24. Алгоритм движения вдоль стенки на основе П-регулятора.

Та же программа на языке RobotC. Не забудьте объявить датчик ультразвука (Sonar) через меню Robot → Motor and Sensors Setup. Желательно задать датчику уникальное имя, которое будет использоваться вместо $S1$.

```
task main()
{
  float u, k=3, v=50;
  int L=SensorValue[S1];
  while(true) {
     u=k*(SensorValue[S1]-L);
     motor[motorB]=v+u;
     motor[motorC]=v-u;
     wait1Msec(1);
  }
}
```

В данном случае регулятор будет эффективно работать только при малых углах отклонения. Кроме того, движение практически всегда будет происходить по волнообразной траектории. Сделать регулирование более точным позволит введение новых принципов, учитывающих отклонение робота от курса.

Пропорционально-дифференциальный регулятор

Движение вдоль стенки на ПД-регуляторе

В некоторых ситуациях П-регулятор может вывести систему из устойчивого состояния (рис. 7.25 слева). Например, если робот направлен от стенки, но находится по отношению к ней ближе заданного расстояния, на моторы поступит команда еще сильнее повернуть от стенки, в результате чего с ней может быть утерян контакт (вспомните, что датчик расстояния получает отраженный сигнал практически только от перпендикулярной поверхности).

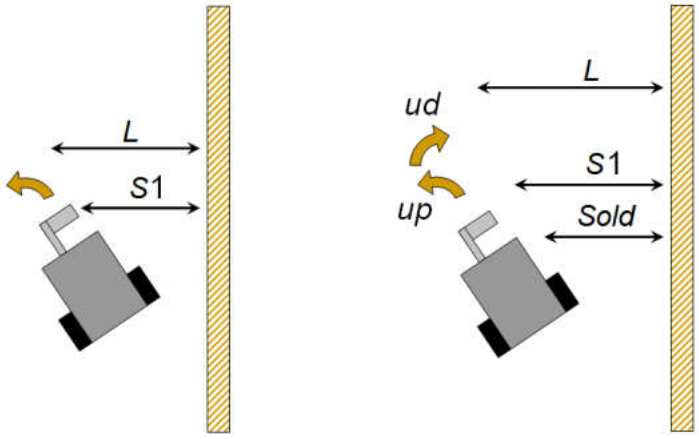


Рис. 7.25. Проблема пропорционального регулятора — потеря контакта со стенкой (слева). Необходима дифференциальная составляющая (справа).

Для защиты от подобных ситуаций добавим в регулятор дифференциальную составляющую, которая будет следить за направлением движения робота (рис. 7.25, справа). Иными словами, значение скорости будет влиять на управляющее воздействие. Известно, что скорость находится как $v = \Delta s / \Delta t$, где Δs — это изменение расстояния за промежуток времени Δt. Определим дифференциальную составляющую через скорость отклонения робота от заданного положения:

$$ud = k \cdot (S1 - Sold) / \Delta t,$$

где $S1$ — текущее расстояние до стенки, $Sold$ — расстояние на предыдущем шаге.

Поскольку замеры производятся через равные промежутки времени, то Δt можно принять за константу, взяв $k2 = k \cdot \Delta t$:

$$ud = k2 \cdot (S1 - Sold).$$

Таким образом, ПД-регулятор описывается формулой из двух слагаемых

$$u = up + ud = k1 \cdot (S1 - L) + k2 \cdot (S1 - Sold)$$

Можно доказать математически, что для устойчивого достижения цели коэффициент $k2$ при дифференциальной составляющей должен превышать $k1$.

Алгоритм движения вдоль стенки на ПД-регуляторе в целом будет выглядеть так (рис. 7.26):

```
task main()
{
  float u, k1=2, k2=10, v=50;
  int Sold, L;
  Sold=L=SensorValue[S1];
  while(true)
  {
    u= k1*(SensorValue[S1]-L) + k2*(SensorValue[S1]-Sold);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    Sold=SensorValue[S1];
    wait1Msec(1);
  }
}
```

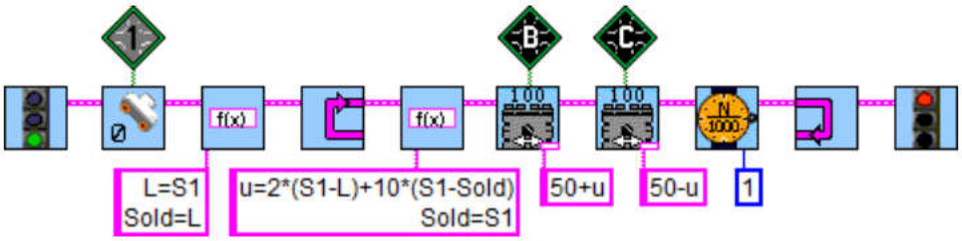


Рис. 7.26. Алгоритм движения вдоль стенки, основанный на ПД-регуляторе.

Как было указано, датчик нужно разместить несколько спереди робота и как можно дальше от стены. Один из вариантов конструкции приведен на рис. 7.27—7.29. При горизонтальном расположении датчика часть сигналов может теряться в связи с тем, что робот движется, а расстояние между ультразвуковыми «глазками» приемника и получателя тоже играет роль.

Рис. 7.27. Датчик расстояния слегка выносится вперед.

Рис. 7.28. Ориентацию датчика можно варьировать.

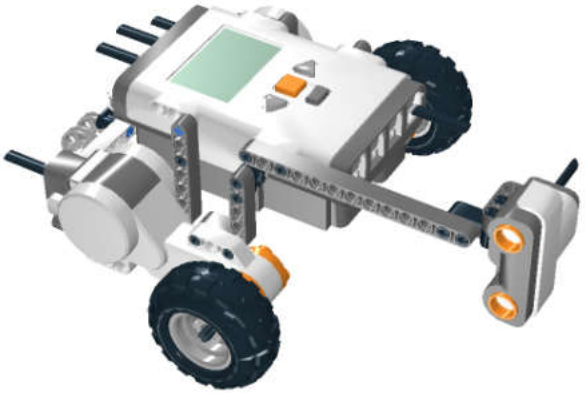


Рис. 7.29. Оптимальный вариант — вертикальное расположение датчика.

Движение по линии

Как ни странно, задача движения по линии оптимальным образом решается именно с помощью ПД-регулятора. Однако эффект от него проявляется только на больших скоростях, когда робот начинает сбиваться с линии. Чтобы добиться такой скорости, необходимо увеличить диаметр колес и даже поставить повышающую передачу. Если обычная тележка из Lego за секунду преодолевает около 40 см, то самый быстрый Lego-робот на линии может достигать скорости 1 м/с.

Для случая с одним датчиком задача решается аналогично предыдущей. В примере на рис. 7.30 в качестве среднего значения серого взято число 48. Базовая скорость моторов заведомо установлена на значение 80. Пропорциональный коэффициент $k1 = 3$, дифференциальный коэффициент $k2 = 10$:

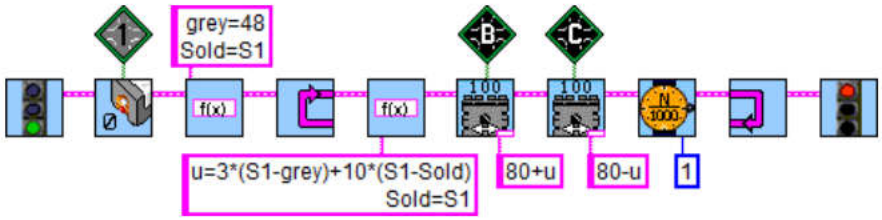


Рис. 7.30. Движение по линии на ПД-регуляторе с одним датчиком света.

```
task main()
{
  float u, k1=3, k2=10, grey=48, v=80;
  int Sold=SensorValue[S1];
  while(true) {
    u=k1*(SensorValue[S1]-grey)+k2*(SensorValue[S1]-Sold);
    Sold=SensorValue[S1];
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

Для нового робота некоторое время займет подбор подходящих коэффициентов. Вообще говоря, точный расчет коэффициентов регулятора — это сложная инженерная задача, но мы можем получить некоторые результаты опытным путем. Первым делом добьемся стабилизации движения при нулевом дифференциальном коэффициенте. Когда робот начинает ехать по линии на одном П-регуляторе, теряя ее только на крутых поворотах, постепенно увеличиваем дифференциальную составляющую. Ее действие будет выражаться в резком сдерживании робота при попытке отклониться от курса. В идеале робот должен двигаться по линии как по рельсам.

При построении ПД-реглятора для двух датчиков в качестве предыдущего значения можно рассматривать общее отклонение *err*:

$$err = (s1 - left) - (s2 - right),$$
$$u = k1 \cdot err + k2 \cdot (err - errold),$$
$$errold = err.$$

Кубическая составляющая

Глядя на результаты применения регуляторов при движении по линии с двумя датчиками, можно заметить, что робот показывает лучшее время, если на прямолинейных участках движется с малыми отклонениями, а на изгибах поворачивает резко (рис. 7.32). Такого эффекта можно достичь, используя вместе с пропорциональной кубическую составляющую с малым коэффициентом:

$$err = (s1 - left) - (s2 - right),$$
$$up = k1 \cdot err,$$
$$uk = k2 \cdot err \cdot err \cdot err, \text{ где } k2 < 0.05,$$
$$u = up + uk.$$

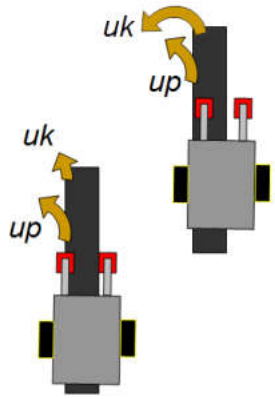


Рис. 7.32. Принцип действия кубического регулятора.

Эта идея была предложена коллегами из Железногорского филиала СФУ и прижилась среди юных робототехников: многим пришелся по душе «кубический регулятор» (рис. 7.33). В чем же его достоинства?

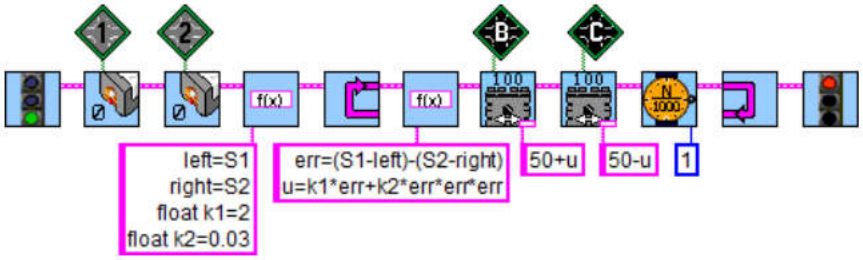


Рис. 7.33. Пропорционально-кубический регулятор для движения по линии с двумя датчиками.

Во-первых, он позволяет «разгрузить» пропорциональную составляющую, понизив ее коэффициент. При малых отклонениях «куб» практически не оказывает влияния на движение. Зато на поворотах, когда невязка повышается значительно, кубическая составляющая стремительно «вырастает» из своего понижающего коэффициента.

Рассмотрим пример с коэффициентом $k2 = 0.03$. При малом отклонении $err = 5$ кубическая составляющая принимает значение $uk = 0.03 \cdot 5 \cdot 5 \cdot 5 = 3.75$, что несущественно для диапазона мощностей мотора –100...100. Когда отклонение достигнет $err = 10$, мы уже получим $uk = 30$. При $err = 15$ «куб с коэффициентом» превысит значение 100.

Особенно ярко преимущества кубического регулятора проявляются при движении по линии с поворотами на 90 градусов.

Плавающий коэффициент

Используя третий датчик, поставленный спереди по направлению движения, можно научиться «предсказывать» состояние робота на следующем шаге. Нормальное положение для центрального датчика — над черной линией (рис. 7.35). Если он начинает отклоняться к серому вплоть до белого, это означает, что робот теряет линию. В такой ситуации можно повлиять на коэффициенты регулятора, который используется для слежения за линией, в сторону усиления. Получится упреждающее воздействие, и робот сможет «подготовиться» в грядущему повороту.

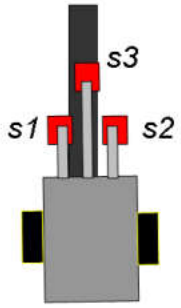


Рис. 7.35. Расположение трех датчиков.

Если крайние датчики при движении на пропорциональном регуляторе перед стартом калибруются на белом, то центральный, очевидно, калибруется на черном. Далее его отклонение от черного будет влиять на коэффициент усиления регулятора. Но надо учитывать то, что при калибровке могло быть получено не наименьшее значение освещенности и на линии могут найтись участки темнее. Поэтому, чтобы не получить отрицательное значение коэффициента, следует дополнить его положительным значением. А чтобы понизить резко вырастающее отклонение на центральном датчике, следует разделить его на другой коэффициент. Все эти значения подбираются для конкретного робота. Например, $c = 1$, $k2 = 3$. Далее следует калибровка:

center = *S*3.

Коэффициент определяется в цикле:

*k*1 = *c* + (*S*3 - *center*) / *k*2.

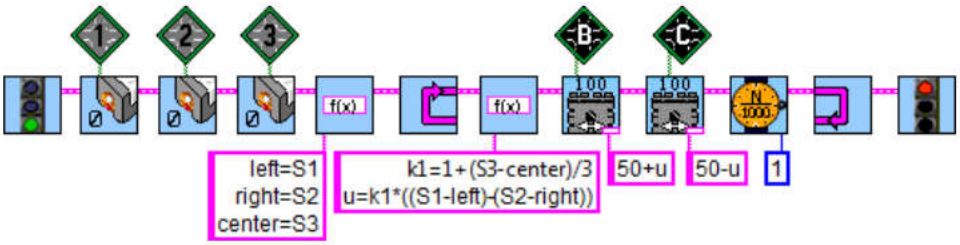


Рис. 7.36. Движение по линии на пропорциональном регуляторе с плавающим коэффициентом.

Полученный закон управления коэффициентами усиления можно применить не только к пропорциональной составляющей, но и к любой другой, а также к управляющему воздействию в целом (рис. 7.36).

ПИД-регулятор

Пропорционально-интегрально-дифференциальный (ПИД) регулятор является одним из наиболее популярных и используется в огромном количестве устройств самых разных типов, в которых требуются быстрота реакции и точность позиционирования системы. Как следует из названия, этот регулятор состоит из суммы трех компонент и графически изображен на рис. 7.37.

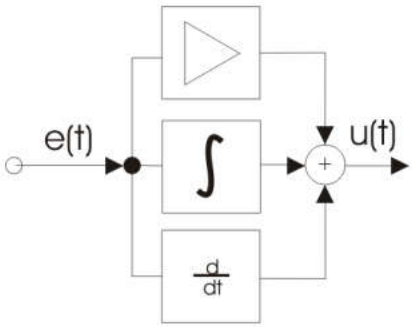


Рис. 7.37. Схема ПИД-регулятора.

Это упрощенная схема. На вход регулятора подается значение динамической ошибки $e(t)$, а на выходе вырабатывается управляющее воздействие $u(t)$:

$$u(t) = p + i + d = k_p \cdot e(t) + k_i \cdot \int_0^t e(\tau)d\tau + k_d \cdot \frac{de}{dt}.$$

Пропорциональная составляющая, изображенная на схеме треугольником, отвечает за позиционирование системы в заданном состоянии. В некоторых случаях может вызвать перерегулирование с последующими автоколебаниями. То есть П-регулятор может «перестараться» и робота начнет заносить из стороны в сторону.

Интегральная составляющая накапливает отрицательный опыт (суммирует ошибки) и вырабатывает компенсирующее воздействие. При минимальных отклонениях пропорциональная составляющая «слабеет» и интегральная, за счет своего быстрого увеличения суммированием, помогает «дотянуть» регулируемую величину до уставки.

Дифференциальная составляющая (Д-составляющая) следит за скоростью изменения состояния системы и препятствует возможному перерегулированию. В некоторых случаях Д-составляющая противоположна пропорциональной по знаку, а в некоторых совпадает.

С пропорциональной составляющей мы уже знакомы, дифференциальная описана в предыдущей главе 6. Возьмемся за интегральную. Эта составляющая определяется динамически, суммируясь с предыдущим значением:

$$i = i + k_i \cdot e(t) \cdot dt.$$

Физический смысл величины $e(t) \cdot dt$ состоит в том, что она пропорциональная длительности нахождения системы в состоянии ошибки. Поскольку коэффициент k_i выносится за скобки, можно говорить о величине i как сумме длительностей ошибок. Таким образом, мы находим интеграл путем суммирования.

Рассмотрим применение ПИД-регулятора на примере робота, балансирующего на двух колесах. Эта классическая задача может быть решена с помощью различных датчиков множеством способов. В предложенном примере использован датчик освещенности и простейшая форма ПИД-регулятора. Однако для достижения стабилизации робота потребуется использовать более точные показания датчика.

Формат RAW

Данные с датчиков поступают в контроллер NXT в необработанном «сыром» виде. Все датчики передают операционной системе цифровое значение от 0 до 1023, которое затем обрабатывается соответствующим драйвером и приводится к более понятному виду (расстояние 0...255, освещенность 0...100, касание 0 или 1 и т. д.). Но данные можно получать и, минуя драйвер, напрямую. Такой необработанный формат принято называть RAW (от англ. «сырой»). В некоторых случаях с помощью него можно получить бóльшую точность. Так, например, диапазон значений датчика освещенности может увеличиться примерно в 10 раз. Именно эта возможность использована далее.

Получать данные в формате RAW можно и в Robolab, и в RobotC. Для этого датчик инициализируется соответствующим образом, а данные с него считываются с использованием специальной предопределенной переменной.

Балансирующий робот

Конструкция робота-сигвея изображена на рис. 7.38: вертикально расположенный контроллер, близко поставленные колеса и датчик освещенности, направленный вниз. Алгоритм будет несколько сложнее.

Принцип стабилизации сигвея в положении равновесия состоит в следующем. Если робот наклоняется вперед, показания на датчике освещенности повышаются за счет отраженного света. В ответ на это вырабатывается управляющее воздействие, заставляющее робота ехать вперед и тем самым снова принимать вертикальное положение.

При отклонении назад показания датчика понижаются и робот начнет движение назад. За все это отвечает пропорциональная составляющая. На роль интегральной и дифференциальной составляющих отводится страховка от перерегулирования.

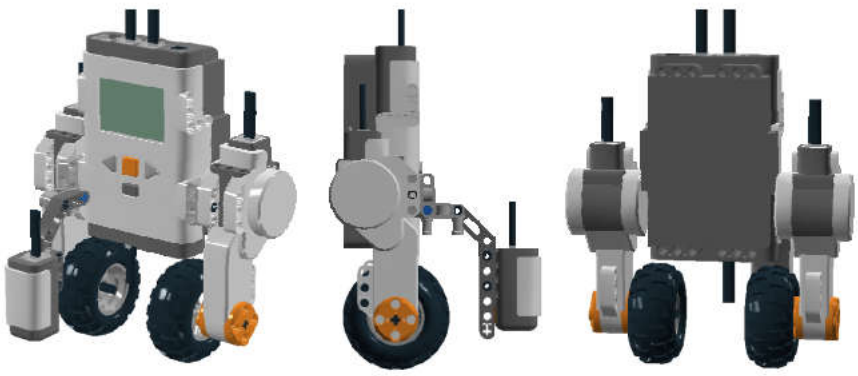


Рис. 7.38. Балансирующий робот-сигвей.

На рис. 7.39 представлен алгоритм в Robolab. Бóльшую его часть занимает инициализация переменных. Для повышения точности не только данные с датчика считываются в формате RAW, но большинство переменных объявляется в вещественном формате float. Собственно ПИД-алгоритм находится в цикле.

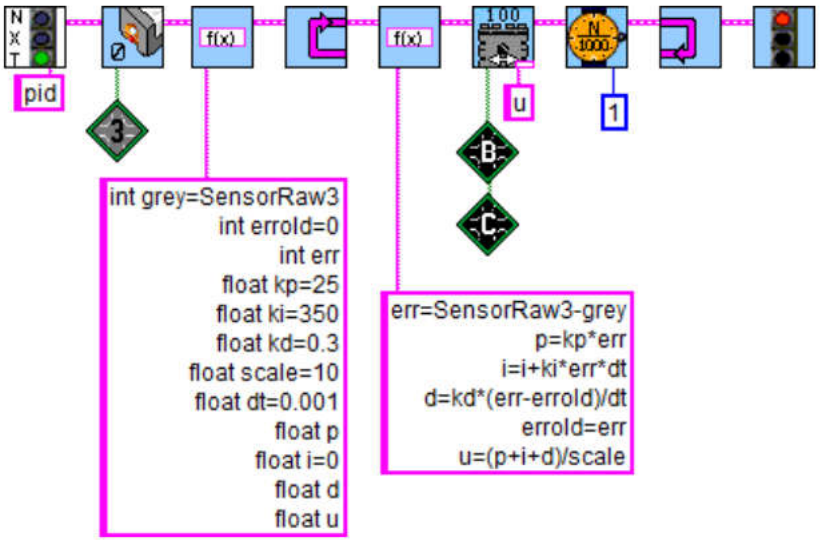


Рис. 7.39. Алгоритм балансировщика основан на ПИД-регуляторе.

Следуя традиции движения по линии, в качестве уставки используем переменную *grey* — средние показания датчика освещенности в положении равновесия. Новый параметр *scale* задает масштабирование управляющего воздействия. По сути, это ослабляющий коэффициент, поскольку вырабатываемое регулятором значение слишком высоко для моторов NXT. Можно было бы внести его внутрь уже имеющихся коэффициентов, но для RobotC этот параметр будет другой, а коэффициенты те же.

С приведенными коэффициентами робот хорошо стабилизируется на однотонном светлом линолеуме или парте. То есть ему не требуется белый цвет поверхности. Для запуска нужно достаточно точно установить сигвея в положение равновесия. Если робот стартует при некотором наклоне вперед или назад, то сразу начнет движение в направлении наклона.

Аналогичный пример на RobotC несколько отличается в силу ряда причин. Во-первых, быстродействие NXT с прошивкой этой среды выше примерно в 1.4 раза, чем у Robolab, поэтому коэффициент scale следует увеличить. Во-вторых, RAW-значения передаются в правильном порядке и потребуется установить реверс моторов или просто подавать отрицательное управляющее воздействие:

```
task main()
{
  int grey=SensorRaw[S3];
  int err, errold=0;
  float kp=25, ki=350, kd=0.3;
  float scale=14;
  float dt=0.001;
  float p, i=0, d, u;
  while (true)
  {
    err= grey-SensorRaw[S3]; //Отклонение с обратным знаком
    p=kp*err;
    i=i+ki*err*dt;
    d=kd*(err-errold)/dt;
    errold=err;
    u=(p+i+d)/scale;
    motor[motorB]=u;
    motor[motorC]=u;
    wait1Msec(1);
  }
}
```

Элементы теории автоматического управления в школе[1]

Важной и интересной методической задачей является «переброска мостика» между областями знаний специалиста и школьника, помогающая учащимся школы увидеть перспективу будущей специальности, т.е. осуществить профориентацию, а студентам увидеть практическую применимость своих профессиональных знаний. Для достижения подобного эффекта были разработаны приемы расчета регуляторов, использующие математический аппарат, не выходящий за рамки школьных программ по математике и физике. В частности, вместо дифференциальных уравнений использованы разностные, хорошо соответствующие дискретному характеру взаимодействия объекта и регулятора при компьютерном управлении.

Рассмотрим, например, задачу построения пропорциональных (П) и пропорционально-дифференциальных (ПД) регуляторов в задаче управления движением мобильного робота вдоль стены. Обозначим через x_t расстояние между роботом и стеной, через θ_t — курсовой угол робота, а через u_t — управляющее воздействие в момент с порядковым номером t, соответственно, где $t = 0, 1, 2, \ldots$ — номера моментов изме-

[1] Автор статьи – д. техн. наук, профессор А. Л. Фрадков.

рений. Считается, что опрос датчиков и изменения величины управляющего воздействия производится через равные промежутки времени h. Для задач управления Lego NXT роботами естественно считать, что управляющим воздействием является разность угловых скоростей вращения колес, пропорциональная скорости изменения курсового угла:

$$\theta_{t+1} = \theta_t + u_t hr / b, \tag{1}$$

где r — радиус колес, b — база транспортного средства (расстояние между колесами). Легко также видеть, что расстояние робота от стены за время h изменится на величину $h \sin \theta_t$, т.е. имеет место соотношение

$$x_{t+1} = x_t + v_t h \sin \theta_t . \tag{2}$$

Считая отклонения курса от номинального $\theta t = 0$ малыми, а среднюю скорость робота постоянной: $vt=v$, динамику изменения переменных состояния робота в первом приближении можно описать линейными уравнениями состояния:

$$x_{t+1} = x_t + vh\theta_t , \ \theta_{t+1} = \theta_t + u_t hr / b. \tag{3}$$

Исключая переменную θ_t, приходим к разностному уравнению 2-го порядка, непосредственно связывающему управляющую и регулируемую переменные:

$$x_{t+2} - 2 x_{t+1} + x_t = g u_t , \tag{4}$$

где $g = h2vr / b$.

Зададим желаемое расстояние до стены $x^*>0$ и определим цель управления (ЦУ) соотношением

$$x_t \to x^* \text{ при } t \to \infty. \tag{5}$$

Теперь естественным образом введем на содержательном уровне понятие асимптотической устойчивости, как свойства решений системы (4), обеспечивающего достижение ЦУ (5) при любых начальных условиях, достаточно мало отличающихся от целевых. Легко видеть, что при $u_t = 0$ решением уравнения (4) является любое постоянное значение $x_t = x^*$. Но поскольку уравнение (4), соответствующее модели двойного интегратора (двойного сумматора), не обладает свойством асимптотической устойчивости, ЦУ (5) при постоянном управлении не достигается. Это легко демонстрируется как аналитически — суммированием ря-

да натуральных чисел, так и экспериментально, выставляя робот в различные начальные положения.

После такой демонстрации весьма естественно ввести понятия регулятора и обратной связи по результатам измерений и они легко осваиваются школьниками. Сначала рассматривается простейший пропорциональный регулятор (П-регулятор):

$$u_t = K_0(x^* - x_t), \tag{6}$$

где K_0 — коэффициент усиления регулятора. Подстановкой (6) в (4) получаем уравнение замкнутой системы

$$x_{t+2} - 2\,x_t + {}_1 + (1 + gK_0)x_t = gK_0x^*. \tag{7}$$

Переходя к уравнениям для отклонений $yt = x^* - xt$, получим однородное уравнение

$$y_{t+2} - 2\,y_{t+1} + (1 + gK_0)y_t = 0. \tag{7a}$$

Возникающий вопрос об устойчивости системы (7) теперь можно решить в случае общей однородной системы 2-го порядка:

$$x_{t+2} + a_1x_{t+1} + a_0x_t = 0, \tag{8}$$

записав ее общее решение в виде суммы двух геометрических прогрессий со знаменателями λ_1 и λ_2 (возможно, комплексными) равными корням квадратного трехчлена $\lambda_2 + a_1\lambda + a_0$. Условием асимптотической устойчивости оказывается пара неравенств $|\lambda_1| < 1$, $|\lambda_2| < 1$, означающая, что обе геометрические прогрессии являются бесконечно убывающими. Теперь стандартное условие устойчивости в терминах коэффициентов

$$a_0 < 1,\ 1 + a_0 > |\,a_1| \tag{9}$$

школьники могут вывести самостоятельно.

Применяя правило (9) к уравнению системы с П-регулятором (7), убеждаемся в том, что неравенства (9) несовместны, т.е. что не существует числа K_0, делающего систему (7) асимптотически устойчивой. Таким образом, П-регулятор оказывается неработоспособным, и для решения задачи поиски нужно продолжить.

Следующим этапом является построение ПД-регулятора, который можно описать соотношением

$$u_t = K_0(x^* - x_t) + K_1(x_{t+1} - x_t), \qquad (10)$$

где K_1 — коэффициент усиления по разности (дифференциальный). Поскольку в момент времени t измерить x_{t+1} невозможно, для реализации регулятора (10) можно воспользоваться соотношением

$$x_{t+1} - x_t = vh\theta_t = vh\theta_{t-1} + u_{t-1}\, vh2r / b.$$

Подставляя в (10), получим ПД-закон управления в виде

$$u_t = [K_0(x^* - x_t) + K_1\, vh(\theta_{t-1} + u_{t-1}\, hr / b)], \qquad (11)$$

т. е. для применения (11) требуется помнить значения курсового угла и управления на предыдущем шаге. Для исследования устойчивости и выбора коэффициентов ПД-регулятора подставим (10) в (4). Получим уравнение замкнутой системы:

$$x_{t+2} - (2 + gK_1)x_{t+1} + (1 + gK_0 + gK_1)x_t = gK_0x^*. \qquad (12)$$

Проверяя его на устойчивость с помощью критерия (9), убеждаемся в том, что достаточными условиями устойчивости служат два неравенства

$$K_0 > 0,\ K_0 + K_1 < 0. \qquad (13)$$

Таким образом, для построения работоспособного регулятора следует выбирать дифференциальный коэффициент, превосходящий пропорциональный по абсолютному значению и противоположный по знаку. В справедливости этого правила также следует убедиться в экспериментах.

Предложенные методики исследования и реализации мобильных роботов на основе элементарной теории управления способствуют росту интереса учащихся к робототехнике и, более широко, к инженерным наукам.

Глава 8. Задачи для робота

Управление без обратной связи

В задачах управления обычно существуют два объекта: управляющий и управляемый. В простейшем варианте от управляющего объекта поступает команда и управляемый выполняет ее, ничего не сообщая о результате или об изменившихся условиях работы. В этом суть прямой связи (рис. 8.1).

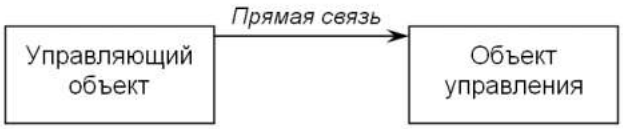


Рис. 8.1. Прямая связь в управлении.

С точки зрения мобильного робота управляющий объект — это его контроллер с запущенной программой, объект управления — это его колеса и корпус (шасси). Управляющие команды контроллер подает на моторы, при прямой связи руководствуясь показаниями своих внутренних часов — таймера.

Первый класс задач, с которых начинается программирование, — это управление перемещениями робота. Рассмотрим их по порядку. В качестве сред программирования используем Robolab 2.9.4 для начинающих и RobotC для подготовленных программистов. В качестве модели робота — любую двухмоторную тележку.

Движение в течение заданного времени вперед и назад

Для движения вперед используются команды управления моторами. Эти команды просто включают моторы. Особенность NXT заключается в том, что после окончания выполнения программы сохраняются все установки в поведении робота, но на моторы перестает подаваться напряжение. Таким образом, просходит пуск и сразу плавное торможение (рис. 8.2).

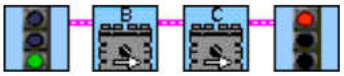


Рис. 8.2. Включение моторов.

```
task main()
{
  motor[motorB] = 100; // моторы вперед с
  motor[motorC] = 100; // максимальной мощностью
}
```

Обе команды выполняются практически мгновенно. Если сразу следом за ними выключить моторы, то тележка просто дернется и останется стоять на месте (рис. 8.3):

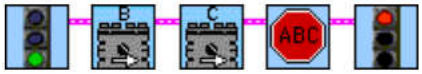


Рис. 8.3. Остановка при попытке начать движение.

```
task main()
{
  motor[motorB] = 100;
  motor[motorC] = 100;
  motor[motorB] = 0; // стоп мотор
  motor[motorC] = 0;
}
```

Таким образом, для осуществления движения требуется некоторая задержка перед выключением моторов. Команды ожидания не производят никаких конкретных действий, зато дают возможность моторам выполнить свою часть работы (рис. 8.4):

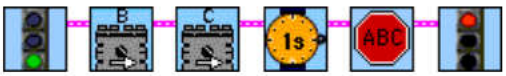


Рис. 8.4. Правильный порядок управления моторами.

```
task main()
{
  motor[motorB] = 100;
  motor[motorC] = 100;
  wait1Msec(1000);   // Ждать 1000 мс
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

Движение вперед или назад, очевидно, определяется направлением вращения моторов (рис. 8.5). Для смены направления не требуется остановка:

Рис. 8.5. Проехать секунду вперед, секунду назад и остановиться.

```
task main()
{
  motor[motorB] = 100;
  motor[motorC] = 100;
  wait1Msec(1000);
  motor[motorB] = -100; // «Полный назад»
  motor[motorC] = -100;
  wait1Msec(1000);
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

В момент смены направления на высокой скорости возможен занос. Плавное торможение возможно. Для этого перед подачей команды «назад» с моторов снимается напряжение и робот некоторое время едет по инерции (рис. 8.6).

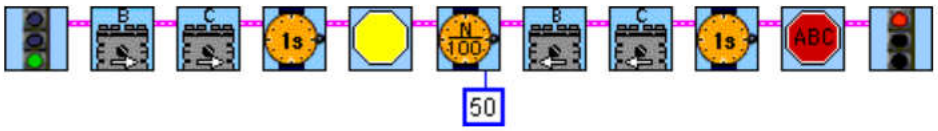


Рис. 8.6. Перед сменой направления полсекунды ехать по инерции.

Более краткий промежуток, чем 1 секунда, задается с помощью команды «N/100» и модификатора. В Robolab 2.9.4 можно задавать время в миллисекундах командой «N/1000»:

```
task main()
{
  motor[motorB] = 100;
  motor[motorC] = 100;
  wait1Msec(1000);
  // Включить плавающий режим управления моторами
  bFloatDuringInactiveMotorPWM = true;
  motor[motorB] = 0;
  motor[motorC] = 0;
  wait1Msec(500);
  motor[motorB] = -100;
  motor[motorC] = -100;
  wait1Msec(1000);
  // Включить режим «торможения»
  bFloatDuringInactiveMotorPWM = false;
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

В Robolab обычными командами моторы включаются в плавающем режиме, а в RobotC по умолчанию используется режим «торможения»,

который позволяет достичь более точного управления. Но и в Robolab существуют «продвинутые» команды управления моторами в режиме торможения да еще с диапазоном мощностей –100...100.

Повороты

Для выполнения поворота на месте достаточно включить моторы в разные стороны. Тогда робот будет вращаться приблизительно вокруг центра оси ведущих колес со смещением в сторону центра тяжести. Для более точного поворота надо подбирать время в сотых долях секунды (рис. 8.7). Однако при изменении заряда батареек придется вводить новые параметры поворота:

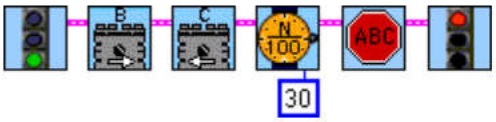


Рис. 8.7. Поворот на месте.

```
task main()
{
  motor[motorB] = 100;  // Моторы в разные
  motor[motorC] = -100; // стороны
  wait1Msec(300);
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

Существует другой тип поворотов. Если один из моторов остановить, а другой включить, то вращение будет происходить вокруг стоящего мотора. Поворот получится более плавным (рис. 8.8):

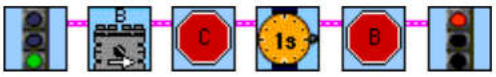


Рис. 8.8. Плавный поворот.

```
task main()
{
  motor[motorB] = 100;
  motor[motorC] = 0;
  wait1Msec(1000); // вращается только мотор B
  motor[motorB] = 0;
}
```

Движение по квадрату

Используя полученные знания управления моторами, можно запрограммировать движение по квадрату или другому многоугольнику с помощью цикла или безусловного перехода (рис. 8.9):

Рис. 8.9. Движение по многоугольнику с плавными поворотами.

```
task main()
{
  while (true){
    motor[motorB] = 100;
    motor[motorC] = 100;
    wait1Msec(1000);
    motor[motorC] = 0;
    wait1Msec(1000);
    motor[motorB] = 0;
  }
}
```

Уточнив длительность поворотов и число повторений, научим тележку объезжать квадрат по периметру 1 раз (рис. 8.10). Для точности поворотов снизим мощность моторов примерно вдвое. Задержки придется подобрать самостоятельно:

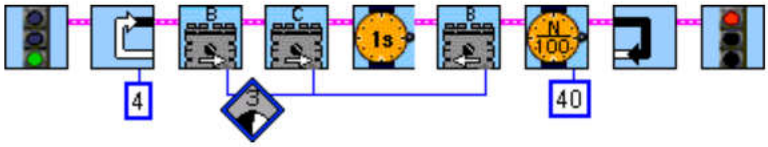


Рис. 8.10. Для поворота на 90 градусов длительность придется подобрать самостоятельно.

```
task main()
{
  for(int i=0;i<4;i++){ // Цикл выполняется 4 раза
    motor[motorB] = 50;
    motor[motorC] = 50;
    wait1Msec(1000);
    motor[motorC] = -50;
    wait1Msec(400);
    motor[motorB] = 0;
  }
}
```

Управление с обратной связью

Обратная связь

Появление обратной связи в системе означает то, что управляющий объект начинает получать информацию об объекте управления (рис. 8.11).

Рис. 8.11. Управление с обратной связью.

Обратная связь осуществляется с помощью датчиков, прикрепленных, например, на корпус робота. Данные поступают в контроллер, который является управляющим объектом.

Точные перемещения

Чтобы поворот не зависел от заряда батареек, можно воспользоваться встроенным в двигатели датчиком оборотов, «энкодером», который позволяет делать измерения с точностью до 1 градуса. Для более эффективного управления задействуем в Robolab «продвинутые» команды, считая что при повороте тележки на 90 градусов левое колесо поворачивается на 250 градусов вокруг своей оси (рис. 8.12):

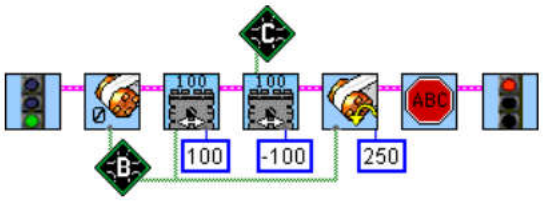


Рис. 8.12. Точный поворот на месте.

```
task main() {
  nMotorEncoder[motorB]=0; // Инициализация энкодера
  motor[motorB] = 100;
  motor[motorC] = -100;
  // Пустой цикл ожидания показаний энкодера
  while(nMotorEncoder[motorB]<250);
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

Теперь читателю нетрудно будет самостоятельно построить алгоритм движения по квадрату с использованием датчика оборотов. А при движении по лабиринту использование датчика оборотов значительно повысит вероятность его прохождения.

Кегельринг

Танец в круге

Для выполнения этой задачи надо собрать стандартную трехколесную тележку: два передних колеса ведущие, одно заднее на шарнире. В наборе 8527 для этой цели может быть приспособлена конструкция Tribot, правда, центральный третий мотор не будет использован.

По центру должен быть расположен датчик освещенности, направленный строго вниз на расстоянии около 10 мм от пола (рис. 8.13).

Рис. 8.13. Трехколесная тележка с креплением датчика освещенности.

Теперь приготовим ринг. Это может быть круг или его подобие диаметром 50—100 см, очерченный двумя-тремя слоями черной изоленты (ширина черной линии около 50 мм). Цвет поверхности, на которой круг расположен, особого значения не имеет. Важно, чтобы она была светлой и однотонной. Главное условие успешности опыта состоит в том, что показания датчика на черной линии и внутри круга должны различаться не менее чем на 10—15 пунктов, а лучше на 20—25.

Робот ставится в центр и при старте должен двигаться внутри круга, не выходя за его пределы.

Последовательность действий такова:

— ехать вперед, пока показания датчика не понизятся на 5 пунктов (лучше 10);

— отъехать немного назад (полсекунды);

— развернуться примерно на 120—150 градусов (тоже по времени);

— повторять пункты 1—3 бесконечно.

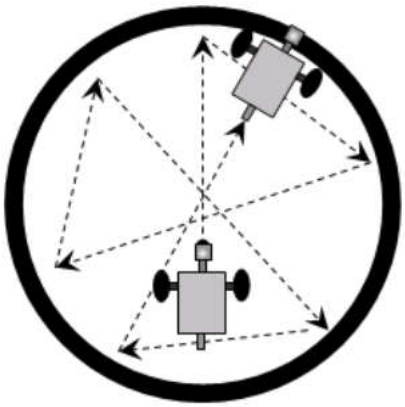


Рис. 8.14. Траектория движения робота в круге.

Рассмотрите примеры программ на языках Robolab (рис. 8.15) и RobotC:

Рис. 8.15. Алгоритм «Танец в круге».

```
task main()
{
  int white=SensorValue[S1]; // Запомнить показания
  while (true)
  {
    motor[motorB] = 100;
    motor[motorC] = 100;
    while(SensorValue[S1]>white-8); // Ждать понижения
    motor[motorB] = -100;
    motor[motorC] = -100;
    wait1Msec(500);
    motor[motorB] = 100;
    wait1Msec(600);
  }
}
```

В результате выполнения программы робот будет двигаться внутри круга, «вычерчивая» ломаную линию (рис. 8.14).

Параметры, указанные в модификаторах, можно подобрать самостоятельно: степень понижения освещенности на черной линии, время отъезда назад и время поворота.

Не упасть со стола

Сделать выступление робота более зрелищным можно и без изоленты. Достаточно иметь светлый стол, на котором робот станет исполнять свои «па», стараясь не упасть на краю. Это возможно благодаря тому, что датчик освещенности, направленный в пол с высоты стола, практически не получает отраженного света и показывает значение, сравнимое с черной границей. Только, страхуя тележку руками на краю стола, не следует подносить их слишком близко. Отраженным светом от ладошек робот может быть сбит с толку и просто поедет вперед, в пропасть.

Датчик освещенности имеет смысл установить на длинной балке примерно на 10—15 см впереди робота. Это даст возможность удержаться на поверхности стола в самых сложных положениях. Программа будет практически такой же, как и в предыдущем опыте.

Вытолкнуть все банки

Вернемся к нашему кругу. Будем считать, что его диаметр — 1 м. Несколько пластиковых стаканчиков или пустых жестяных банок, расставленные внутри за черной линией, — это мусор, от которого необходимо очистить круг за кратчайшее время.

Первые попытки запуска с робота покажут несколько недостатков:

— стаканчики попадают под колеса, падают и плохо выталкиваются;

— даже вытолкнутые стаканчики остаются частично внутри круга, поскольку, увидев край, робот сразу устремляется назад;

— робот ведет себя, как слон в посудной лавке;

— робот делает много движений впустую.

Избавимся от первого недостатка. Для этого следует построить бампер шириной 20—25 см непосредственно перед датчиком освещенности (рис. 8.16). Подумайте, можно ли спрятать датчик за бампером?

Рис. 8.16. Бампер для защиты колес.

Второй и третий недостатки устраняются программно. Пусть, увидев край, тележка еще немного движется вперед, выталкивая стаканчик, и только после этого отъезжает внутрь круга. Например, как показано на рис. 8.17.

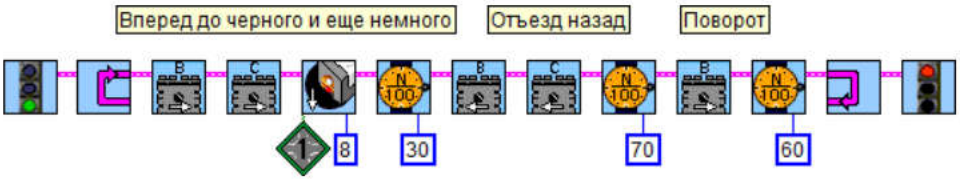


Рис. 8.17. Алгоритм «Танец в круге» с выталкиванием кеглей.

Самый надежный способ заехать точно за пределы черной линии — это дождаться значения белого на датчике освещенности. Поэтому время можно заменить на «ожидание белого». Для экономии места стоит сгруппировать команды управления моторами, а также использовать «реверс» при смене направления на работающих моторах (рис. 8.18).

Рис. 8.18. Алгоритм «Танец в круге» с выездом точно за его пределы.

Ожидаемое значение черного повышено до 10, поскольку следующее за ним ожидание значение белого относительно и может не сработать даже при небольших помехах освещенности:

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-10);
    while(SensorValue[S1]<white-5);
    motor[motorB] = motor[motorC] = -100;
    wait1Msec(500);
    motor[motorB] = 100;
    wait1Msec(600);
  }
}
```

Теперь стоит поработать над точностью движения, по возможности не теряя скорости.

В зависимости от конструкции робота при резкой смене направления он может потерять равновесие или просто «встать на дыбы» на передние колеса. Поэтому последние несколько сантиметров можно проехать на торможении по инерции, т. е. полностью освободив моторы.

Точность поворота будет зависеть от того, какие команды подаются на моторы и по какому принципу рассчитывается длительность поворота. К сожалению, таймер — ненадежный помощник. По инерции на малых промежутках времени робот может поворачиваться на различные углы.

Можно пожертвовать реверсом в последней команде управления мотором B, чтобы достичь неторопливого движения обоими моторами. Длительность поворота при этом немного возрастет (рис. 8.19):

Рис. 8.19. Алгоритм «Танец в круге» с плавным торможением.

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-10);
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = motor[motorC] = 0;
    while(SensorValue[S1]<white-5);
    bFloatDuringInactiveMotorPWM = false;
    motor[motorB] = motor[motorC] = -100;
    wait1Msec(500);
    motor[motorB] = -50;
    motor[motorC] = 50;
    wait1Msec(600);
  }
}
```

Есть еще одна тонкость. На разных поверхностях соотношение времени вращения колес и реального перемещения тележки будет различным. Поэтому для надежности при возврате назад и повороте можно ввести ожидание оборотов моторов.

Для точности управления моторами необходимо использовать другой тип команд: с контролируемым вращением. Эти команды находятся в разделе Advanced Output Control и позволяют задавать мощность моторов от –100 до 100. В новом примере для компактности разместим все числовые параметры сверху, а модификаторы портов — снизу (рис. 8.20).

Рис. 8.20. Алгоритм «Танец в круге» с движением по датчикам оборотов.

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-10);
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = motor[motorC] = 0;
    while(SensorValue[S1]<white-5);
    bFloatDuringInactiveMotorPWM = false;
    nMotorEncoder[motorC]=0; // Инициализация энкодера
    motor[motorB] = motor[motorC] = -100;
    while(nMotorEncoder[motorC]>-180);
    nMotorEncoder[motorC]=0;
    motor[motorB] = -50;
    motor[motorC] = 50;
    while(nMotorEncoder[motorC]<120);
  }
}
```

Как устранить четвертый недостаток, связанный с избыточными движениями, показано далее.

Не делать лишних движений

Желание проконтролировать положение робота приводит к необходимости изменить траекторию движения таким образом, чтобы каждый раз, доехав до края, он возвращался в центр круга. Если бы не было встроенных датчиков оборотов, пришлось бы засекать время, которое

робот ехал вперед, и давать команду столько же ехать назад. Точность такого решения оставляет желать лучшего, хотя и оно имеет право на существование. Вот соответствующий пример (рис. 8.21):

Рис. 8.21. Алгоритм «Танец в круге» с возвратом по времени.

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    ClearTimer[T1];  // Сбросить таймер
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-5);
    int t = time1[T1];  // Запомнить прошедшее время
    ClearTimer[T1];
    motor[motorB] = motor[motorC] = -100;
    while(time1[T1]<t); // Двигаться назад то же время
  }
}
```

Эта программа гоняет робота вперед-назад: до линии и на исходную позицию (пока без поворотов).

К счастью, сервомоторы NXT имеют встроенный датчик оборотов, этим непременно надо воспользоваться. Но как определить, сколько оборотов надо сделать, чтобы вернуться в центр? Так же как с таймером, запоминать результат в контейнер? Гораздо проще. Для этого нужно всего-навсего каждый раз обнулять показания датчика оборотов, когда робот оказывается в центре. Отчет времени невозможно повернуть вспять, чтобы снова прийти в нулевую точку, а моторы — можно. По замыслу робот проезжает некоторое число оборотов вперед, после чего следует назад, пока на датчике оборотов снова не будет ноль.

В Robolab для этого следует использовать специфический блок, который не обнуляет показания датчика оборотов при вызове (с буквой A на пиктограмме). Кроме того, в примере добавлены модификаторы *Encoder C* из палитры *NXT Commands* для ясности различия между командами управления мотором и датчиками (рис. 8.22).

Рис. 8.22. Алгоритм «Танец в круге» с датчиками оборотов.

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    nMotorEncoder[motorC]=0;
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-5);
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = motor[motorC] = 0;
    while(SensorValue[S1]<white-5);
    bFloatDuringInactiveMotorPWM = false;
    motor[motorB] = motor[motorC] = -100;
    while(nMotorEncoder[motorC]>0);
    nMotorEncoder[motorC]=0;
    motor[motorB] = -50;
    motor[motorC] = 50;
    while(nMotorEncoder[motorC]<60);
  }
}
```

Результат уже значительно лучше, но робот все равно иногда промахивается мимо кеглей. Конечно, ведь он поворачивает вслепую. Надо бы оснастить его зрением. Для этого подойдет датчик расстояния. Прежде чем начать использовать его, попробуем разобраться, как «сонар» работает.

Два глазка, которые делают робот похожим на живое существо, служат для разных целей. Один из них передает ультразвуковой сигнал, другой принимает его. По наблюдениям автора, передающим является правый, если смотреть на датчик со стороны разъема подключения, т. е. сзади. Учитывая то, что скорость ультразвука в воздухе относительно невелика (330—350 м/с), расположим датчик горизонтально так, чтобы передающий глазок был первым по ходу вращения робота. Тогда при вращении будет больше вероятность, что отраженный сигнал попадет в принимающий глазок, идущий следом. В обратном случае часть стаканчиков робот будет пропускать.

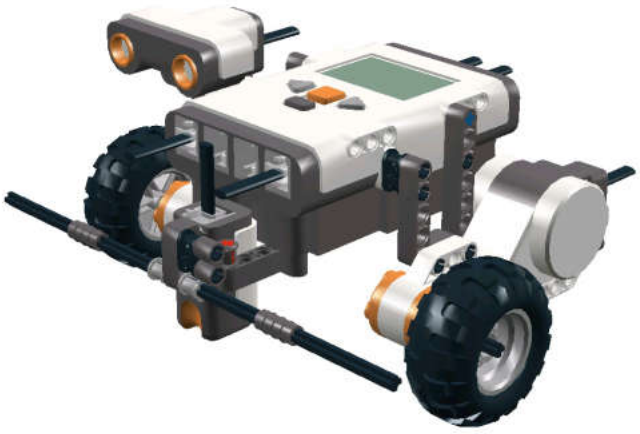


Рис. 8.23. Ультразвуковой датчик для поиска кеглей и модифицированный бампер.

Кроме того, датчик можно расположить вертикально, любой стороной, и он будет работать вполне сносно. Заодно усовершенствуем конструкцию, спрятав датчик освещенности за бампером (рис. 8.23).

Итак, вращение осуществляется до тех пор, пока на датчик расстояния не поступит сигнал, например «ближе 45». То есть обнаружен объект на расстоянии ближе 45 см. Надо иметь ввиду, что на кегли, которые робот уже вытолкнул за линию, он не должен обращать внимания. Поэтому указанное расстояние не следует делать больше радиуса круга (рис. 8.24, 8.25):

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    motor[motorB] = 50;
    motor[motorC] = -50;
    while(SensorValue[S4]>45); // Ждем появление объекта
    nMotorEncoder[motorB]=0;
    motor[motorB] = motor[motorC] = 100;
    while(SensorValue[S1]>white-5);
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = motor[motorC] = 0;
    wait1Msec(200);
    bFloatDuringInactiveMotorPWM = false;
    motor[motorB] = motor[motorC] = -100;
    while(nMotorEncoder[motorB]>0);
  }
}
```

Рис. 8.24. Алгоритм «Танец в круге» с поиском кеглей.

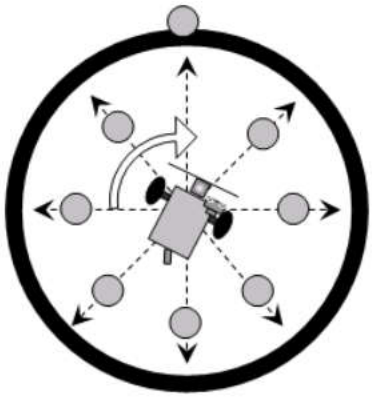


Рис. 8.25. Поиск и выталкивание кеглей с возвратом к центру.

В зависимости от конструкции робота в приведенном алгоритме может быть один недостаток. Вернувшись в центр круга, робот начинает вращение до появления кегли. Однако в силу неточности ультразвукового датчика он может среагировать сразу на уже вытолкнутую кеглю. Поэтому имеет смысл, во-первых, выталкивать их подальше за пределы круга, во-вторых, начинать поворот в центре «вслепую», чтобы успеть отвернуться от кегли, а, в-третьих, все действия поначалу выполнять в замедленном темпе (рис. 8.26).

Рис. 8.26. Усовершенствованный поиск кеглей.

```
task main()
{
  int white=SensorValue[S1];
  while (true){
    motor[motorB] = 20;
    motor[motorC] = -20;
    wait1Msec(200);
    while(SensorValue[S4]>45);
    nMotorEncoder[motorB]=0;
    motor[motorB] = motor[motorC] = 50;
    while(SensorValue[S1]>white-5);
    bFloatDuringInactiveMotorPWM = true;
    motor[motorB] = motor[motorC] = 0;
    wait1Msec(500);
    bFloatDuringInactiveMotorPWM = false;
    motor[motorB] = motor[motorC] = -50;
    while(nMotorEncoder[motorB]>0);
  }
}
```

Если все кегли оказались за пределами круга, надо попробовать вытолкнуть их еще раз, засечь время и подумать о том, как это сделать вдвое, втрое быстрее. Очевидно, сэкономить можно на остановках и поворотах, которые отнимают драгоценные секунды. Стоит подумать и над траекторией движения: возврат в центр совсем не обязателен; вытолкнув одну кеглю, можно сразу браться за вторую. Робот способен вычерчивать звезды, восьмерки, спирали и другие фигуры, оптимальным образом очищая круг. Существует целая наука игры в кегельринг, ее основы изложены на сайте *http://www.myrobot.ru/*.

Движение по спирали

Движение по спирали — самый эффективный алгоритм, позволяющий выбить кегли за кратчайшее время. В конце 2012 г. в Санкт-Петербурге был поставлен очередной рекорд — 2.4 с. Надо признать, что, выбивая кегли таким образом, робот не использует датчик расстояния и лишь повторяет выверенную траекторию спирали (рис. 8.27). Но и это само по себе интересно.

Итак, будем рассматривать спираль как дугу с меняющимся радиусом. Таким образом, для того чтобы начать двигаться по спирали, сперва надо научиться ездить по кругу. Очевидно, что радиус определяется разностью скоростей моторов. Для круга кегельринга, диаметр которого равет 1 м, скорости левого и правого моторов могут быть равны соответственно $v1 = 100$ и $v2 = 70$. Важным параметром является расстояние между колесами, поэтому окончательные значения следует найти самостоятельно (рис. 8.28).

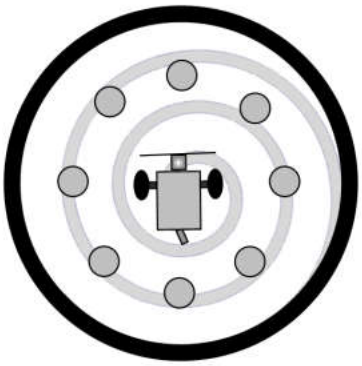


Рис. 8.27. Траектория движения робота по спирали в кегельринге.

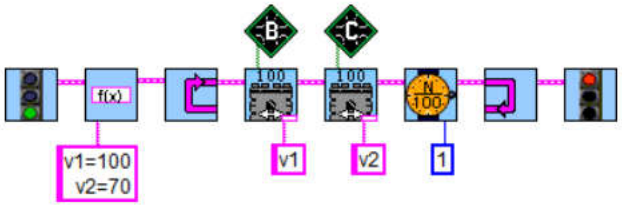


Рис. 8.28. Движение по часовой стрелке по окружности диаметром 1 м.

```
task main()
{
  int v1=100, v2=70;
  while(true) {
    motor[motorB] = v1;
    motor[motorC] = v2;
    wait1Msec(1);
  }
}
```

Минимальный радиус круга будет получен при значении $v2=0$ (робот будет крутиться вокруг своего правого колеса). Таким образом, для постепенного увеличения радиуса достаточно увеличивать скорость правого мотора от 0 до 70. Это можно выполнить, увеличивая $v2$ на единицу 70 раз. Достигнув желаемого значения, можно продолжить движение по кругу в течение некоторого времени (рис. 8.29).

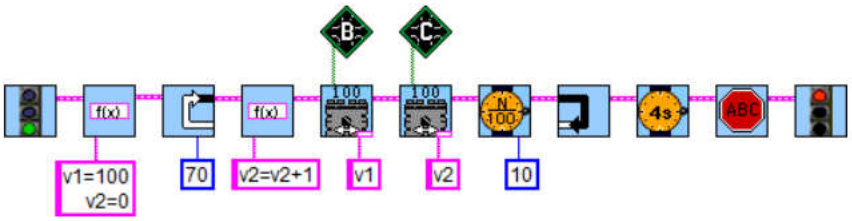


Рис. 8.29. Движение по спирали из центра круга.

```
task main()
{
  int v1=100, v2=0;
  for(v2=0; v2<70; v2++) {
    motor[motorB] = v1;
    motor[motorC] = v2;
    wait10Msec(10);
  }
  wait1Msec(4000);
  motor[motorB] = motor[motorC] = 0;
}
```

Задержка 0.1 с обеспечит выполнение цикла за 7 с, чего на первом этапе вполне достаточно. Однако надо признать, что такая спираль «раскручивается» слишком быстро, в результате чего робот пропускает часть кеглей. Увеличив время задержки или уменьшив величину приращения скорости, можно повысить плотность витков спирали. Для экспериментов со скоростью потребуются дробные значения, поэтому используем тип float. Введем величину *a*, которая будет означать ускорение второго мотора. Ускорение будет снижаться по мере приближения к краю круга, что обеспечит плотность движения в той области, где выше всего вероятность нахождения кеглей (рис. 8.30).

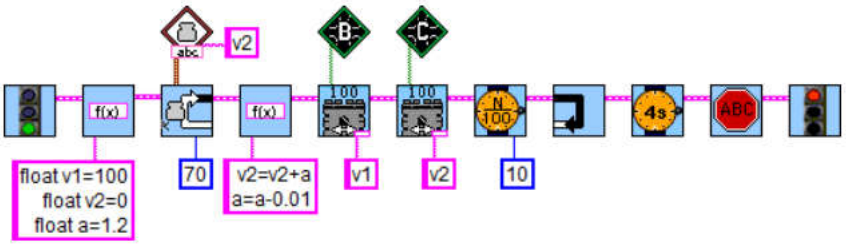


Рис. 8.30. Движение по спирали с повышением плотности витков.

В программе на языке RobotC применено ограничение величины *v*2 с помощью цикла for, а в программе на рис. 8.30 установлен блок цикла с условием по значению контейнера.

```
task main()
{
  float v1=100, v2=0, a=1.2;
  for(v2=0; v2<70; v2=v2+a) {
    a=a-0.01;
    motor[motorB] = v1;
    motor[motorC] = v2;
    wait10Msec(10);
  }
  wait1Msec(4000);
  motor[motorB] = motor[motorC] = 0;
}
```

По правилам кегельринга робот не может выходить из круга больше чем на 5 с, даже если все кегли уже выбиты. Таким образом, даже в спиральном алгоритме требуется использование датчика освещенности. Оптимальное решение — при наезде на ограничительную черную линию прекратить наращивание радиуса дуги, по которой движется робот, и переключиться на движение по линии с помощью датчика. Для этой цели будет использован цикл с условием: пока освещенность больше серого. Значение серого условно возьмем равным 45 (рис. 8.31).

Рис. 8.31. Движение по спирали с последующим движением по линии с помощью датчика освещенности.

```
task main()
{
  float v1=100, v2=0, a=1.2, k=3;
  int grey=45;
  while(SensorValue[S1]>grey) {
    v2=v2+a;
    a=a-0.01;
    motor[motorB] = v1;
    motor[motorC] = v2;
    wait10Msec(10);
  }
  while(true) {
    u=(SensorValue[S1]-grey)*k;
    motor[motorB] = v1+u;
    motor[motorC] = v2-u;
    wait1Msec(1);
  }
}
```

В последнем алгоритме, забегая вперед, мы коснулись темы движения по линии, которая будет подробно описана в следующем разделе.

Состязания «Кегельринг» уже несколько лет проводятся в России и находят все больше поклонников, поскольку отличаются простотой и доступностью. Разнообразные роботы (не только из Lego) выталкивают кегли из круга, возраст участников колеблется от 8 до 40 лет. Быть может, у того, кто прочел этот материал, найдется достаточно решимости подготовить своего робота и принять участие в подобных состязаниях.

Движение вдоль линии

Один датчик

Для первого опыта подойдут робот, созданный для задания «Танец в круге», и то же поле — черная окружность на белом фоне. Если уже все готово для изготовления полноценного поля для траектории, можно обратиться к разделу «Поле» в конце этой части. Единственная поправка: датчик освещенности следует выдвинуть немного вперед, чтобы он образовывал вместе с ведущими колесами равносторонний или хотя бы равнобедренный прямоугольный треугольник (рис. 8.32).

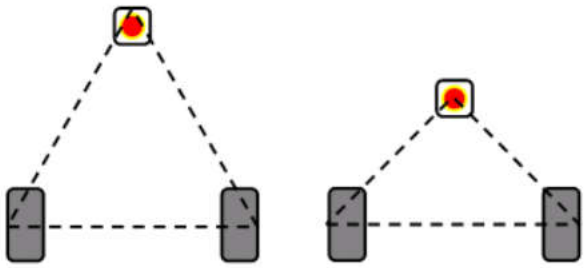


Рис. 8.32. Варианты расположения датчика освещенности относительно ведущих колес.

Задача такова: двигаться вдоль окружности по границе черного и белого. Решается элементарно применением релейного (или пропорционального) регулятора, который рассмотрен в главе «Алгоритмы управления». Только алгоритм будет записан не в виде ветвления, а с использованием блоков «Жди темнее» и «Жди светлее». Базовая конструкиця приведена на рис. 8.33—8.35, а простейшая программа для начинающих — на рис. 8.36. Без модификаторов предполагается, что датчик освещенности подключен к первому порту, а на моторы подается максимальная мощность.

Рис. 8.33. Крепление датчика освещенности к трехколесной тележке.

Рис. 8.34. Ось для регулировки высоты датчика может быть любой длины.

Рис. 8.35. Высота датчика над поверхностью поля — от 5 до 10 мм.

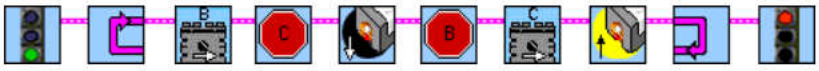


Рис. 8.36. Алгоритм движения по линии с одним датчиком освещенности.

Перед стартом ставим робота на линию так, чтобы датчик был чуть слева. По алгоритму робот плавно поворачивает направо, пока освещенность не понизится на 5 пунктов (по умолчанию). Затем поворачивает налево, пока освещенность не повысится на 5 пунктов. Движение получается похожим на «змейку».

Возможные проблемы

Перечислим трудности, которые могут возникнуть:

— робот крутится на месте, не заезжая на линию. В этом случае следует либо стартовать с другой стороны линии, либо поменять подключения моторов к контроллеру местами;

— робот проскакивает линию, не успевая среагировать. Следует понизить мощность моторов;

— робот реагирует на мелкие помехи на белом, не доезжая до черного. Надо увеличить порог чувствительности датчика (например, не на 5, а

на 8 пунктов). Вообще говоря, это число можно рассчитать. Для этого следует снять показания датчика на белом, затем на черном, вычесть одно из другого и поделить пополам. Например, (56 – 40) / 2 = 8.

Усовершенствованная программа показана на рис. 8.37.

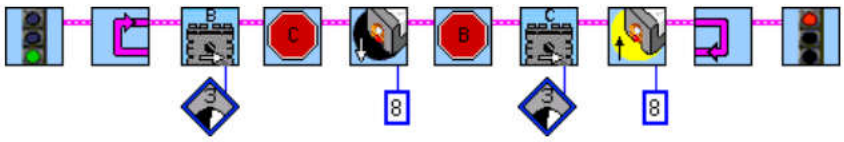


Рис. 8.37. Алгоритм движения по линии с одним датчиком освещенности: понижена скорость, увеличена разность между черным и белым.

Более устойчиво алгоритм работает, если использовать моторы с управлением скоростью –100...100. В этом случае есть возможность отрегулировать плавность поворота в соответствии с кривизной линии (рис. 8.38).

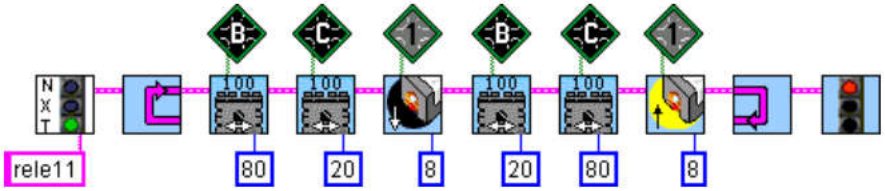


Рис. 8.38. Алгоритм движения по линии с одним датчиком освещенности: улучшено управление моторами.

В этом алгоритме притормаживающие моторы на повороте не останавливаются полностью, а лишь понижают скорость до 20 пунктов. Это делает поворот более плавным, но может привести и к потере линии на резком повороте. Поэтому числа 80 и 20 поставлены условно, их стоит подобрать самостоятельно.

П-регулятор

И, наконец, для сравнения надо посмотреть, как будет работать П-регулятор для одного датчика. Этот пример уже приводился в главе 7. Но его стоит повторить с некоторыми дополнениями (рис. 8.39).

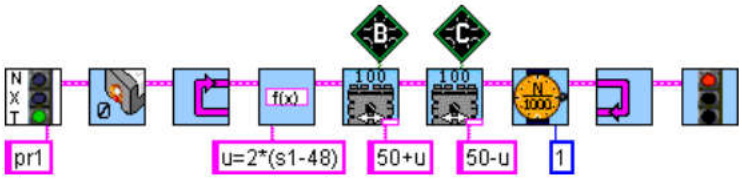


Рис. 8.39. Алгоритм движения по линии с одним датчиком освещенности на пропорциональном регуляторе.

Число 48, использованное в формуле управления *u*, — это среднее арифметическое показаний датчика освещенности на черном и на белом, например (40 + 56) / 2 = 48. Однако показания датчиков часто меняются по разным причинам: другая поверхность, изменение общей освещенности в помещении, небольшая модификация конструкции и т.п. Поэтому имеет смысл научить робота самостоятельно вычислять среднее арифметическое, т. е. значение границы белого и черного.

Есть несколько способов выполнить калибровку датчика. В простейшем случае вместо вычисления среднего арифметического просто понижается значение белого. Смысл способа в том, что робот снимает показания на белом, вычитает из него некоторое предполагаемое значение и полученное число считает границей белого и черного. Например, 56 – 7 = 49 можно считать значением серого (рис. 8.40).

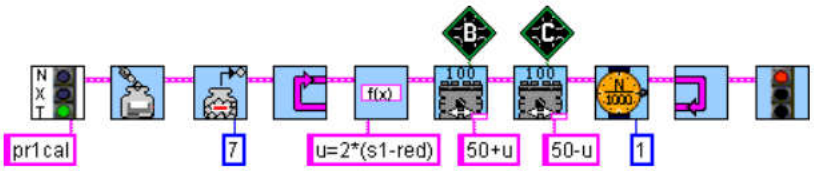


Рис. 8.40. Алгоритм движения по линии с одним датчиком освещенности на пропорциональном регуляторе с предварительной калибровкой (определением значения серого).

```
task main()
{
  int u, v=50;
  float k=2;
  int red=SensorValue[S1]-7;
  while(true)
  {
    u=k*(SensorValue[s1]-red);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

По умолчанию значение освещенности с датчика на порту 1 считывается в красный контейнер, после чего оно уменьшается на число 7, и в формуле управления *u* используется уже измененное значение красного контейнера red. Если указывать все модификаторы, программа будет выглядеть так, как показано на рис. 8.41.

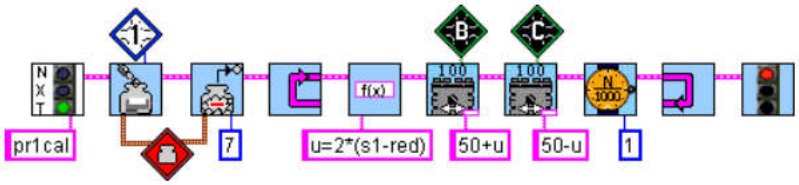


Рис. 8.41. Алгоритм движения по линии с одним датчиком освещенности на пропорциональном регуляторе — с модификаторами.

Надо иметь ввиду, что такой способ калибровки не учитывает все возможные варианты, а только экономит время на программирование и отладку. Если же времени достаточно, есть другой способ, при котором действительно производится расчет среднего арифметического показаний датчика освещенности на черном и на белом (рис. 8.42).

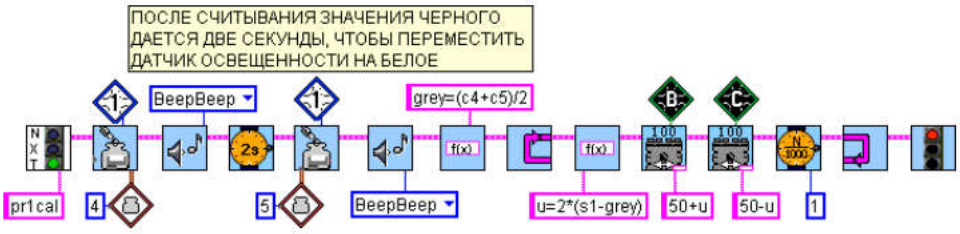


Рис. 8.42. Алгоритм движения по линии с одним датчиком освещенности на пропорциональном регуляторе с расчетом значения серого.

```
task main()
{
  int u, v=50;
  float k=2;
  int c4=SensorValue[S1];
  PlaySound(soundBeepBeep);
  wait1Msec(2000);
  int c5=SensorValue[S1];
  PlaySound(soundBeepBeep);
  int grey=(c4+c5)/2;
  while(true)
  {
    u=k*(SensorValue[S1]-grey);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

Предложенный алгоритм обладает некоторым неудобством: при запуске потребуется быть внимательным и не пропустить звукового сигнала, после которого робота надо переместить так, чтобы датчик освещенности оказался над белым полем. Понятно, что в начале следует

поместить робота точно над черной линией. В контейнере с номером 4 (обозначается *c*4) будет сохранено значение черного, в контейнере с номером 5 (*c*5) — значение белого. В переменную *grey* помещается значение серого, которое используется в регуляторе. Сразу после второго звукового сигнала робот начнет движение.

Калибровку можно сделать более управляемой. Для этого после каждого считывания данных необходимо вставить ожидание какого-либо внешнего события, например нажатия на датчик касания, уменьшения расстояния на ультразвуковом датчике или просто нажатия на кнопку NXT.

Рассмотрим простейший пример с дополнительным датчиком касания, подсоединенным ко второму порту. Запустить программу имеет смысл, аккуратно установив тележку датчиком освещенности над черной линией (рис. 8.43).

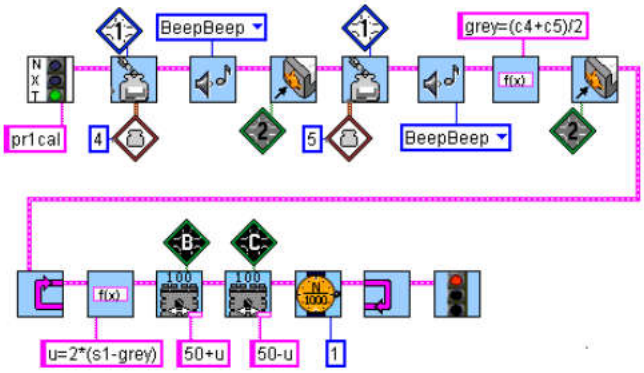


Рис. 8.43. Калибровка датчика освещенности с ожиданием касания.

```
task main()
{
  int u, v=50;
  float k=2;
  int c4=SensorValue[S1];
  PlaySound(soundBeepBeep);
  while(SensorValue[S2]==0);   // Жди, пока не нажато
  wait1Msec(100);              // Защита от залипаний
  while(SensorValue[S2]==1);   // Жди, пока нажато
  wait1Msec(100);
  int c5=SensorValue[S1];
  PlaySound(soundBeepBeep);
  int grey=(c4+c5)/2;
  while(SensorValue[S2]==0);
  wait1Msec(100);
  while(SensorValue[S2]==1);
  while(true)
  {
```

```
    u=k*(SensorValue[S1]-grey);
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

После первого звукового сигнала нужно переставить тележку так, чтобы датчик освещенности оказался над белым. После второго сигнала подготовиться к старту (датчик освещенности на границе между черным и белым) и по нажатию кнопки стартовать.

Аналогичный опыт можно провести, используя датчик расстояния вместо датчика нажатия. Преимущество здесь в том, что старт робота будет осуществляться бесконтактно. Это поможет стартовать в точно выбранном положении. Только надо быть внимательным и несвоевременно не провести рукой возле датчика расстояния (рис. 8.44).

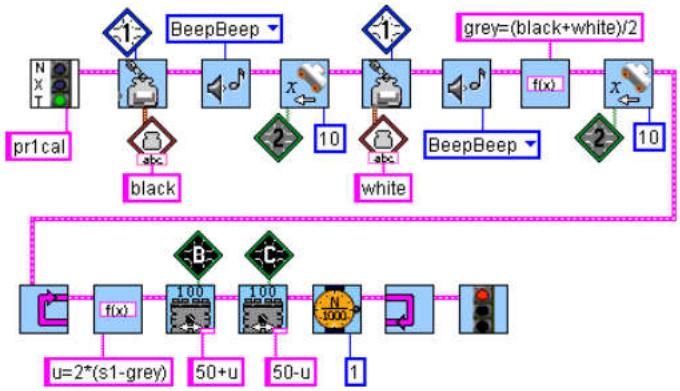


Рис. 8.44. Калибровка датчика освещенности с ожиданием объекта (руки).

В примере использованы именованные контейнеры (black и white), которые по сути являются переменными, как в обычном языке программирования. Обратите внимание, что звуковой сигнал между двумя ожиданиями изменения расстояния способствует тому, чтобы робот не среагировал дважды подряд на одно приближение руки.

Полученный опыт стоит применить для окончательного выталкивания кеглей из круга, если таковые еще остались на границе. Доработайте самостоятельно алгоритм, приведенный на рис. 8.31.

Поле для следования по линии

Более интересную траекторию, чем окружность, стоит сделать самостоятельно на светлой поверхности достаточно большой площади с помощью той же черной изоленты. В качестве поверхности подойдет лист фанеры или оргалита, обратная сторона листа линолеума, белая

клеенка и многое другое. Размеры поля желательно делать не меньше, чем 100 × 150 см. При разметке траектории следует учесть отступ от линии до края поля не менее 20 см, чтобы колеса робота не съезжали с трассы во время движения.

Имея определенный навык, можно наклеить изоленту так, что получится замкнутая кривая. Если не получается с одним куском изоленты, смело пользуйтесь ножницами, чтобы изгибы с малым радиусом кривизны составить из нескольких кусочков. Для начала не стоит рисовать слишком резких поворотов. Линию можно составить как из одной, так и из двух и даже трех полос изоленты. Тогда роботу будет легче ориентироваться и не съехать с курса. Помимо изоленты может быть использована матовая черная самоклеющаяся пленка. И, наконец, оптимальное решение — печать графического файла на баннерной ткани. Стоимость такой печати обычно не превосходит 400 руб. за 1 м^2. Небольшое поле для движения по линии приведено на рис. 8.45.

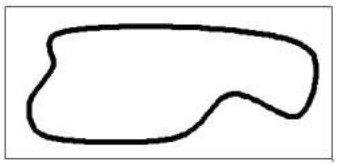


Рис. 8.45. Пример самодельного поля для движения по линии.

На рис. 8.46 приведен пример траектории для состязаний по правилам Открытого турнира на кубок Политехнического музея (г. Москва). Ширина линии составляет 5 см, а минимальный радиус кривизны — 30 см. Актуальные регламенты состязаний размещены на сайте *http://railab.ru*. Регламент состязаний «Гонки по линии» и само поле в векторном формате можно найти на сайте *http://myrobot.ru*[1].

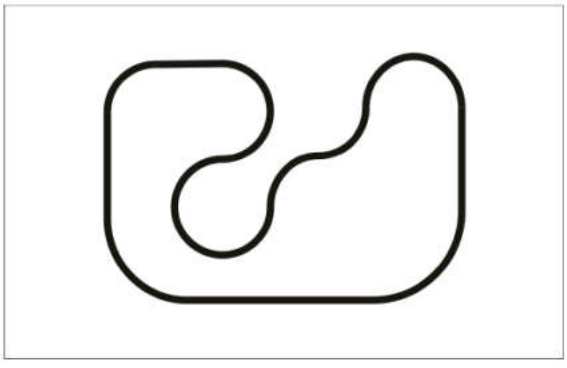


Рис. 8.46. Поле для состязаний «Гонки по линии».

[1] http://myrobot.ru/sport/index.php?n=Reglaments.LineFollowing

Два датчика

Счастливец, которому удалось раздобыть второй датчик освещенности, может построить гораздо более эффективный алгоритм движения по линии. Перекрестки и сложные повороты становятся преодолимыми для такого робота. Но, как и везде, придется столкнуться с проблемой: чем выше скорость, тем ниже надежность. Поэтому начинать надо с малых скоростей и, добившись стабильности, понемногу их увеличивать.

Калибровка и релейный регулятор

Для движения по линии датчики следует расположить на такой ширине, чтобы линия помещалась точно между ними (рис. 8.47). Возможно поставить и шире, стоит поэкспериментировать. Высота датчиков над уровнем пола — от 10 мм и более.

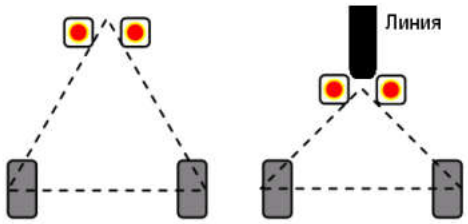


Рис. 8.47. Расположение двух датчиков освещенности.

Один из вариантов крепления датчиков дан на рис. 8.48.

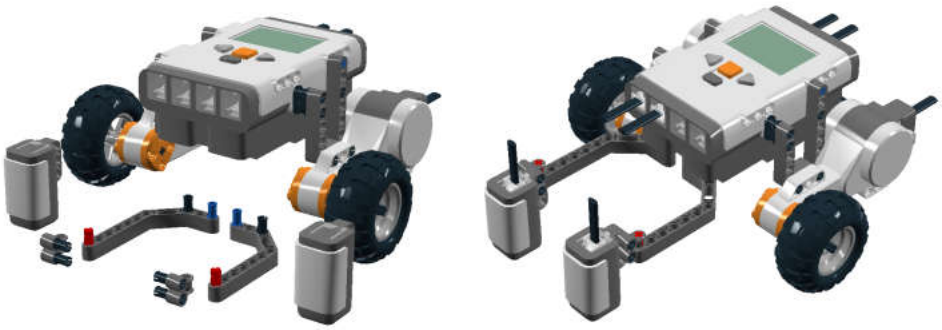


Рис. 8.48. Крепление двух датчиков на изогнутых балках.

Если робот готов, сразу применим уже знакомые алгоритмы.

Начать стоит с релейного регулятора. Он может пригодиться и впоследствии. Уже знакомый нам простейший алгоритм, в который включена автоматическая калибровка двух датчиков на белом, показан на рис. 8.49.

В переменные s1value и s2value помещаются значения соответственно левого и правого датчиков. Считаем, что левый мотор — B, правый — C. Используя опыт калибровки робота с одним датчиком, постройте калибровку с двумя датчиками самостоятельно. Понятно, что в переменных s1value и s2value должны оказаться средние значения на сером, и считывать значения с двух датчиков можно практически одновременно, т.е. без задержки.

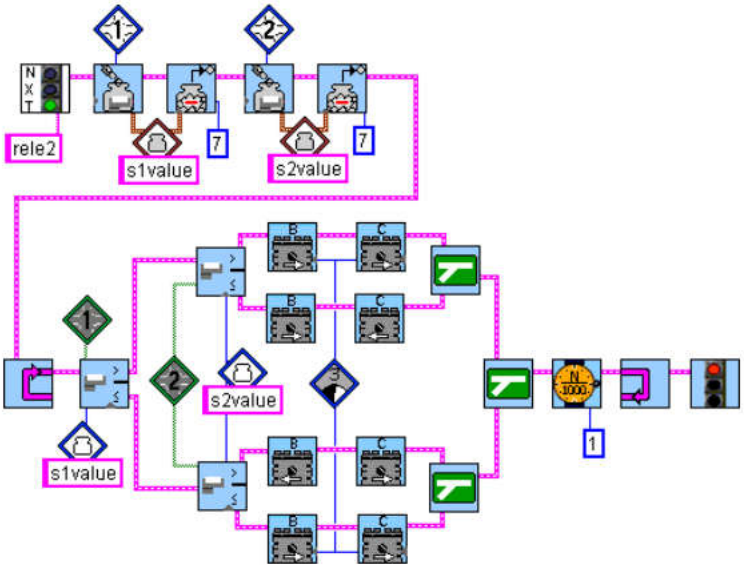


Рис. 8.49. Алгоритм движения по линии на релейном регуляторе с калибровкой двух датчиков освещенности на белом.

Пропорциональный регулятор с калибровкой

Самое интересное движение начнется, если включить П-регулятор. Калибровка для него чрезвычайно проста: достаточно запомнить показания датчиков на белом (рис. 8.50). Далее, если робот не отклоняется от этих значений, то и движется прямолинейно.

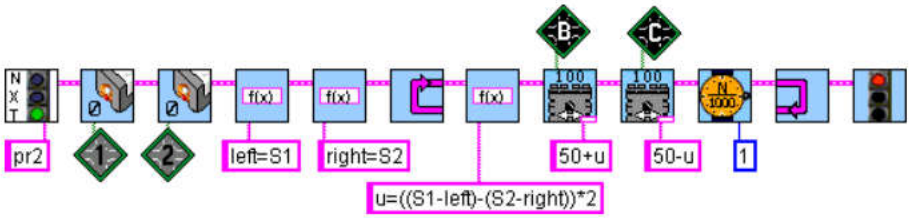


Рис. 8.50. Алгоритм движения по линии на П-регуляторе с калибровкой двух датчиков освещенности.

Инициализация датчиков перед считыванием значений, произведенная в этом алгоритме, может дать более точные показания. Сразу после считывания значений робот стартует. Но если в начальный момент он поставлен немного криво, это может привести к тому, что в ходе всего движения будет прижиматься к линии одним из датчиков. Поэтому калибровку стоит произвести не над линией, а немного в стороне и с небольшой задержкой, чтобы переместить тележку на старт.

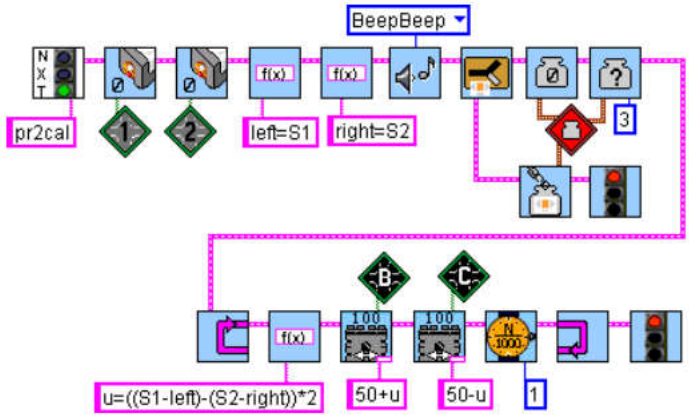


Рис. 8.51. Алгоритм движения по линии с использованием П-регулятора. Ожидание повторного нажатия оранжевой кнопки после сигнала.

В данном примере (рис. 8.51) введено ожидание нажатия оранжевой кнопки NXT. Обработка кнопок производится в специальной повторяющейся параллельной задаче, и число 3, которое записывается в контейнер в случае нажатия, соответствует оранжевой кнопке. Далее приведена аналогичная программа на RobotC. Поскольку звуковые сигналы в RobotC производятся в фоновом режиме, возможно преждевременное срабатывание ожидания нажатия оранжевой кнопки, поскольку она всегда нажимается при запуске программы. Для защиты от этой помехи в начале программы добавлено ожидание отпускания.

```
task main()
{
  int u, v=50;
  float k=2;
  while(nNxtButtonPressed==3);  // Защита от помехи
  int left=SensorValue[S1];
  int right=SensorValue[S2];
  PlaySound(soundBeepBeep);
  while(nNxtButtonPressed!=3);  // Жди, пока не нажата
  wait1Msec(100);               //   оранжевая кнопка
  while(nNxtButtonPressed==3);  // Жди, пока кнопка нажата
  while(true) {
```

```
    u=k*((SensorValue[S2]-left)-(SensorValue[S2]-right));
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

Подсчет перекрестков

Используя пропорциональный регулятор, робот может пересекать перекрестки, даже не замечая их. Однако для реального робота-автомобиля постоянно возникают ситуации, когда нужно проехать некоторое количество перекрестков прямо, а на очередном повернуть или остановиться. Тогда в алгоритме регулятора потребуются дополнительные условия.

Упростим задачу до следующей. На траектории между стартом и финишем присутствует известное число перекрестков. Старт и финиш также являются перекрестками (может быть, и Т-образными). Необходимо, проезжая каждый перекресток, издавать звуковой сигнал, а на финише остановиться.

Очевидно, что для решения задачи потребуется завести переменную-контейнер, которая будет счетчиком перекрестков. Казалось бы, все просто: увидел датчиками два черных, пиликнул и добавил в контейнер единицу. Когда контейнер превысит заданное число N, цикл заканчивается (рис. 8.52).

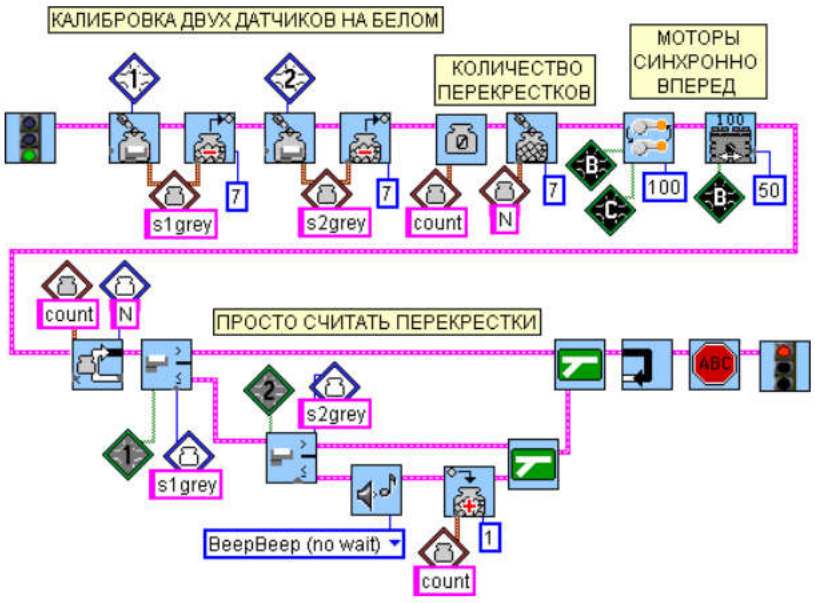


Рис. 8.52. Попытка проехать семь перекрестков.

Пусть тренировка пройдет на учебном поле, по которому надо будет просто двигаться по прямой, не используя П-регулятор. После седьмого перекрестка робот должен остановиться. Для такого опыта датчики следует поставить пошире и проверить, что колеса стоят симметрично, т.е. робот не будет сбиваться с прямой линии. Для синхронизации моторов в программу введен специальный блок.

Все готово, поехали! Но робот пиликнул несколько раз подряд и остановился, не доехав до последнего перекрестка… В чем подвох? Все очень просто. Показания датчиков считываются несколько сотен раз в секунду, скажем, 200. При скорости робота максимум 1 м/с за 2 см ширины линии он успеет увидеть ее 3—4 раза. А то и больше. Что же делать? Понятно, что требуется ввести некоторую задержку, чтобы робот успел миновать перекресток, прежде чем он снова увеличит значение счетчика. Ее можно реализовать несколькими способами (рис. 8.53).

Рис. 8.53. Задержка при увеличении счетчика: слева — с помощью блока времени, справа — с помощью звукового сигнала.

Но при движении на регуляторе нельзя ставить паузу по времени: робот ослепнет и не сможет внести коррективы в движение. Значит, пауза должна быть в параллельном процессе. В этом поможет таймер.

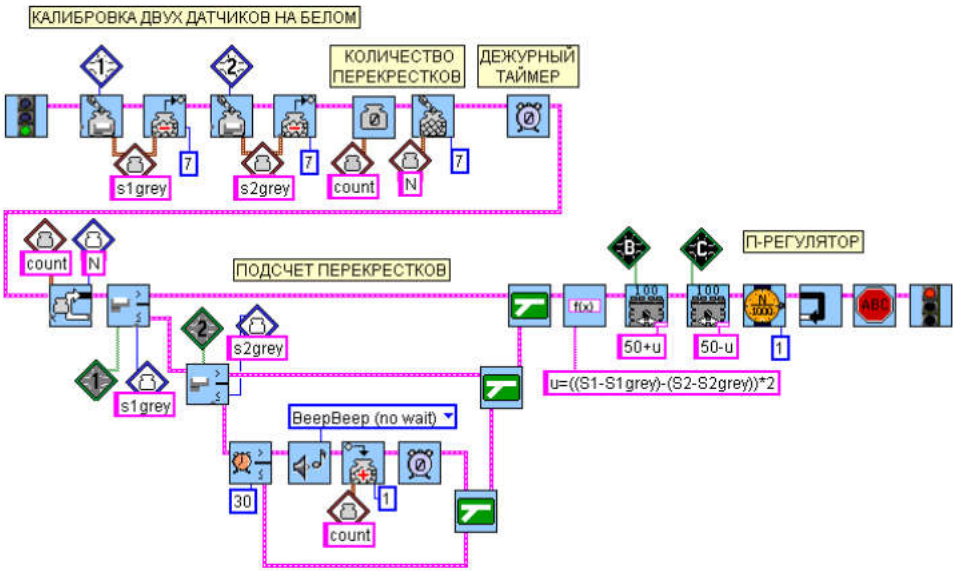


Рис. 8.54. Подсчет перекрестков со звуковым сигналом. На проезд перекрестка выделено 0.3 с.

Поскольку с учетом последних модификаций тренировочные перекрестки можно считать пройденными, подготовим программу сразу для движения по кривой линии с использованием П-регулятора (рис. 8.54).

Возможно, потребуется изменить некоторые числовые параметры: длительность проезда перекрестка, коэффициент усиления в регуляторе, понижение до серого или даже число перекрестков. Например, если робот будет стартовать непосредственно перед первым перекрестком, он может просто не заметить его. Для этого следует включать таймер заранее и обеспечить соответствующую задержку после его инициализации.

ПД-регулятор для движения по линии

Каждому хочется выиграть на соревнованиях. Для этого нужно сделать робота быстрым. Например, поставить повышающую передачу от мотора к колесам. Но увеличение скорости неизбежно приведет к понижению точности движения. В лучшем случае робота будет заносить из стороны в сторону, из-за чего общий результат значительно снизится. В худшем случае робот потеряет линию. Чтобы этого не происходило, необходимо включить контроль скорости отклонения, который позволит компенсировать действие пропорциональной составляющей с высоким усиливающим коэффициентом.

Перепишем алгоритм с использованием статической и динамический ошибки (рис. 8.55):

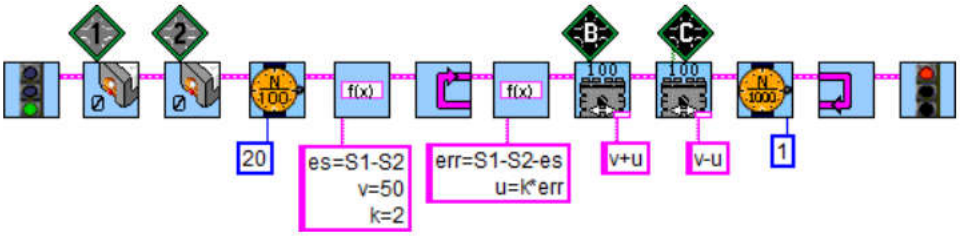


Рис. 8.55. Алгоритм движения по линии на П-регуляторе с устранением статической ошибки. Пауза перед калибровкой.

```
task main()
{
  wait1Msec(200);
  int es=SensorValue[s1]-SensorValue[s2];
  float k=2;
  int v=50, err, u;
  while(true)
  {
    err=SensorValue[s1]-SensorValue[s2]-es;
    u=k*err;
```

```
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

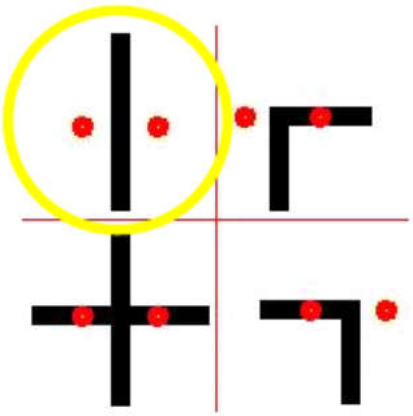


Рис. 8.56. Положение для старта робота с предварительной калибровкой.

Величина *es* представляет собой разницу показаний датчиков освещенности на однотонном белом поле (рис. 8.56) и называется статической ошибкой. В ходе движения при расчете динамической ошибки *err* статическая ошибка устраняется, что приводит к повышению точности.

Также перед автоматической калибровкой добавлена небольшая задержка в целях гарантированной инициализации датчиков. При отсутствии этой задержки первые считанные значения могут быть некорректными.

Теперь, опираясь на текущее и предыдущее значения отклонения *err*, рассчитаем скорость отклонения:

$$ve = (err - errold) / \Delta t.$$

Поскольку Δt является постоянной величиной, ее можно совместить с усиливающим коэффициентом $k2$, который управляет работой дифференциальной составляющей *ud*.

$$ud = k2 \cdot (err - errold),$$
$$errold = err.$$

Управляющее воздействие на моторы будет равно сумме пропорциональной и дифференциальной составляющих (рис. 8.57).

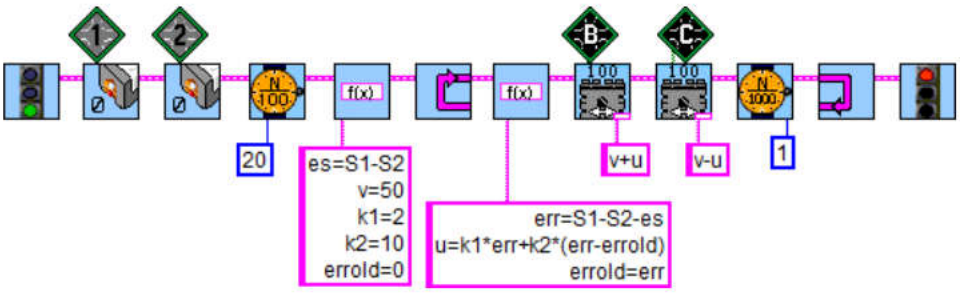


Рис. 8.57. Алгоритм движения по линии с двумя датчками на ПД-регуляторе.

В программе на RobotC необходимо предусмотреть ограничение мощности, подаваемой на моторы, поскольку слишком большие числа могут привести к непредсказуемым последствиям:

```
task main()
{
  wait1Msec(200);
  int es=SensorValue[s1]-SensorValue[s2];
  float k1=2, k2=10;
  int v=50, u, err, errold=0, mB, mC;
  while(true)
  {
    err=SensorValue[s1]-SensorValue[s2]-es;
    u=k1*err+k2*(err-errold);
    errold=err;
    mB=v+u;
    mC=v-u;
    if (mB>100) mB=100;
    if (mB<-100) mB=-100;
    if (mC>100) mC=100;
    if (mC<-100) mC=-100;
    motor[motorB]=mB;
    motor[motorC]=mC;
    wait1Msec(1);
  }
}
```

На скоростном роботе, скорее всего, регулятор не заработает сразу. Порядок настройки его будет таков. На первом этапе отключается дифференциальная составляющая обнулением усиливающего коэффициента $k2 = 0$. С использованием только пропорциональной составляющей робот доводится до предельной скорости, при которой он движется достаточно ровно на прямых участках и находится на грани потери линии на поворотах. Тогда можно начинать постепенно увеличивать коэффициент $k2$.

Скоростной робот с передачей

Эффективность ПД-регулятора будет очевидна на роботе с повышающей передачей. С помощью преложенных алгоритмов можно стабилизировать движение с передаточным отношением 3/5 или даже 1/2 (т. е. с увеличением скорости в 2 раза). При использовании колес диаметром 82.5 мм, имеющихся в ресурсном наборе 9695, робот достигает скорости 1 м/с. Это хороший результат, но придется потрудиться с настройкой коэффициентов.

Важной характеристикой конструкции является расположение центра тяжести робота над ведущими колесами. Это увеличит его манев-

ренность и повысит точность выполнения алгоритма управления за счет снижения потерь на проскальзывании колес. Однако в такой конструкции возникает необходимость в двух волокушах: спереди и сзади.

Побочный эффект высокой чувствительности — возможные заносы робота влево-вправо. Воизбежание этого можно поставить датчики освещенности несколько шире линии.

Приступим к конструированию (рис. 8.58-8.67).

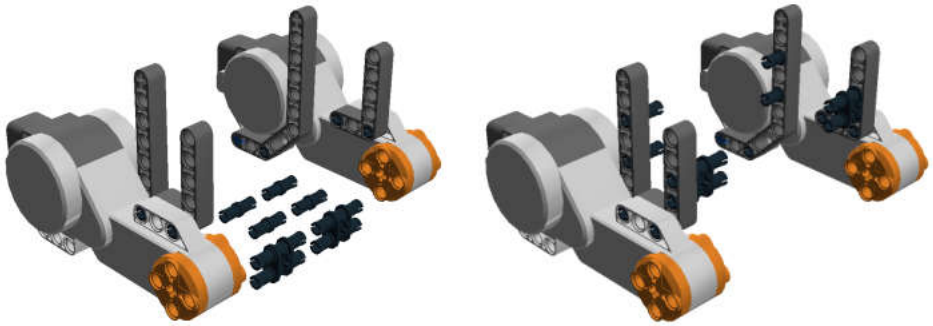


Рис. 8.58. Вначале конструкция несколько напоминает стандартную тележку.

Первые шаги (рис. 8.58) напоминают сборку стандартной трехколесной тележки (рис. 3.73), инструкция по построению которой дана в главе 3.

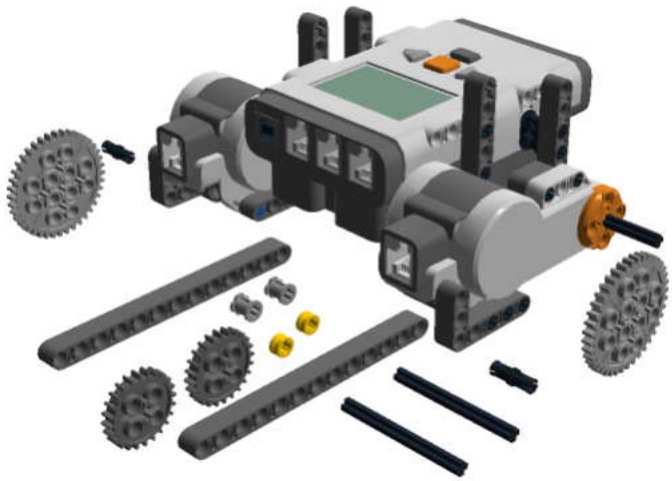


Рис. 8.59. NXT устанавливается на два модуля ниже, чем обычно.

Как настоящий болид, скоростной робот должен быть приземистым. Небольшое изменение конструкции позволит установить NXT на два модуля ниже (рис. 8.59). Следующий этап — установка передачи.

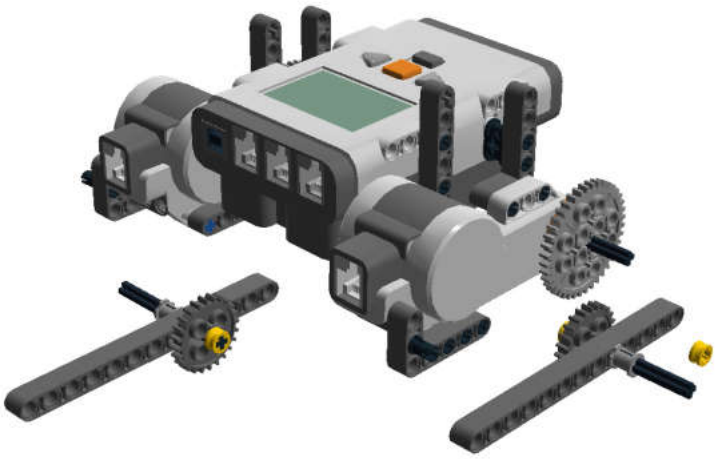


Рис. 8.60. Установка передачи с отношением $i = 3/5$ на 6-модульных осях.

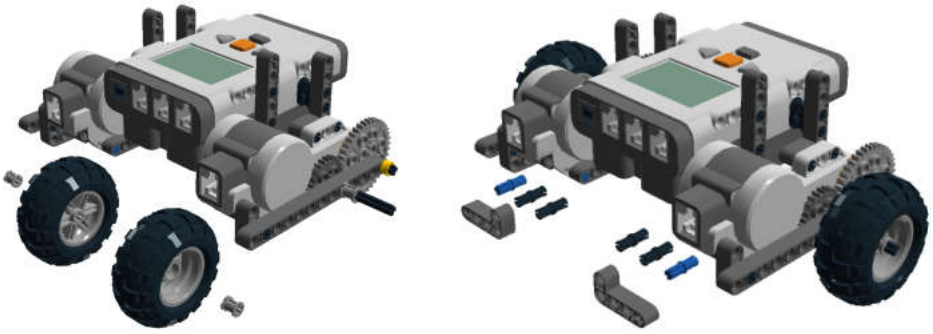


Рис. 8.61. Следующий этап после установки ведущих колес — волокуши.

Рис. 8.62. Задняя волокуша может быть установлена по аналогии с вариантами, показанными на рис. 3.82-3.85, или даже проще.

Рис. 8.63. Подвижное колесо спереди будет совмещено с креплением датчиков.

Рис. 8.64. Для установки датчиков использованы 5-модульные оси, чтобы приподнять их над поверхностью.

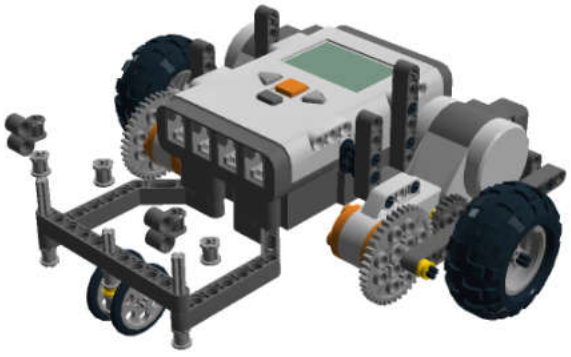


Рис. 8.65. Поперечная балка будет основой для подвижного колеса.

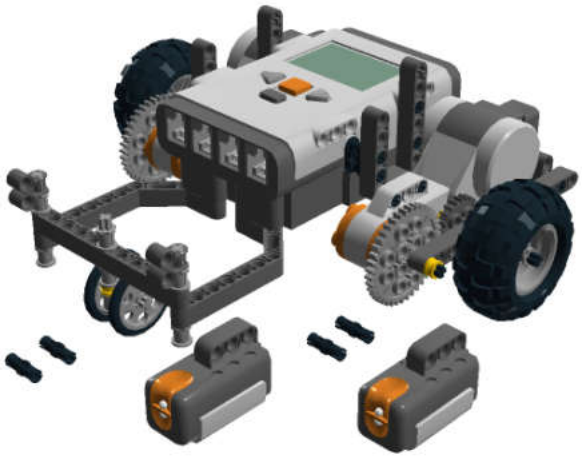


Рис. 8.66. Датчики крепятся традиционным способом с помощью штифтов и фиксаторов.

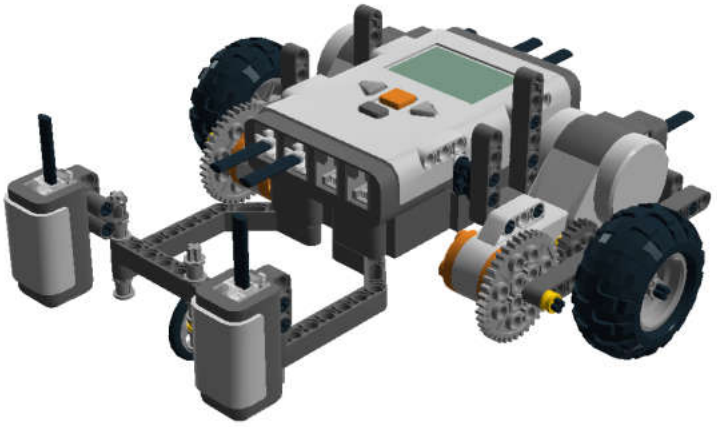


Рис. 8.67. Тележка с передачей 3/5 готова.

Особенность расположения переднего и заднего подвижных колес такова, что робот основной своей массой должен опираться на ведущие колеса и лишь немного — спереди и сзади. Поэтому следует отрегулировать их высоту, заменяя втулки.

В алгоритме также потребуются некоторые дополнения. При пробном запуске робот начинает движение в обратном направлении. Это результат применения одной ступени передачи. Для возвращения работы алгоритма в нормальный режим достаточно в начале программы установить реверс моторов. В Robolab используется команда Flip Motor Orientation (можно без параметров) из палитры NXT. В RobotC это можно сделать двумя путями: с помощью команды bMotorRe-

flected[]=true в тексте программы или через меню Motor and Sensors setup, установив флажки в поле Reversed напротив нужных моторов.

Второе дополнение касается контроля скорости. Как известно, даже опытные гонщики на поворотах притормаживают. А чем для нашего робота характеризуется поворот? Очевидно, увеличением абсолютного значения управляющего воздействия. Таким образом, скорость можно регулировать через модуль управляющего воздействия:

$$v = vmax - |u| \cdot kv,$$

где $kv < 1$ — понижающий коэффициент.

Регулирование скорости обеспечит защиту от потери линии и сделает движение более стабильным:

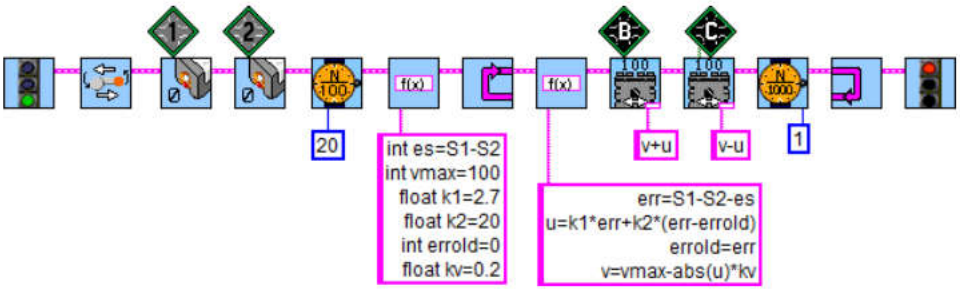


Рис. 8.68. Алгоритм движения по линии для робота со скоростной передачей.

```
task main()
{
  bMotorReflected[motorB]=true;
  bMotorReflected[motorC]=true;
  wait1Msec(200);
  int es=SensorValue[s1]-SensorValue[s2];
  float k1=2.5, k2=20, kv=0.2;
  int v, vmax=100, u, err, errold=0, mB, mC;
  while(true) {
    err=SensorValue[s1]-SensorValue[s2]-es;
    u=k1*err+k2*(err-errold);
    errold=err;
    v=vmax-abs(u)*kv;
    mB=v+u;
    mC=v-u;
    if (abs(mB>100)) mB=sgn(mB)*100;
    if (abs(mC>100)) mC=sgn(mC)*100;
    motor[motorB]=mB;
    motor[motorC]=mC;
    wait1Msec(1);
  }
}
```

Берегите пальцы от попадания между шестеренками, когда будете ловить этого робота! Однако еще более шустрой будет модель с передаточным отношением $i = 1/2$. Основа ее конструкции предлагается на рис. 8.69—8.71.

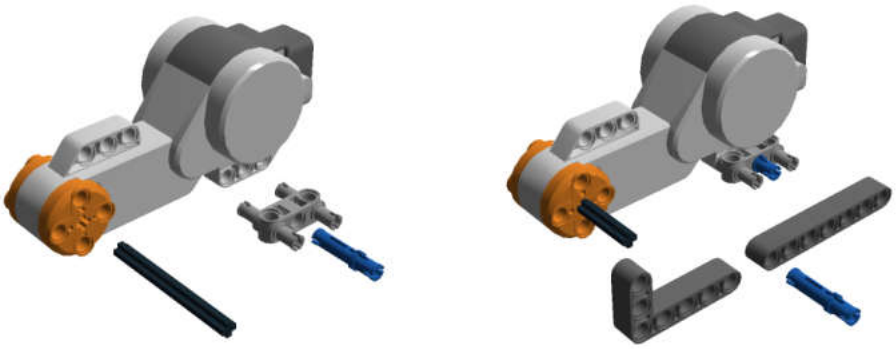


Рис. 8.69. Основа передачи с отношением $i = 1/2$ на одном из моторов.

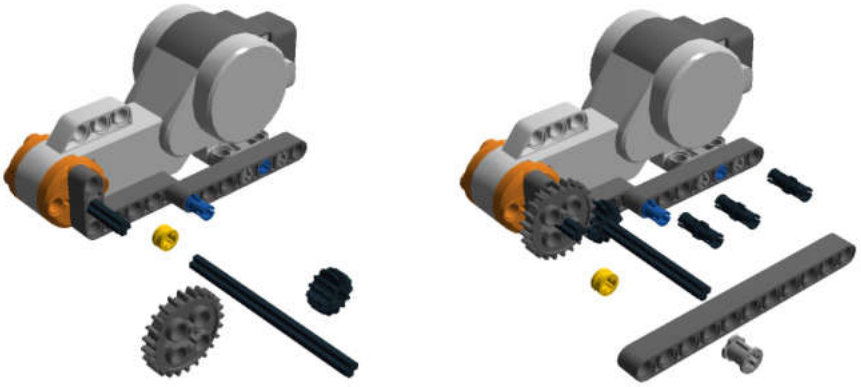


Рис. 8.70. Шестерни с 24 и 12 зубцами стыкуются с помощью угловой балки.

Рис. 8.71. Основа для построения робота готова.

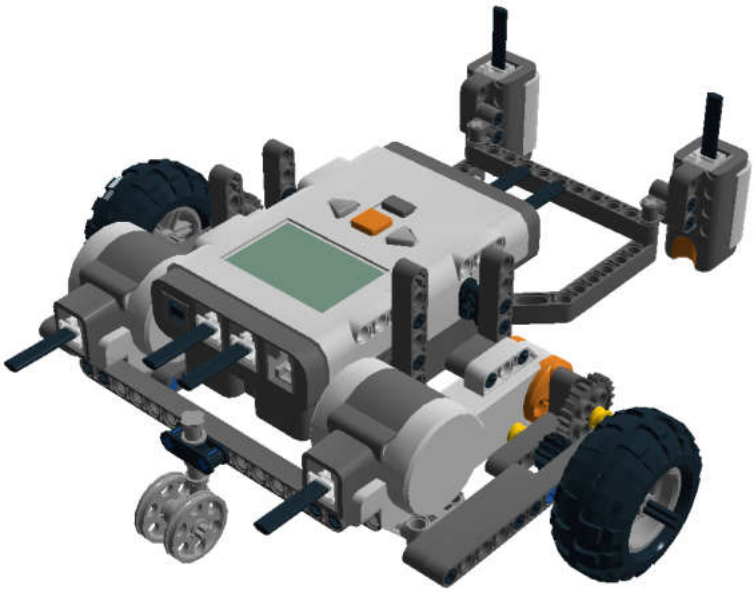


Рис. 8.72. Из-за смещения центра тяжести можно попробовать обойтись одной волокушей.

Собрать тележку полностью читателю предлагается самостоятельно. Крепление NXT и датчиков можно сделать по инструкции, предложенной ранее.

Расположение ведущих колес в новой модели (рис. 8.72) иное, чем в предыдущей (рис. 8.67). Потребуется снова отрегулировать высоту задней волокуши, а передняя, возможно, и не понадобится.

Последним «штрихом» к скоростной тележке можно считать установку больших колес диаметром 82.5 мм. Однако стабилизировать движение такой тележки по линии шириной 5 см и радиусом скругления не менее 30 см будет затруднительно. Пожалуй, это предел возможностей известных нам алгоритмов.

В последние годы в России наблюдается интересное соперничество роботов разных типов в гонках по линии по правилам Открытого турнира на кубок Политехнического музея. В нем участвуют аналоговые BEAM-роботы, самодельные роботы на микроконтроллерах, Лего-роботы на базе NXT и даже устаревшие Лего-роботы на базе RCX. Как ни странно, на состязаниях победы достаются всем типам, что добавляет азарта в этом замечательном виде робоспорта.

Слалом

Движение робота по линии может сопровождаться объездом препятствий. Название состязаний «Слалом по линии», правила которых размещены на сайте *http://myrobot.ru*, произошло от известного вида спорта, в котором спортсмен проезжает через ворота, расположенные то справа, то слева по ходу движения. В нашем случае штангой ворот или препятствием будет служить жестяная банка объемом 0.33 л, оклеенная белым ватманом, похожая на кеглю из кегельринга (рис. 8.73, слева). Для того чтобы робот ее заметил, спереди у него следует установить датчик расстояния. Размещая датчик, будьте внимательны и постарайтесь убрать провода из поля его зрения (рис. 8.73, справа).

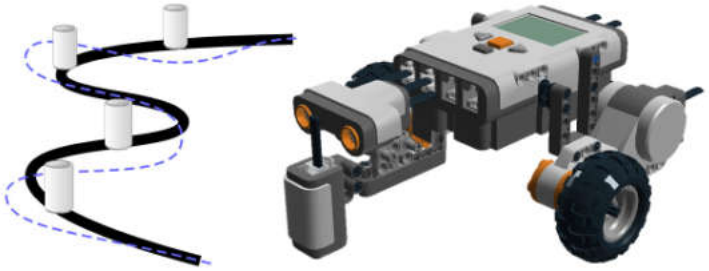


Рис. 8.73. Слалом по линии: траектория движения и конструкция робота.

Рассмотрим алгоритм для движения по линии с вызовом подпрограммы объезда с чередованием направления: то справа, то слева. Знак переменной *k* после каждого объезда изменяется на противоположный, чем обеспечивается его чередование. В этом же алгоритме реализовано плавное торможение перед объектом с помощью регулировки скорости движения *v* по расстоянию *S*2.

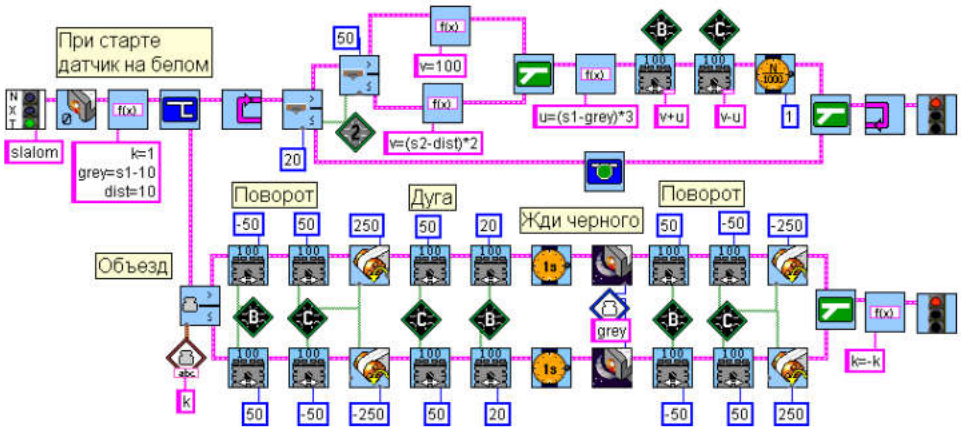


Рис. 8.74. Алгоритм движения по линии с объездом препятствий.

Инверсная линия

Задача движения по инверсной линии решается на студенческих состязаниях в рамках всероссийского научно-технического фестиваля молодежи «Мобильные роботы» им. проф. Е. А. Девянина[1]. По «шахматном» полю из черных и белых квадратов со стороной 150 см проходит линия шириной 5 см, меняющая свой цвет на противоположный в зависимости от фона. Робот должен двигаться по линии. Для школьников разработана упрощенная версия с квадратами со стороной 50 см. Тренировочный вариант такого поля приведен на рис. 8.75 слева.

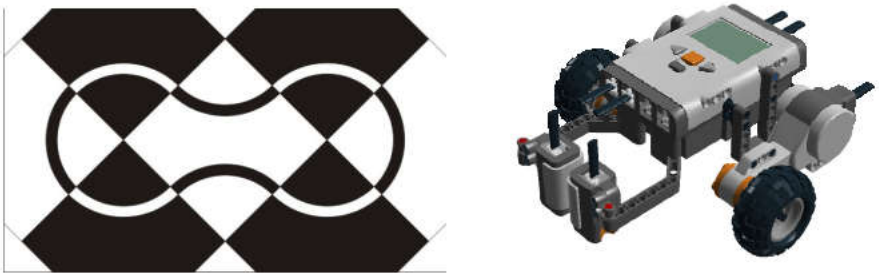


Рис. 8.75. Тренировочное поле и робот для следования по инверсной линии.

Задача довольно просто решается с использованием двух датчиков освещенности. Конструкция робота приведена на рис. 8.75 справа. Один из датчиков будет обеспечивать вдижение по линии, а второй поможет отслеживать цвет квадрата, на котором находится робот. В зависимости от показаний второго датчика будет меняться знак управляющего воздействия. Особенность робота в том, что оба датчика должны быть с одной стороны от линии. Рассмотрим алгоритм (рис. 8.76).

Рис. 8.76. Алгоритм следования по инверсной линии с двумя датчиками освещенности.

[1] Сайт фестиваля http://www.mobilerobots.msu.ru/.

Для определения знака k управляющего воздействия использована функция sgn, возвращающая знак числа в виде –1, 0 или 1. Запись получается достаточно короткой, но могут возникнуть сложности, если аргумент функции окажется равным нулю. В такие моменты робот будет ехать прямо. Воизбежание недоразумений можно использовать обычное ветвление с проверкой показаний второго датчика (рис. 8.77).

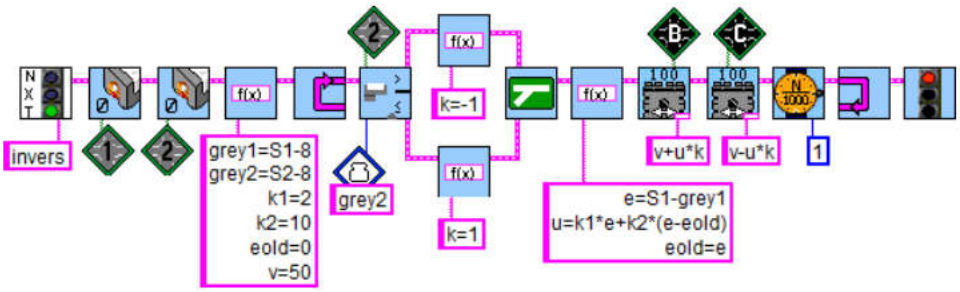


Рис. 8.77. Определение знака управляющего воздействия на инверсной линии с помощью ветвления.

```
task main()
{
  float k1=2, k2=10;
  int u, v=50, eold=0;
  int grey1=SensorValue[S1]-8;
  int grey2=SensorValue[S2]-8;
  while(true)
  {
    if (SensorValue[S2]>grey2)
      k=-1;
    else
      k=1;
    e=SensorValue[S1]-grey1;
    u=k1*e+k2*(e-eold);
    eold=e;
    motor[motorB]=v+u*k;
    motor[motorC]=v-u*k;
    wait1Msec(1);
  }
}
```

Обратите внимание, что в приведенных выше алгоритмах движения по инверсной линии использована упрощенная калибровка на белом. Для получения наибольшего эффекта следует произвести калибровку полноценно через поиск среднего арифметического.

Могут возникнуть трудности с определением правильной ориентации робота относительно линии. Тестирование можно провести на обычном поле: при правильно подключенных датчиках и моторах такой робот будет ехать справа от черной линии.

Путешествие по комнате

Маленький исследователь

Естественно, нормальная среда обитания для робота, построенного из домашнего конструктора, — это комната с мебелью. И для начала неплохо было бы научиться путешествовать по ней, по возможности не натыкаясь на предметы и не застревая.

Подходящая конструкция для такого робота — это трехколесная тележка с установленным ультразвуковым датчиком наверху (рис. 8.78). Данный датчик следует расположить строго горизонтально относительно пола, иначе любая соринка может быть воспринята как непреодолимое препятствие или, наоборот, что-то серьезное не будет замечено.

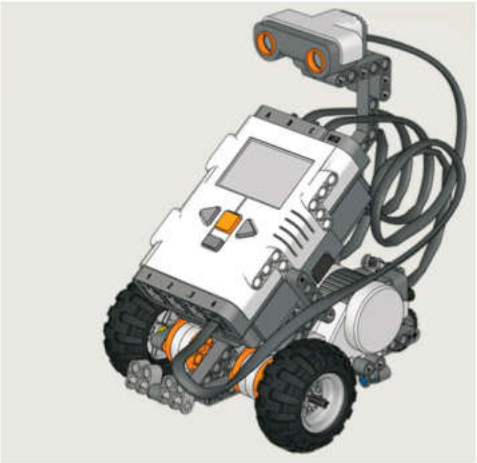


Рис. 8.78. Маленький исследователь из набора 9797 с ультразвуковым датчиком.

Более простой вариант конструкции (рис. 8.79) можно построить на основе тележки, которая рассматривалась в главе 3. Программа очень похожа алгоритм на путешествия в круге. Меняется лишь датчик (рис. 8.80).

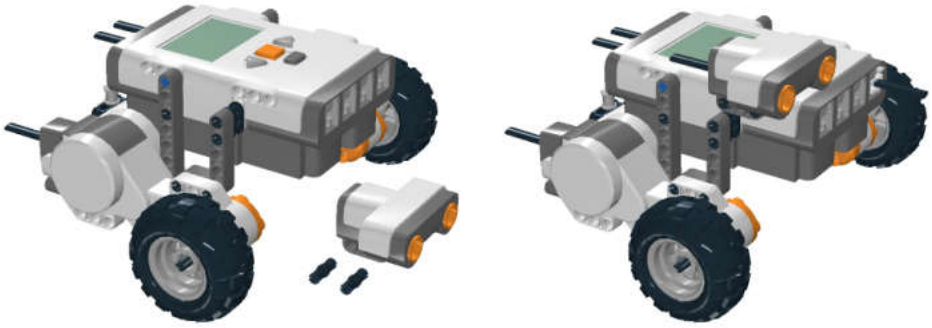


Рис. 8.79. Датчик, прикрепленный к корпусу тележки, должен смотреть строго горизонтально.

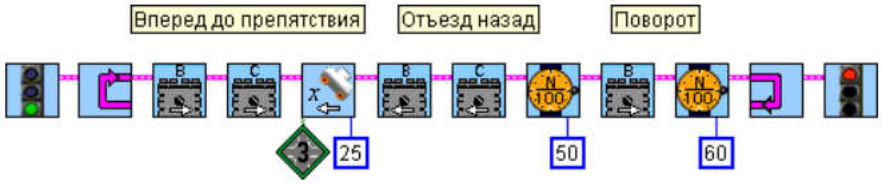


Рис. 8.80. Алгоритм путешествия по комнате.

Можно сделать несколько короче, если заменить отъезд назад с поворотом на месте одним действием: плавным поворотом задним ходом (рис. 8.81).

Рис. 8.81. Алгоритм путешествия по комнате с поворотом задним ходом.

Правда, в некоторых условиях такой поворот может привести к небольшой аварии, так что будьте с ним осторожнее. Кстати, и в первой и во второй программах следует подобрать свои параметры для расстояния до предметов и длительности поворотов.

Защита от застреваний

Присмотревшись к поведению робота повнимательнее, можно заметить, что не все предметы на пути попадают в его поле зрения. Например, если препятствие достаточно низкое, то ультразвуковой датчик его может не заметить. Или покрытая тканевой обивкой поверхность вовсе поглощает ультразвуковые сигналы, т. е. не отражает их на чувствительный элемент.

Не увидев препятствие (тапок или ножку стула), робот может застрять и будет бесконечно пытаться продолжать движение вперед. Однако если поразмыслить, можно прийти к выводу, что в комнате движение не должно быть бесконечным. Скажем, от одной стенки до другой робот может доехать за 10 с. Если за это время он не увидит ни одного препятствия, можно с уверенностью утверждать, что произошло застревание и надо предпринять экстренные меры. Что же делать? Ничего особенного. Просто отъехать назад и развернуться. Поможет в этом «сторожевой таймер» (рис. 8.82). Такие устройства применяются в микроконтроллерах и защищают их от зависаний.

Рис. 8.82. Если на сторожевом таймере «натикает» 10 с, включается защита от застреваний.

Можно заметить в нашей программе повторяющиеся группы блоков. Их стоит объединить в подпрограмму, которая будет осуществлять отъезд назад с разворотом (рис. 8.83). Таким образом, подпрограмма отъезда будет вызываться в двух случаях: 1) при наличии препятствия, 2) при срабатывании сторожевого таймера.

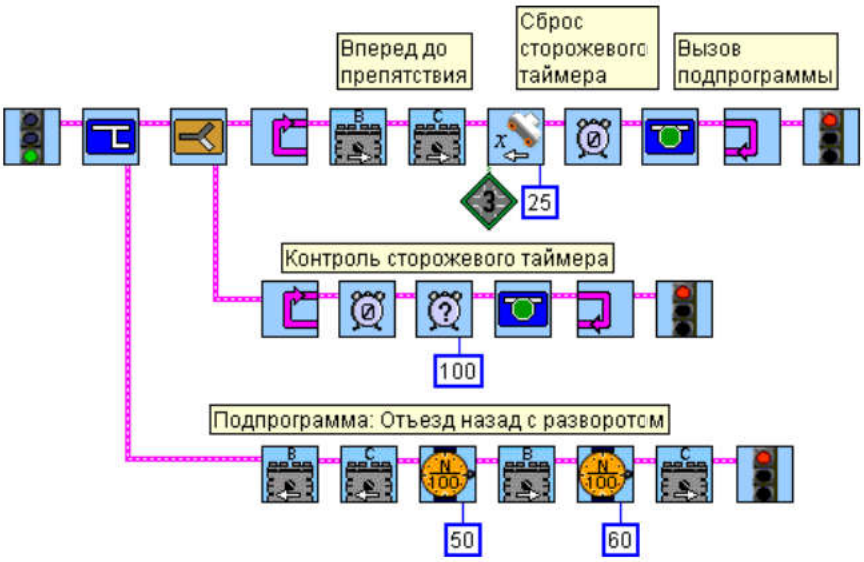


Рис. 8.83. Защита от застреваний с использованием подпрограмм.

Но в этой программе есть один существенный недостаток. Из двух параллельных задач происходит обращение к одним и тем же моторам. Если эти обращения совпадут во времени, может возникнуть непредвиденное поведение робота. В чем-то это даже интересно. Но наиболее корректная программа описана в следующем разделе.

Дополнительный датчик

Для надежности следует оснастить робота еще одним датчиком и бампером, который можно расположить достаточно низко и широко (рис. 8.84). Такой бампер с датчиком касания поможет роботу обнаружить те препятствия, которые не были замечены ультразвуком.

Рис. 8.84. Бампер с датчиком касания для маленького исследователя.

Тогда команда на разворот должна быть выполнена в нескольких случаях: когда поступил соответствующий сигнал с любого из датчиков и при застревании.

Поскольку предполагается, что тележка будет соприкасаться с препятствиями, можно понизить ее скорость во избежание аварий.

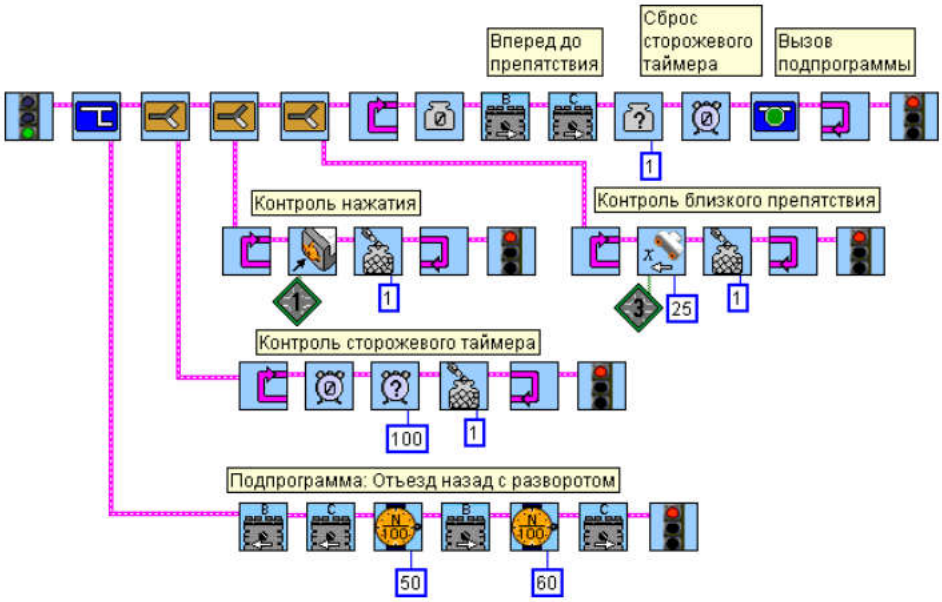


Рис. 8.85. Алгоритм путешествия по комнате со сторожевым таймером.

В предложенном алгоритме (рис. 8.85) повод для отъезда назад с разворотом — появление числа 1 в контейнере-семафоре, что возможно вследствие одного из трех событий: 1) нажатия на датчик касания, 2) появления препятствия в зоне обзора датчика расстояния и 3) критиче-

ского значения сторожевого таймера. Все три процесса контролируются из параллельных задач, а моторами управляет только главная задача.

```
int c=0;  // Глобальная переменная-семафор
void otjezd() // Подпрограмма отъезда назад
{
  motor[motorB]=-100;
  motor[motorC]=-100;
  wait1Msec(500);
  motor[motorB]=100;
  wait1Msec(600);
  motor[motorC]=100;
}
task storoj()  // Контроль сторожевого таймера
{
  while(true) {
    while (Time1[T1]<10000) wait1Msec(1);
    c=1;
  }
}
task touch() // Контроль датчика касания
{
  while(true) {
    while (SensorValue[S1]==0) wait1Msec(1);
    c=1;
  }
}
task objects() // Контроль датчика расстояния
{
  while(true) {
    while (SensorValue[S2]>25) wait1Msec(1);
    c=1;
  }
}
task main()
{
  ClearTimer(T1);
  StartTask(storoj);
  StartTask(touch);
  StartTask(objects);
  while(true)
  {
    c=0;
    motor[motorB]=100;
    motor[motorC]=100;
    while(c!=1) wait1Msec(1);
    ClearTimer(T1);
    otjezd();
  }
}
```

Аналогичную программу можно было бы создать, используя мониторинг событий Robolab. Оставляем читателю это в качестве домашнего задания.

Рассмотрим простую и эффективную конструкцию для датчика касания, которая обеспечит срабатывание практически в любом положении (рис. 8.86—8.88). Ширину бампера, который выполнен в виде оси, закрепленной на изогнутую балку, можно увеличить до ширины самого робота.

Рис. 8.86. Эффективная конструкция для датчика касания.

Рис. 8.87. Нажатие осуществляется «пяточкой» изогнутой балки 3 × 5.

Рис. 8.88. Конструкция с датчиком крепится непосредственно к NXT.

Объезд предметов

Новая конструкция

Первые шаги к объезду предметов сделаны в главе «Алгоритмы управления». Движение вдоль стены с небольшими отклонениями возможно с помощью ПД-регулятора. Однако описанный робот сможет объезжать стены только при малых отклонениях от прямой линии. При резких изгибах робот может потерять контакт со стеной и начать крутиться на месте. Эту проблему можно отчасти разрешить конструктивно.

Рассмотрим вариант, при котором на пути движения будут возникать серьезные повороты, вплоть до прямых углов. Потребуется внести модификации и в конструкцию, и в программу.

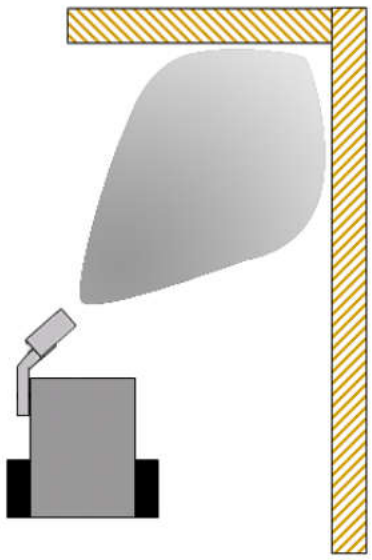


Рис. 8.89. Датчик расстояния устанавливается под острым углом к направлению движения.

Во-первых, робот должен будет смотреть не только направо, но и вперед. Ставить второй дальномер довольно затратно. Однако можно воспользоваться эффектом того, что ультразвуковой датчик имеет расширяющуюся область видимости (рис. 8.89). Это напоминает периферийное зрение человека: кое-что он может увидеть краем глаза. Учитывая это свойство, разместим датчик расстояния не перпендикулярно курсу движения, а под острым углом (рис. 8.90—8.91). Так можно «убить сразу двух зайцев». Во-первых, робот будет видеть препятствия спереди; во-вторых, более стабильно будет придерживаться курса вдоль стены, постоянно находясь на грани видимости. Таким образом без добавления новых устройств можно более эффективно использовать возможности дальномера.

Важное замечание. При старте робота его надо будет направлять датчиком строго на стену, чтобы считывание начального значения прошло без помех.

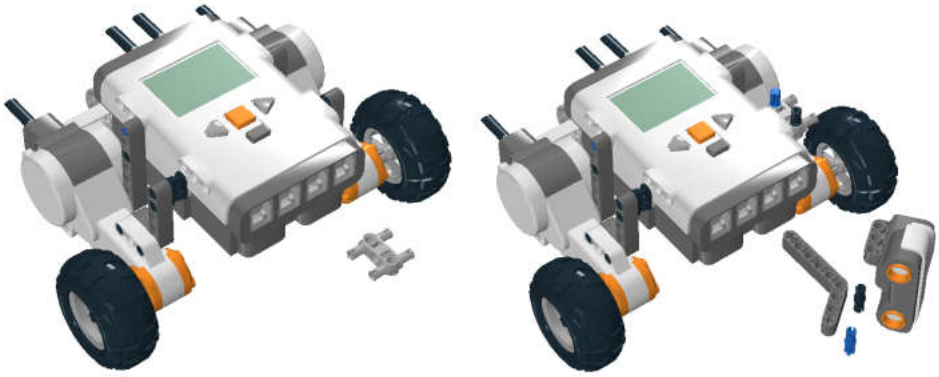


Рис. 8.90. Крепление размещается на левой стороне. Как и в первой конструкции, датчик располагается вертикально.

Рис. 8.91. Увеличенное за счет корпуса робота расстояние до стены способствует расширению области обзора.

Очевидно, что изменение конструкции влечет изменение коэффициентов регулятора $k1$ и $k2$. Обычно подбор начинается с пропорционального коэффициента при нулевом дифференциальном. Когда достигнута некоторая стабильность на небольших отклонениях, добавляется дифференциальная составляющая.

Поворот за угол

Следующим шагом необходимо ограничить реакцию робота на «бесконечность». Как известно, когда в поле видимости нет объекта, показания датчика расстояния NXT равны 250 или 255 см. Если это число попадает на пропорциональный регулятор, робот начинает кру-

титься на месте. А в ситуации, когда роботу следует завернуть за угол, именно это и произойдет.

Для объезда предметов потребуется ввести контроль показаний датчика расстояния: при резком изменении робот должен делать вывод о возможном повороте, который надо будет производить с другими коэффициентами или просто с постоянным значением управляющего воздействия.

Рассмотрим пример поворота направо «за угол» (рис. 8.92). Если робот движется на расстоянии L от стены, то и поворот, очевидно, он будет выполнять по окружности с радиусом L.

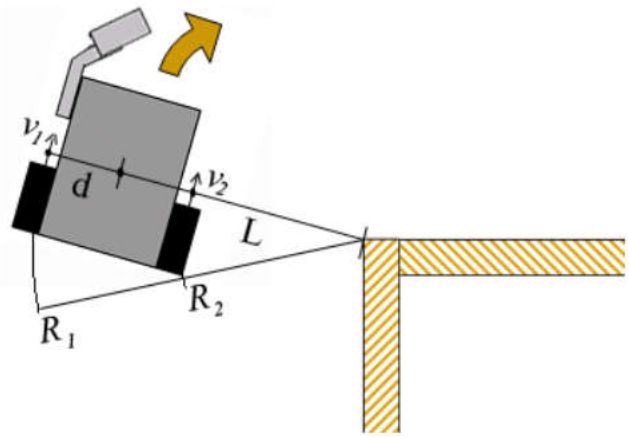


Рис. 8.92. Выполнение поворота при потере контакта со стенкой.

Нетрудно рассчитать, каким должно быть отношение скоростей колес, чтобы радиус поворота оказался равен L. Для этого достаточно измерить расстояние между передними колесами. Пусть в нашем роботе оно будет равно $k = 16$ см, а его половина $d = 16 / 2 = 8$ см. Тогда левое и правое колеса движутся по окружностям радиусов, соответственно, $R_1 = L + d$ и $R_2 = L - d$. Пройденные ими пути за единицу времени должны быть пропорциональны радиусам, следовательно, скорости точек крепления колес v_1 и v_2 связаны следующим отношением:

$$\frac{v_1}{v_2} = \frac{R_1}{R_2}.$$

Выражая скорости перемещения колес через базовую скорость v и неизвестную x, а радиусы через L, получаем следующее:

$$\frac{v+x}{v-x} = \frac{L+d}{L-d},\ vL + xL - vd - xd = vL + vd - xL - xd,\ 2xL = 2vd,$$

$$x = \frac{vd}{L},\ v_1 = v + \frac{vd}{L} = v(1+\frac{d}{L}),\ v_2 = v - \frac{vd}{L} = v(1-\frac{d}{L}).$$

Линейная скорость v пропорциональна угловой скорости колеса ω, которая в свою очередь пропорциональна мощности, подаваемой на моторы (в режиме торможения). Мы привели закон управления к стандартному виду, что позволяет задать управляющее воздействие на время поворота за угол. Таким образом, получаем расчет для управления моторами нашего робота.

```
u=v*8/L;
motor[motorB]=v+u;
motor[motorC]=v-u;
```

Когда расстояние до стены становится больше $2L$ (используем такой порог видимости), т. е. открывается поворот за угол, управляющее воздействие начинает вычисляться по приведенным формулам (рис. 8.93).

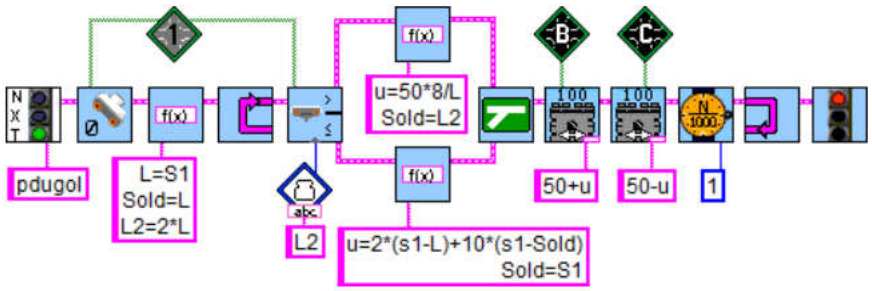


Рис. 8.93. Объезд предметов на заданном расстоянии по правилу правой руки.

```
task main()
{
  float u, k1=2, k2=10;
  int v=50, d=8, Sold, L, Snew;
  Sold=L=SensorValue[S1]; // Запомнили начальное состояние
  while(true) {
    Snew=SensorValue[S1]; // Получили показания датчика
    if (Snew>L*2) {
      u=v*d/L;
      Sold=L*2;
    }
    else {
      u = k1*(Snew-L) + k2*(Snew-Sold);
      Sold=Snew;
    }
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

В программе на RobotC добавлены переменные, чтобы сделать ее более строгой и наглядной. При этом исключена переменная $L2$, поскольку, в отличие от Robolab, здесь имеется возможность писать в условиях формулы.

Надо заметить, что мы слегка обманываем робота, подставляя ему ограниченные сверху значения. Такой алгоритм будет стабильно работать на расстоянии L от 25 до 125 см.

Фильтрация данных

Наконец, для окончательной стабилизации робота следует ввести защиту от помех. В силу особенностей работы ультразвукового датчика сигнал время от времени не попадает на глазок-приемник и в результате не всегда поступают адекватные значения, что может внести серьезные возмущения в наш алгоритм. Например, небольшая щель в стене для робота выглядит целым проемом. Поскольку требования к скорости пока что невысоки, можно несколько замедлить реакцию датчика на изменения расстояния, установив программный фильтр его значений. Простейшая реализация такого фильтра состоит в усреднении очередного показания и предыдущего отфильтрованного значения:

$$Snew = (Snew + S1) / 2.$$

Среднее арифметическое несколько сглаживает резкие скачки показаний датчика. Во всех формулах вместо S1 будет использоваться отфильтрованное значение Snew. Однако и этого может не хватить при слишком резких изменениях. Тогда следует ввести весовые коэффициенты $c1$ и $c2$ для усредненной и новой величины:

$$Snew = (c1 \cdot Snew + c2 \cdot S1) / (c1 + c2).$$

Раскрываем первые скобки:

$$Snew = \frac{c1}{c1 + c2} \cdot Snew + \frac{c2}{c1 + c2} \cdot S1.$$

Легко заметить, что сумма полученных коэффициентов равна 1. Поэтому, заменив их соответственно на $1 - a$ и a, получим

$$Snew = (1 - a)*Snew + a*S1, \text{ где } 0 \le a \le 1.$$

Подбирая значение a, можно регулировать степень фильтрации. Для начала примем $a = 0.2$. Сравнение результатов фильтрации показа-

ний датчика ультразвука в течение 1 с приведено на рис. 8.94. Синий график показывает показания датчика, пурпурный — фильтр по среднему арифметическому, желтый — фильтр с параметром $a = 0.2$.

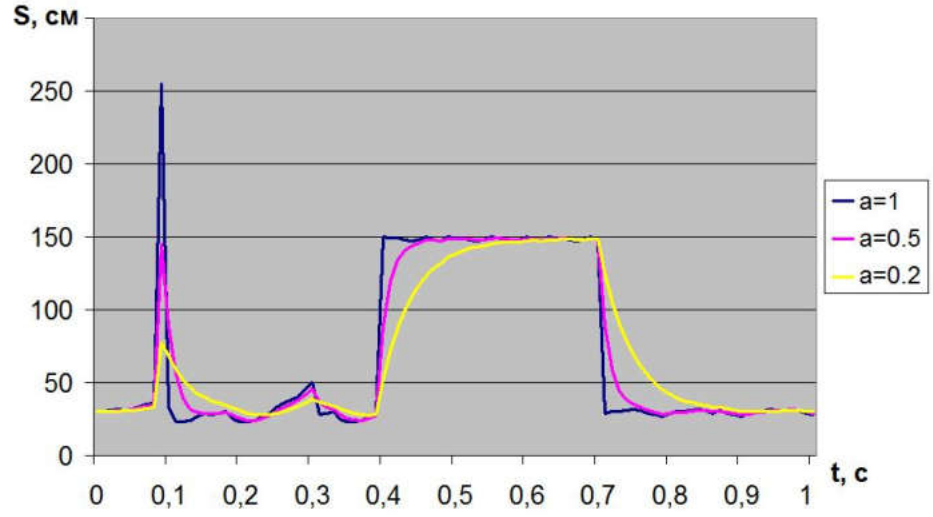


Рис. 8.94. Результаты фильтрации показаний датчика ультразвука с различными коэффициентами.

По графикам (рис. 8.94) видно, что при малых значениях параметра фильтрация происходит эффективнее и случайные скачки показаний практически нивелируются. Кроме того, более-менее постоянные высокие значения достигаются в доли секунды, причем переход становится достаточно плавным. Иначе такой алгоритм можно назвать сглаживанием.

Перепишем программы под новые отфильтрованные показания датчика (рис. 8.95). Поначалу результат может оказаться неожиданным, однако подбор коэффициентов все расставит на свои места:

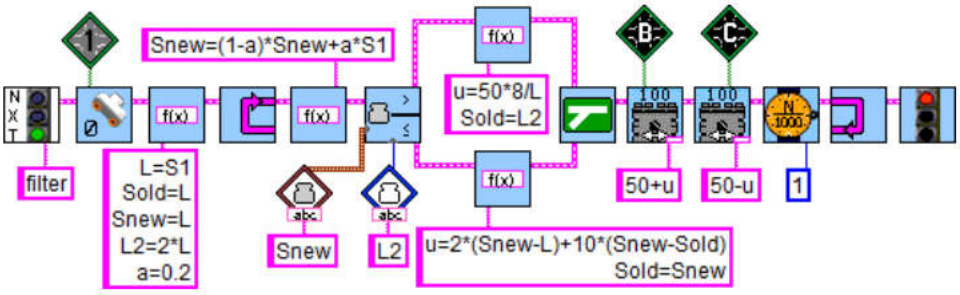


Рис. 8.95. Алгоритм движения вдоль стенки с фильтрацией данных.

```
task main()
{
  float u, k1=2, k2=10, a=0.2, Snew;
  int v=50, d=8, Sold, L;
  Snew=Sold=L=SensorValue[S1];
  while(true)
  {
    Snew=(1-a)*Snew+a*SensorValue[S1];
    if (Snew>L*2) {
      u=v*d/L;
      Sold=L*2;
    }
    else {
      u = k1*(Snew-L) + k2*(Snew-Sold);
      Sold=Snew;
    }
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
```

Фильтрация данных становится особенно актуальной, если на их основе требуется принимать решение о дальнейших действиях в долгосрочной перспективе. Например, увидев проем, остановиться или повернуть назад. Достаточно одной помехи, чтобы робот остановился не в том месте. Поэтому фильтры, хоть и затормаживают реакцию робота, но делают ее более стабильной и предсказуемой.

Роботы-барабанщики

Предыстория

Идея построить робота-барабанщика из Lego появилась в 2009 году на кружке робототехники физико-математического лицея № 239 г. Санкт-Петербурга. Ребятам она понравилась и вскоре было создано несколько моделей, которые запоминали и воспроизводили ритм, импровизировали, управлялись удаленно, играли заранее записанные мелодии и даже аккомпанировали компьютеру в проигрывании midi-файлов. Через 10 месяцев робот-барабанщик получил бронзовую медаль на Всемирной олимпиаде роботов в Южной Корее (фотография награждения команды лицея «Старый барабанщик» на задней стороне обложки). Впоследствии он много путешествовал по России от Москвы до Сибири, выступая на фестивалях, выставках и форумах.

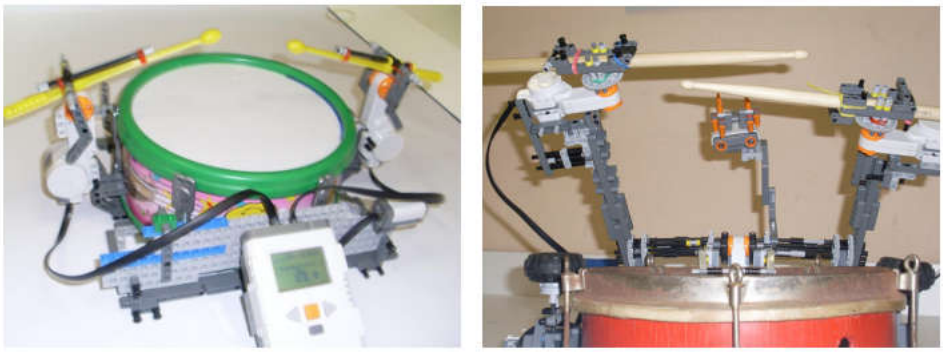


Рис. 8.96. Первые версии робота-барабанщика с игрушечным и пионерским барабанами.

Первые версии робота были сделаны целиком из деталей Lego и программировались через Robolab. Основой служил игрушечный барабан (рис. 8.96). Андроид-барабанщик был построен из пластиковых водопроводных труб (рис. 8.97), перед ним стояла целая барабанная установка, а программировался он на языках RobotC и Java.

Рис. 8.97. Андроид-барабанщик.

Алгоритмы управления роботами-барабанщиками основаны на П-регуляторах, что дает возможность повысить точность позиционирования барабанных палочек. Это описано далее.

Калибровка и удар

Конструкция для барабанщика с одной палочкой изображена на рис. 7.1. Ничего особенного, разве что балку стоит сделать подлиннее (рис. 8.98). Барабанить он будет по столу. Однако прежде чем нанести

удар, надо определиться, где именно находится поверхность «барабана». Для этого необходимо выполнить калибровку. Простейший ее вариант заключается в следующем.

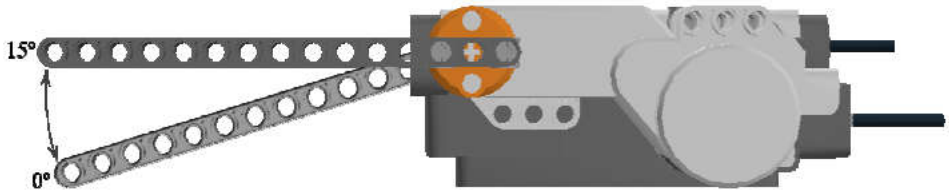


Рис. 8.98. Движения «барабанной палочки».

Робот опускает «палочку» на низкой скорости в течение достаточно большого промежутка времени, около 2 с (рис. 8.99). Поскольку скорость и мощность моторов взаимосвязаны, то, упершись в поверхность, палочка не причинит роботу никакого вреда и просто остановится. Полученное положение мотора запоминается как нулевое. Калибровка произведена.

Рис. 8.99. Простейшая импровизация одной палочкой.

Удар будет представлять из себя поднятие палочки в абсолютное положение под углом 15 градусов (замах) и опускание в нулевое положение (удар). Однако, учитывая люфт моторов Lego, а также инерцию их движения, следует прекращать опускание не в нулевом положении, а значительно раньше, например при достижении угла 10 градусов. Это позволит сократить длительность соприкосновения с поверхностью и воспользоваться энергией отскока, что значительно ускорит движение палочек. Робот получит возможность воспроизводить довольно быстрый ритм.

Управление с помощью датчика

Заменив ожидание таймера ожиданием нажатия датчика касания, можно получить управляемую игрушку (рис. 8.100). Человек будет нажимать на кнопку, а робот — послушно выполнять удар.

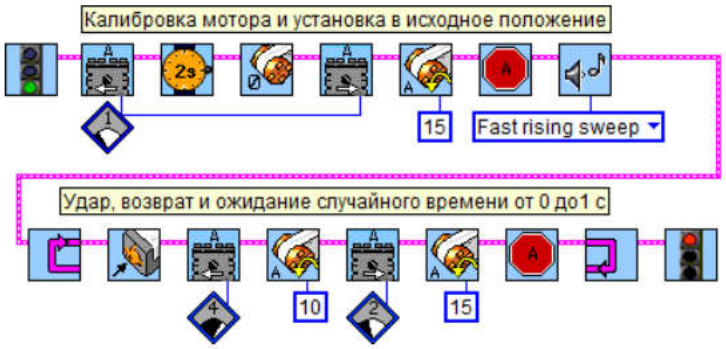


Рис. 8.100. Управление палочкой с помощью датчика касания.

В этом алгоритме стоит естественная защита от слишком частого нажимания датчика: робот будет барабанить с той скоростью, с которой успевает (рис. 8.101).

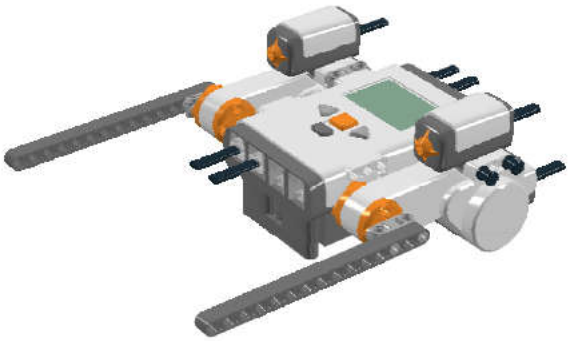


Рис. 8.101. Датчики можно прикрепить на робота или вынести отдельно.

Чтобы настроить управление двумя палочками, потребуется поместить калибровку второго мотора и удар в параллельный процесс. Вытянем алгоритм в цепочку и сдублируем его. Единственное незначительное изменение — это изменение типа и мощности управления моторами при калибровке. Поскольку уже два двигателя будут прижимать палочки к полу, то во избежание поднятия всего робота стоит понизить их мощность до 5—10 %.

Дублируя часть алгоритма, постарайтесь внимательно заменить все команды мотору A на команды мотору B (рис. 8.102).

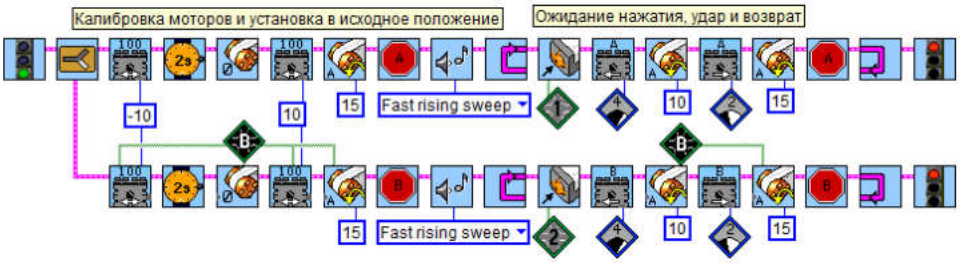


Рис. 8.102. Алгоритм управления палочками с помощью двух датчиков.

Рис. 8.103. Конструкции для управления роботом-барабанщиком: слева — на датчиках касания, справа — на гироскопических датчиках.

Ребята из команды «Старый барабанщик» сконструировали специальную перчатку с закрепленным на ней двумя датчиками касания (рис. 8.103, слева). Быть может, и читатель придумает нечто подобное. А при наличии пары гироскопических датчиков не составит труда настроить управление роботом с помощью настоящих барабанных (рис. 8.103, справа).

Создаем свой ритм

Вернемся к одномоторному барабанщику. Преобразуем удар в процедуру, которая вызывается через определенные промежутки времени. Задавая длительность этих промежутков, создадим ритмический рисунок (рис. 8.104). Лучше будет использовать длительности, кратные 0.2 секунды: 20 сотых, 40 сотых, 60 сотых и т. д. Если задать чересчур малую длительность, то либо робот не успеет произвести удар, либо удар будет слишком тихим. Эту особенность можно использовать для регулирования громкости.

Очевидно, что для повышения точности процесса надо поднимать палочку не до удара, а сразу после него.

Рис. 8.104. Создаем собственный ритимический рисунок.

Удар обладает некоторой длительностью, и ее тоже надо учитывать при определении пауз. Удобнее всего это сделать с помощью системного таймера. Команда «Жди таймер» будет предшествовать каждому удару, а очередное ожидаемое значение задаваться непосредственно перед ней (рис. 8.105).

Рис. 8.105. Ритмический рисунок соответствует реальному времени.

В приведенном алгоритме много умолчаний: на энкодере используется мотор A, значение ожидания обнуляется и добавляется в красный контейнер, да и таймер тоже используется по умолчанию красный.

Барабанщик с двумя палочками

Для управления двумя палочками следует сделать их независимыми друг от друга по времени, т. е. поместить управление ими не просто в разные процедуры, но и в параллельные задачи (рис. 8.106). Тогда вторая палочка сможет приступать к удару, не дожидаясь окончания удара первой.

При задании длительностей пауз надо учитывать длительность ударов. Минимальная пауза для отдельного мотора — примерно 0.2 с. Значит два мотора смогут работать с минимальной паузой 0.1 с, т.е. наносить до 10 ударов в секунду.

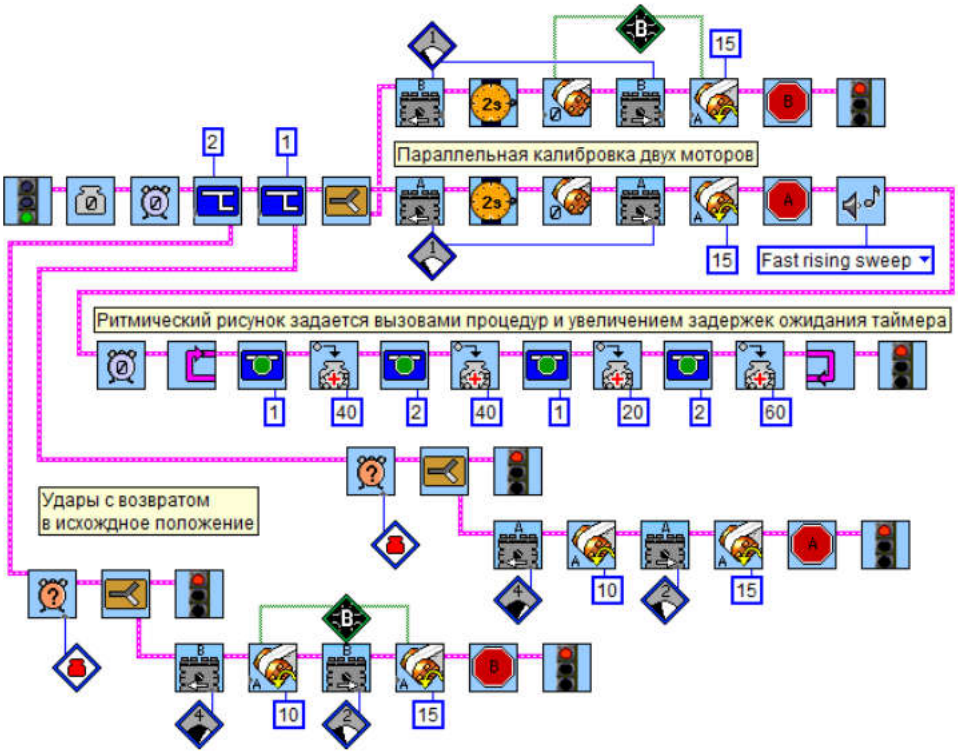


Рис. 8.106. Алгоритм барабанщика с двумя палочками.

Имея массив длительностей, без особого труда можно построить алгоритм его воспроизведения. Это реализуется и в Robolab, только массив будет ограниченной длины — всего допустимо к использованию 48 нумерованных контейнеров (от 0 по 47). А в реальности удастся задействовать чуть больше 20. Это значит, что барабанщик может запомнить ритмический рисунок из 20—25 ударов. Самое интересное заключается в том, чтобы эти удары запоминать и распознавать с помощью датчика звука.

Барабанщик на П-регуляторе

Использование П-регулятора позволит стабилизировать работу барабанщика на очень больших скоростях. Для регулятора будет два ключевых положения уставки, в которых должна располагаться барабанная палочка: 0 и 10 градусов. Оба положения немного сдвинем для компенсации инерции. Построим алгоритм для одного мотора (рис. 8.107).

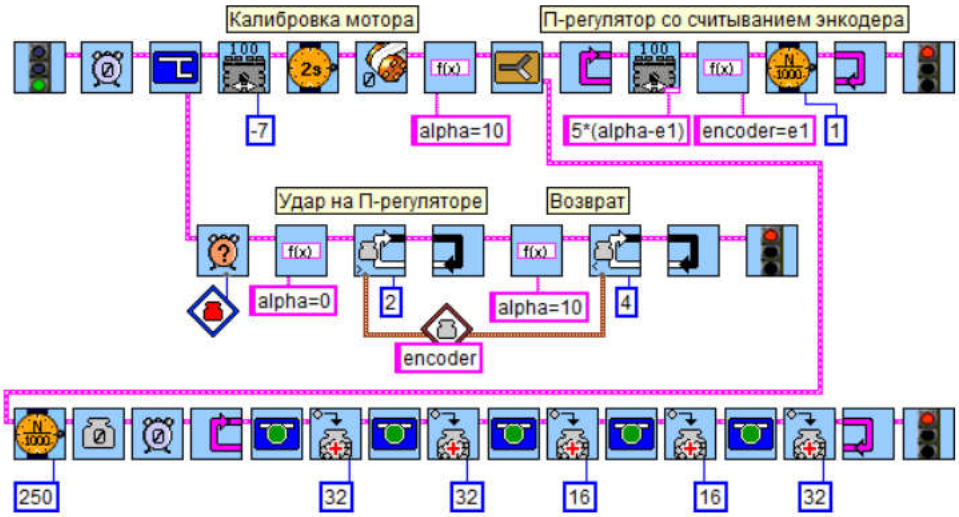


Рис. 8.107. Управление сбалансированной барабанной палочкой с помощью П-регулятора.

Как уже было отмечено ранее, есть такое правило: к одному датчику не обращаться из разных задач одновременно. Поэтому показания энкодера записываются в переменную encoder в цикле П-регулятора. Ее значение используется для задания гарантированного диапазона действия регулятора при ударе с помощью двух пустых циклов с условием.

В чем же будет преимущество такого управления? Представьте себе, что робот не успел поднять палочку до конца, а уже поступил сигнал о начале следующего удара. Естественно, барабанщик не станет дожидаться верхней точки замаха, а сразу приступит к выполению команды. Пусть удар получится немного тише, но он произойдет вовремя. Согласитесь, тихий своевременный удар лучше, чем громкий несвоевременный. Задача робототехника в том, чтобы определить пределы возможностей робота и не допускать запредельных режимов.

Чтобы действия робота приблизить к игре настоящего барабанщика, следует сбалансировать палочку. Тогда скорость ударов значительно повысится. Для этого достаточно прикрепить еще одну балку в обратную сторону (рис. 8.108).

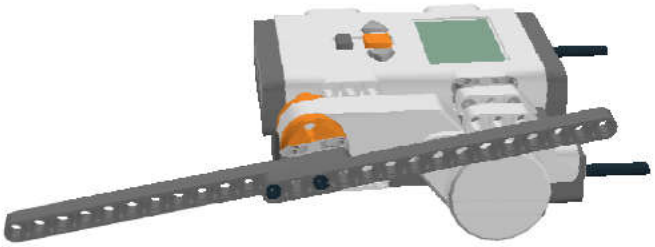


Рис. 8.108. Сбалансировать палочку дополнительной балкой.

Реализовать алгоритм на П-регуляторе для двух палочек читатель может попробовать самостоятельно.

Запоминание ритма

Одной из самых интересных задач для робота-барабанщика является запоминание ритма, предложенного человеком. Используя датчик звука, можно определить момент удара и с помощью таймера запомнить соответствующее время. Доступная память в Robolab ограничена и беспрепятственно можно записать около 32 ударов, сформировав своеобразный массив времени из нумерованныйх контейнеров с номерами от 1 до 31. Контейнер с номером 0, т. е. красный контейнер, будет выступать в качестве индекса массива (рис. 8.109).

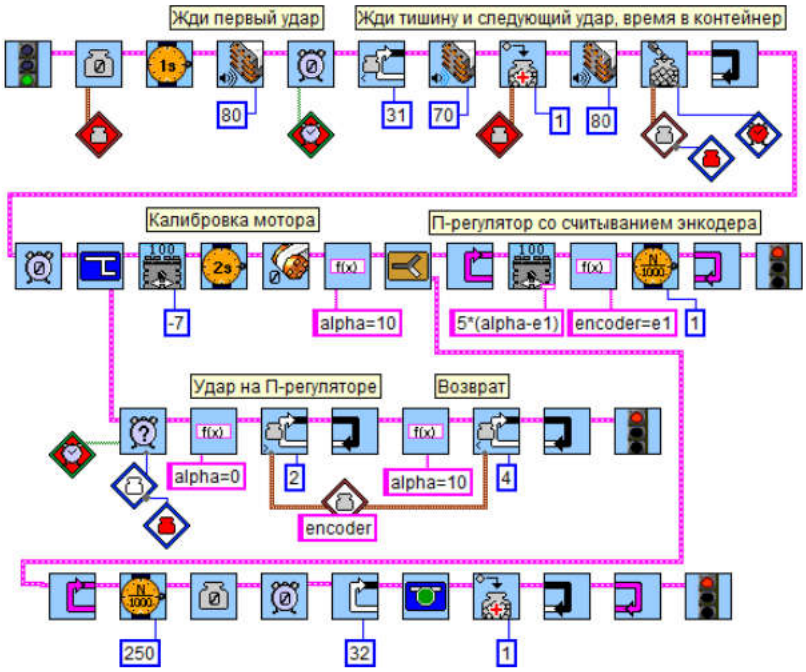


Рис. 8.109. Алгоритм запоминания и воспроизведения ритма человека.

Алгоритм в Robolab запоминает и воспроизводит ровно 32 удара, циклически повторяя рисунок ритма. В среде RobotC снято это ограничение, и по нашей программе робот прекращает запись лишь при паузе длиннее 2 с, а воспроизводит записанное число ударов.

```
int alpha=10, e1=0;  // Угол и значение энкодера
int m[1000];         // Массив для хранения ритма
bool flag=true;      // Условие продолжения
void udar(int count) { // Подпрограмма выполняет 1 удар
  while(Time10[T1]<m[count]) wait1Msec(1); // Время удара
  alpha=0;
  while(e1>2) wait1Msec(1);  // Нижняя точка
  alpha=10;
  while(e1<4) wait1Msec(1); // Контроль подъема палочки
}
task preg() {     // П-регулятор с корректной остановкой
  nNxtExitClicks=2;
  while(flag) {
    e1=nMotorEncoder[motorA];
    motor[motorA]=(alpha-e1)*5;
    wait1Msec(1);
    if(nNxtButtonPressed==0)  // Если нажата кнопка 0
      flag=false;
  }
  StopAllTasks();
}
task main() {
  int count=0, lasttime=0;
  wait1Msec(1000);
  while(SensorValue[S1]<80); // Жди первый удар
  ClearTimer(T1);
  while(Time10[T1]-lasttime<200) { // До паузы 2 с
    lasttime=Time10[T1];
    while(SensorValue[S1]>70);  // Жди тишину
    wait1Msec(10);
    while(SensorValue[S1]<80);  // Жди очередной удар
    m[count]=Time10[T1];
    count++;
  }
  motor[motorA]=-7; // Калибровка
  wait1Msec(2000);
  nMotorEncoder[motorA]=0;
  StartTask(preg);
  while(flag) {    // Повторение ритмического рисунка
    wait1Msec(250);
    ClearTimer(T1);
    for(int i=0; i<count; i++)
      udar(i);
  }
}
```

Лабиринт

Виртуальные исполнители

Перед решением задачи прохождения лабиринта стоит познакомиться со средой «Исполнители»[1], созданной проф. К. Ю. Поляковым из Санкт-Петербургского Государственного морского технического университета. Исполнитель робот, реализованный в ней, очень подходит для начального освоения алгоритмики. Среда полностью русифицирована и содержит Си-ориентированный язык программирования с русской и английской лексикой. Задача поиска выхода из лабиринта решается в этой среде различными способами. Наиболее интересный набор задач[2] для робота разработан учителем информатики Д. М. Ушаковым из физико-математического лицея № 239 Центральнго района Санкт-Петербурга. Среда «Исполнители» особенно актуальна для желающих в будущем освоить язык RobotC.

Также можно порекомендовать систему программирования «КуМир»[3], в которой есть исполнитель робот, способный найти выход из лабиринта. В системе «КуМир» используется школьный алгоритмический язык с русской лексикой, ориентированный на язык Паскаль. Система «КуМир» разработана в Научно-исследовательском институте системных исследований (НИИСИ) РАН по заказу Российской Академии Наук.

Обе среды распространяются бесплатно.

Полигон

На первый взгляд, построение поля лабиринта может вызвать затруднения, однако вложенные усилия вполне окупаются результатом. Поиск выхода из лабиринта является классической задачей, которую решают не только робототехники, но и программисты. Одно из самых ярких соревнований — состязания роботов Micromouse — проводится среди студентов. В нем участвуют разные роботы, совсем не из Lego, оборудованные большим числом датчиков и сложными алгоритмами. И размеры лабиринта составляют 16 × 16 квадратных ячеек.

Наш лабиринт может быть поменьше, например, размером 5 × 5 ячеек, а размер ячейки специально подобран для Lego-роботов: около 30 × 30 см. С учетом толщины стенок, как правило, сторона квадрата ячейки колеблется в диапазоне от 28 до 30 см. Число ячеек и структура

[1] *http://kpolyakov.narod.ru/school/robots/robots.htm*
[2] *http://inform239.narod.ru/robot.html*
[3] *http://www.niisi.ru/kumir/*

лабиринта могут быть любыми. Распространена столешница 150 × 150 см, что и соответствует размеру 5 × 5 ячеек. Обеспечим ее съемными внутренними стенками. Стенки лабиринта желательно сделать повыше, до 16 см, однако подойдут и стандартные, высотой 10 см (рис. 8.110).

Итак, особенность лабиринта в том, что его можно в любой момент изменить, сделав задачу более интересной.

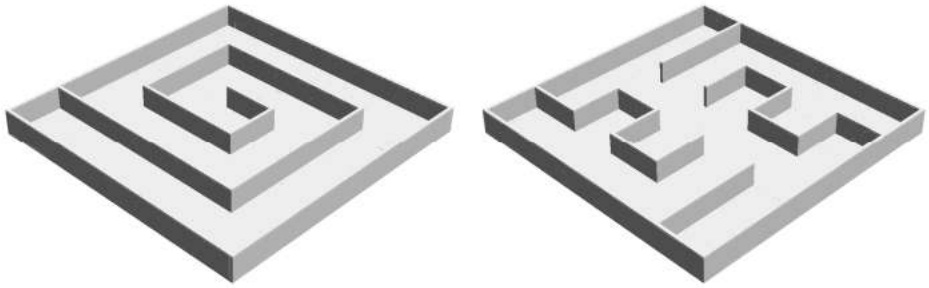


Рис. 8.110. Примеры лабиринтов.

Робот для лабиринта

Каковы характеристики робота? Поскольку приходится иметь дело с замкнутым пространством, робот не должен быть быстрым. В «прямоугольном мире» поворот на 90 градусов должен осуществляться с высокой точностью, как и проезд одной ячейки вперед. Соприкосновение со стеной для робота нежелательно, но не должно вызывать немедленного выхода его из строя. Кроме того, стоит предусмотреть способность двигаться вдоль стены при непрерывном касании. Датчики ультразвука, которые помогут определять расстояние до стен, следует располагать не на самом краю корпуса, чтобы соблюсти минимальное расстояние видимости ультразвукового датчика 5 см.

Исходя из перечисленных условий, оптимальным будет выбор конструкции компактного гусеничного робота. Построить его можно на основе одного конструктора 8547 или конструктора 9797 с добавлением двух гусениц из ресурсного набора 9695. Модель подобного робота предлагается в наборе 8547 в качестве одного из примеров. В нашей версии постараемся сделать робота более компактным за счет извлечения третьего мотора (рис. 8.111—8.117). Самую важную часть — гусеницы — следует закрепить наиболее прочно, по возможности с предельным натяжением.

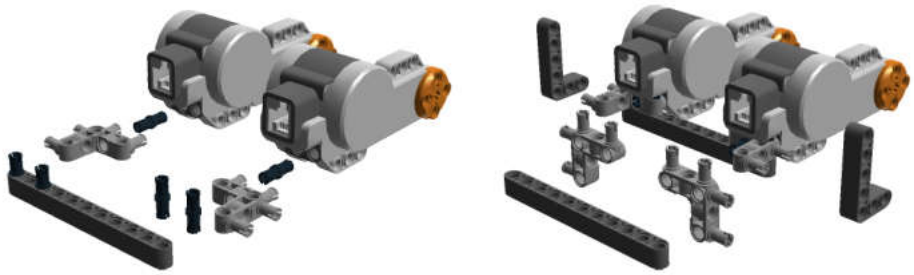


Рис. 8.111. Крепление моторов с помощью 11-модульной балки и угловых соединительных штифтов.

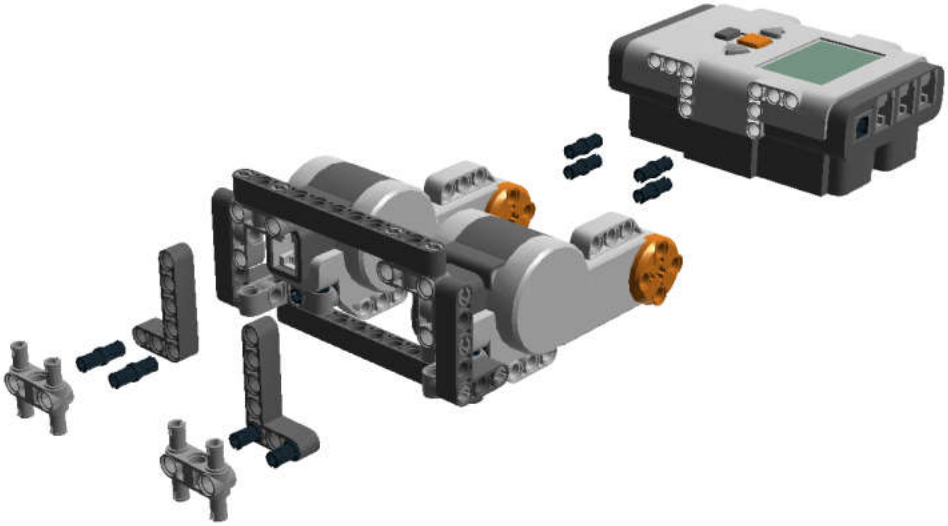


Рис. 8.112. Дополнительные уголки размером 3 × 5 для крепления блока NXT.

Рис. 8.113. Для крепления колесных дисков используются 8-модульные оси в задней части робота.

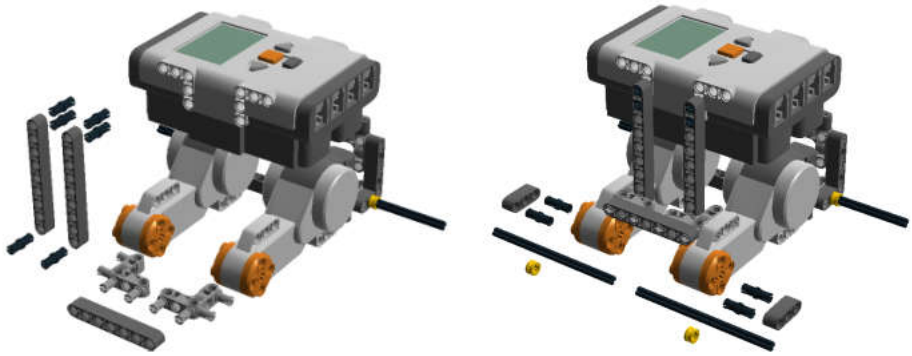


Рис. 8.114. Вертикальные балки для крепления NXT спереди могут быть любой длины. В оранжевые диски моторов вставляются 10-модульные оси.

Рис. 8.115. Колесные диски прижимаются к желтым втулкам.

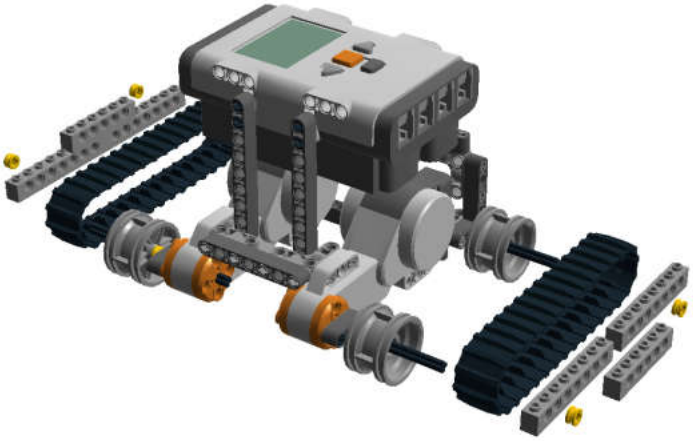


Рис. 8.116. Гусеницы можно закрепить 16-модульными балками с выступами.

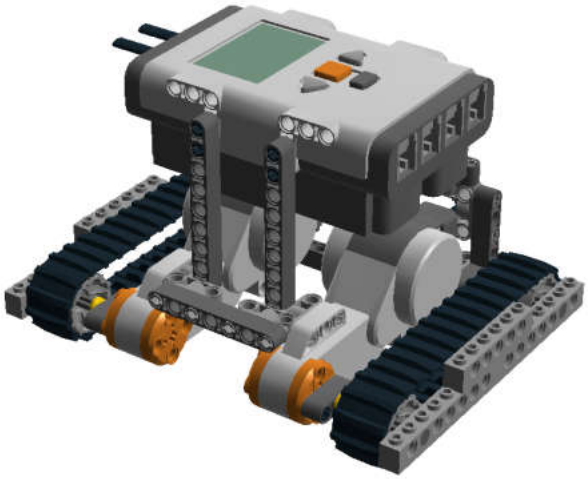


Рис. 8.117. Первая версия тележки готова.

Направление движения робота — вперед оранжевыми дисками моторов, подключенный на порты B (левый) и C (правый). Протестируйте работоспособность модели на примере программ из NXT Program.

Известный лабиринт

Первое, что научится выполнять наш робот, — точные перемещения. Для начала их будет всего три вида: проезд одной клетки вперед, поворот направо на 90 градусов, поворот налево на 90 градусов. Выделим их в три отдельных набора команд (рис. 8.118). Строго говоря, команды такого типа являются высокоуровневыми, т. е. содержат внутри себя какие-то сложные действия, исполняемые операционной системой, которые программисту не видны. В системах с виртуальным исполнителем «Робот» так и происходит. Для реализации этого в физическом мире необходимо научиться управлять роботом на «низком уровне», т. е. все команды управления роботом указывать достаточно подробно.

```
Моторы B, C вперед
Жди 1000 на энкодере B
Моторы стоп
```

Рис. 8.118. Реализация трех базовых команд для лабиринта на языке Robolab.

Конструкция гусеничного робота такова, что проезд ячейки длиной 30 см требует поворота моторов примерно на 1000 градусов. А поворот на 90 градусов на месте выполняется при повороте одного мотора на 500 градусов вперед, а другого на –500 градусов назад.

Пример на языке Robolab показывает прохождение первых трех ячеек лабиринта (рис. 8.119). Программа составлена из трех блоков команд, повторяющихся в разном порядке.

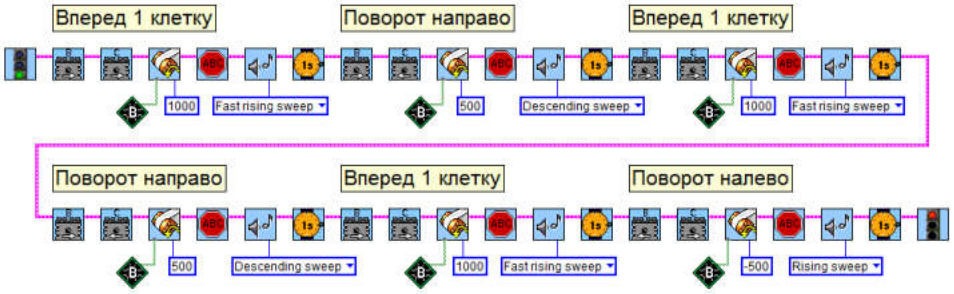


Рис. 8.119. Пример программы прохождения трех ячеек лабиринта.

Первые запуски, скорее всего, приведут к тому, что робот начнет зацепляться за неровности в стенах лабиринта своими бортами. Для защиты от зацепов существует элементарное решение, уже использованное в шагающем роботе для NXT 2.0. Это горизонтальные свободно вращающиеся колесики (рис. 8.120).

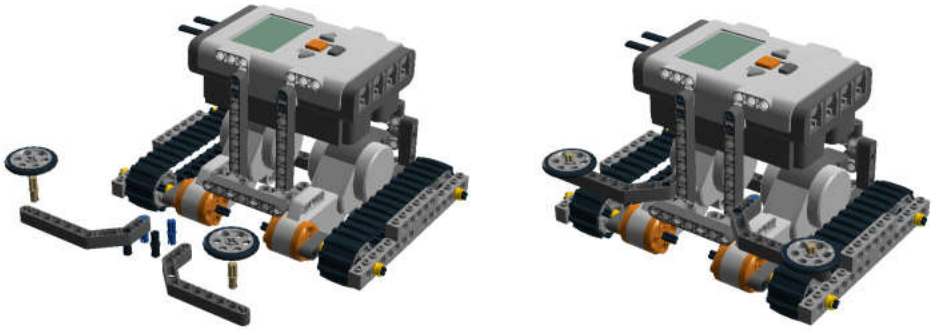


Рис. 8.120. Горизонтальные колесики устанавливаются на гладкие штифты или оси в круглые отверстия на изогнутых балках.

После установки колесиков в передней части робота можно добиться выравнивания при движении вдоль стен лабиринта даже при недостаточно точных поворотах.

Возвращаясь к программированию, придется признать, что действуя по предложенному выше алгоритму, можно составить программу прохождения лабиринта размером 5×5, которая уже не уместится в один экран Robolab. Копирование блоков сильно удлиняет программу,

что увеличивает вероятность возникновения ошибок, а при смене структуры лабиринта делает редактирование затруднительным. Опытный программист сразу догадается, что можно использовать подпрограммы!

Для ясности управления следует представить робота как классического исполнителя с тремя базовыми командами: «Вперед», «Налево», «Направо». Каждая из них является высокоуровневой и с точки зрения реального исполнителя — довольно сложной. Команда «Вперед» — это проезд одной клетки лабиринта с последующей остановкой. Команды «Налево» и «Направо» — это повороты на 90 градусов с максимально возможной точностью. Недостатки исполнения команд, связанные с люфтом и трением, придется компенсировать конструктивно.

По принципу нумерования верхних кнопок контроллера NXT три процедуры получат соответствующие номера: 3 — «Вперед», 2 — «Налево», 1 — «Направо». В процедуры стоит включить необязательные, но полезные действия, которые позволят лучше контролировать выполнение отдельных команд: остановки, задержки и звуковые сигналы.

Используя отдельные вызовы процедур, составьте программу прохождения некоторой части лабиринта (рис. 8.121).

Обратите внимание, что для каждой конструкции робота значения градусов энкодера будут свои, даже при внешне одинаковых моделях они будут зависеть от множества незаметных факторов.

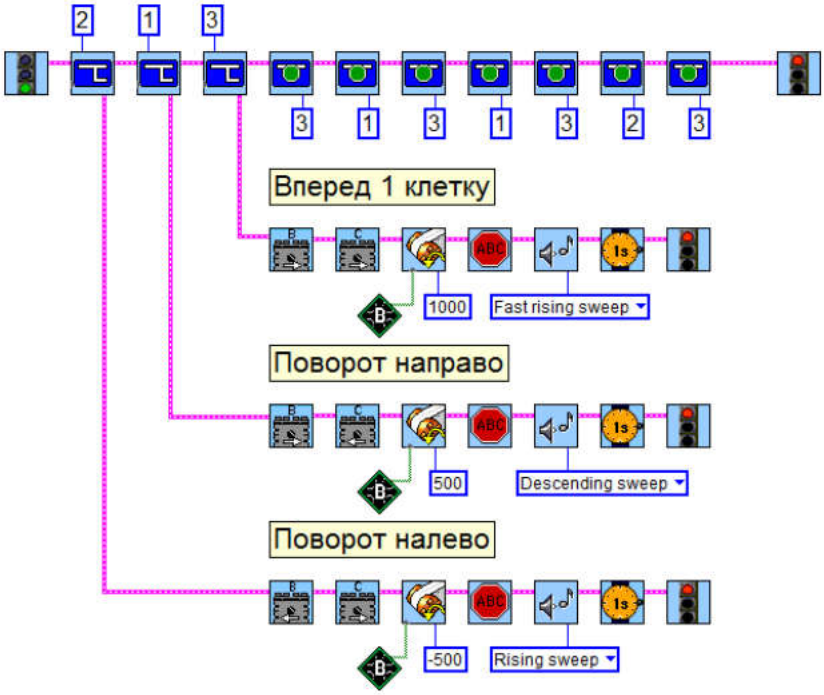


Рис. 8.121. Три базовых действия в процедурах и начало программы прохождения лабиринта.

Используя указанные процедуры, можно построить программу прохождения лабиринта достаточно быстро. Однако при изменении его структуры придется менять и алгоритм. Существует ли универсальный алгоритм прохождения лабиринта?

Правило правой руки

Классическая задача обхода неизвестного лабиринта решается по правилу правой (левой) руки. Его можно сформулировать следующим образом: робот движется, постоянно контролируя стену справа. То есть, если появится проем справа, следует заехать в него. Если же уперлись в стену, следует скомандовать роботу повернуть налево.

Можно представить себе странника, блуждающего по темному лабиринту. Если выход существует на границе лабиринта, то странник найдет его, постоянно придерживаясь стены правой рукой.

Для робота лабиринт тоже в некотором смысле темный, пока в работу не включены датчики. Оптимальное решение можно получить с помощью двух датчиков расстояния, один из которых направлен направо, а другой — вперед. Диапазон видимости ультразвуковых датчиков ограничен снизу расстоянием 5 см, поэтому в конструкции робота их следует располагать не на самом краю корпуса (рис. 8.122-8.123).

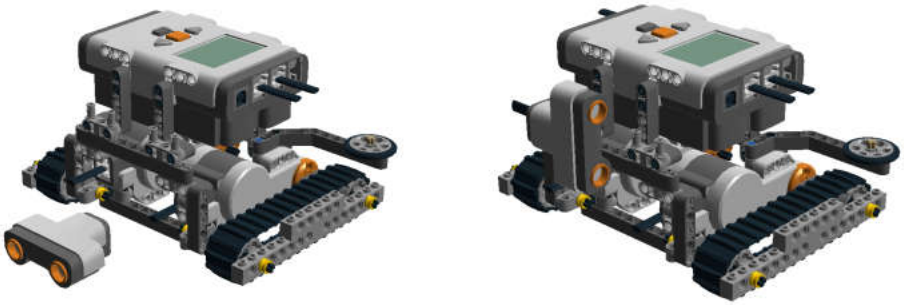


Рис. 8.122. Установка первого датчика расстояния сзади, направление вправо.

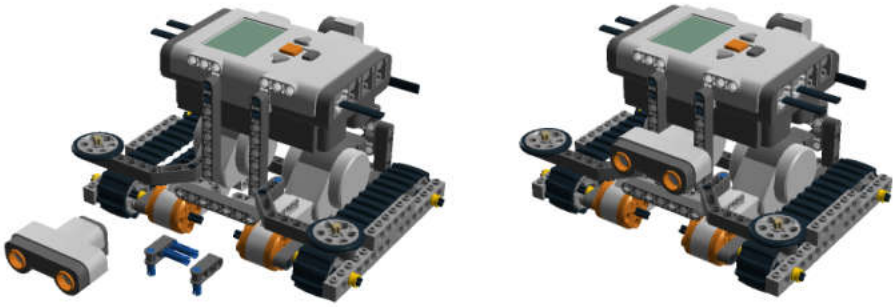


Рис. 8.123. Установка второго датчика расстояния спереди.

Установив датчики соответственно на порты 1 и 2, следует видоизменить программу, оставив только процедуры. Основной алгоритм заменяется на бесконечный цикл с двойной проверкой условия. Поскольку длина ячейки лабиринта составляет 30 см, то разумным будет проверять аналогичное расстояние при поиске стены (рис. 8.124):

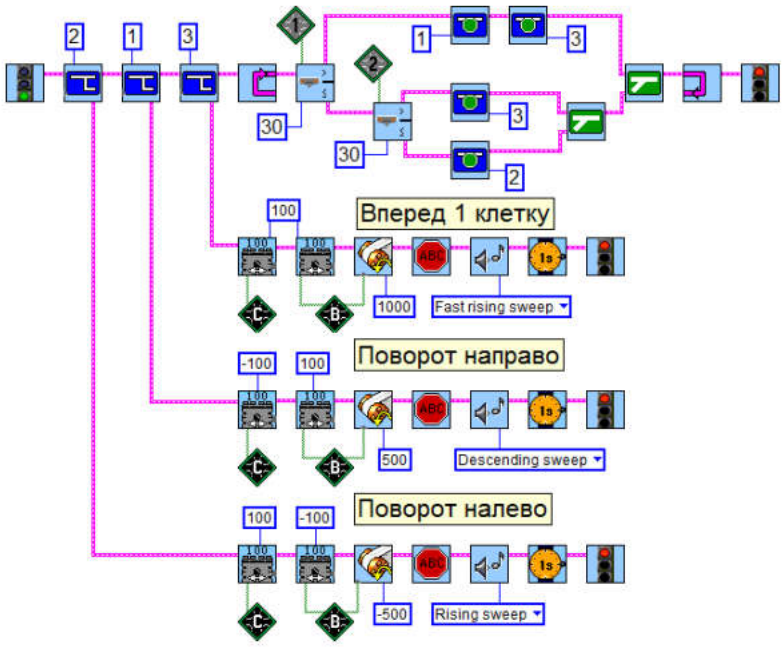


Рис. 8.124. Алгоритм прохождения лабиринта по правилу правой руки.

```
void forward()  // Проезд вперед одну ячейку
{
  motor[motorB]=motor[motorC]=100;
  nMotorEncoder[motorB]=0;
  while(nMotorEncoder[motorB]<1000);
  motor[motorB]=motor[motorC]=0;
  PlaySound(soundFastUpwardTones);
  wait1Msec(1000);
}
void right()  // Поворот направо
{
  motor[motorB]=100;
  motor[motorC]=-100;
  nMotorEncoder[motorB]=0;
  while(nMotorEncoder[motorB]<500);
  motor[motorB]=motor[motorC]=0;
  PlaySound(soundDownwardTones);
  wait1Msec(1000);
}
```

```
void left()  // Поворот налево
{
  motor[motorB]=-100;
  motor[motorC]=100;
  nMotorEncoder[motorC]=0;
  while(nMotorEncoder[motorC]<500);
  motor[motorB]=motor[motorC]=0;
  PlaySound(soundUpwardTones);
  wait1Msec(1000);
}
task main() // Основной алгоритм
{
  while(true) {
    if(SensorValue[S1]>30) {  // Если справа проем
      right();
      forward();
    }
    else
      if(SensorValue[S2]>30) // Если спереди свободно
        forward();
      else
        left();
  }
}
```

Следующим этапом может стать защита от застреваний. Поскольку повороты физической модели неидеальны, на любом из них робот может зацепиться за угол стены. Пусть читатель самостоятельно построит защиту от застреваний, аналогичную предложенной для робота, путешествующего по комнате.

Удаленное управление

Передача данных

Контроллер NXT оснащен устройством беспроводной передачи данных Bluetooth второго класса. Это значит, что бесперебойная связь гарантирована на расстоянии до 10 м. Но качество и скорость обмена данными во многом будут зависеть от команд и алгоритмов, которые используются при программировании. Соединить можно как два контроллера между собой, так и контролер с компьютером или мобильным телефоном, оснащенным Bluetooth. Существует специальное программное обеспечение, которое с компьютера или мобильного телефона позволяет передавать на NXT команды управления подключенными к не-

му устройствами: моторами, датчиками и пр. В этом случае нет необходимости запускать какую-либо программу на NXT-приемнике, достаточно установить соединение. Если же необходимо передавать данные, то на приемнике и получателе запускаются разные программы, которые отправляют и обрабатывают полученные данные.

Обмен информацией следует разделить на четыре составляющих:

1) установка соединения,
2) передача данных или управляющих команд,
3) прием данных или выполнение управляющих команд,
4) завершение соединения.

Bluetooth-устройство, которое инициирует подключение, называется ведущим (master). Устройство, которое принимает подключение, называется ведомым (slave). К одному «мастеру» может быть подключены до трех ведомых NXT, по одному на каждый виртуальный порт (соответственно на 1-й, 2-й и 3-й).

Соединение можно устанавливать вручную, а можно и программно, эта функция реализована в RobotC.

При первом соединении двух устройств запрашивается числовой пароль, который нужно ввести на каждом из них. Впоследствии он не требуется. Подключаемые устройства узнают друг друга по имени (поэтому все имена должны быть уникальными) и способны обмениваться данными. Рассмотрим порядок действий при установке соединения вручную.

1. Включить два NXT и убедиться, что у них уникальные имена.
2. На обоих контроллерах включить Bluetooth, вследствие чего в левом верхнем углу экрана должен появиться фирменный значок.
3. На ведомом контроллере включить опцию «Виден всем» (Visibility → Visible).
4. На ведущем контроллере включить режим поиска соседних устройств (Search).
5. Из списка найденных устройств выбрать ведомый и подключиться к нему.
6. При первом подключении потребуется ввести числовой код (по умолчанию «1234») и нажать «галочку». При последующих подключениях нужный контроллер можно будет найти по имени в разделе My Contacts.
7. Выбрать любой порт: 1, 2 или 3.
8. При успешном соединении ведомый издаст звуковой сигнал (если включен звук) и на экране каждого из контроллеров рядом со значком Bluetooth появится ромбик «<>». Если подключения нет, то останется значок «<», т. е. левая половинка ромбика.

После того, как подключение состоялось, можно передавать файлы или запускать программы, поддерживающие обмен данными. При этом оба контроллера, и master, и slave на равных правах могут посылать и получать информацию.

Со времен инфракрасной связи в RCX по «почте» передавался 1 байт, а в NXT допускаются большие числа: от -2^{15} до $2^{15}-1$, т.е. в диапазоне –32768...32767. Ноль в «почтовом ящике» означает, что ничего не было принято, поэтому отправлять его не имеет смысла.

На NXT-приемник и NXT-передатчик загружаются разные программы (рис. 8.125).

Рис. 8.125. Передача числа 5 по нажатию датчика (слева) и ответный звуковой сигнал (справа).

В общем случае перед приемом данных необходимо изначально инициализировать (т. е. обнулить) почтовый ящик, чтобы очистить его от писем, которые могли остаться от предыдущих сеансов. Команда «Жди письма» (Wait for mail) сама обнуляет его и ждет очередного письма с заданным значением. Если конкретное число не задано, ожидается письмо с ненулевым значением.

Теперь рассмотрим пример с передачей двух различных чисел (рис. 8.126).

Рис. 8.126. Передача чисел 4 или 5 по нажатию соответствующего датчика и ответные звуковые сигналы.

На передатчике две параллельные задачи осуществляют отправку значений 4 или 5 в зависимости от того, какой из двух датчиков касания был нажат. Для повторного срабатывания после нажатия требуется отпустить датчик. На приемнике также две параллельные задачи «заглядывают» в почтовый ящик и вызывают соответствующий звуковой сигнал. При этом вызовы сигналов могут «наслаиваться» друг на друга,

поскольку письма с разными значениями могут приходить довольно часто.

Передача данных является довольно медленной операцией, на гарантированную доставку одного сообщения тратится 0.03 — 0.05 с. Можно, конечно, отправлять сообщения слишком часто, не заботясь о том, будут ли они приняты. В некоторых случаях работает и такой подход. Мы будем стремиться к устранению потерь пакетов при передаче. В двух приведенных выше примерах естественную задержку перед отправкой письма создает сам человек, нажимая на датчик с некоторыми перерывами.

Теперь рассмотрим пример удаленного управления мотором, в котором задержки задаются программно (рис. 8.127). Чтобы увидеть его в действии, необходимо на приемник и передатчик прикрепить по мотору, подключив каждый к порту A (8.128).

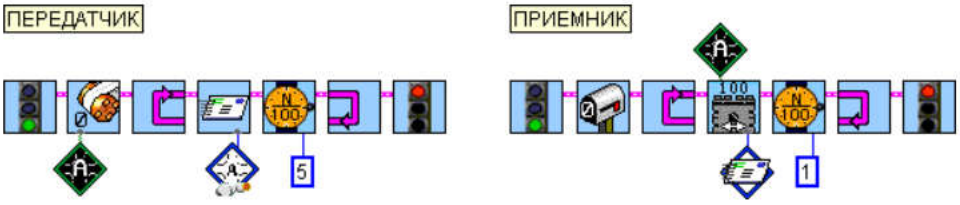


Рис. 8.127. Передача значения энкодера и управление скоростью удаленного мотора.

Для передачи значения энкодера мотора A используется соответствующий модификатор (Value of Encoder A из палитры NXT Commands), значение которого отправляется в виде письма с задержкой 0.05 с на доставку пакета. На приемнике после обнуления почты используется модификатор (Value of Mail из палитры Modifiers), который несет в себе значение полученного письма (т. е. содержимое почтового ящика). Это значение подается в качестве скорости управляемого мотора A.

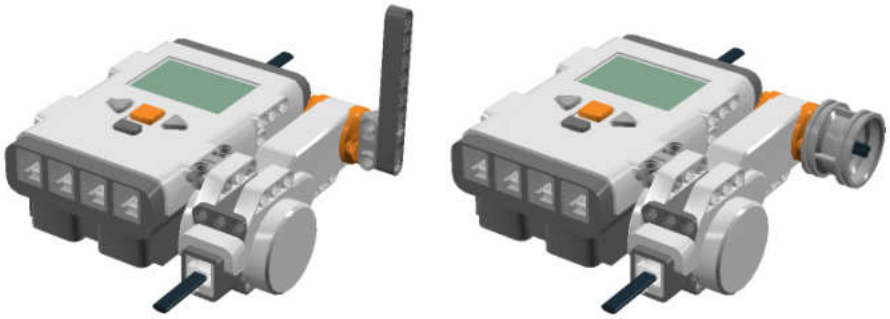


Рис. 8.128. Приемник вращает колесо со скоростью, заданной на передатчике поворотом балки.

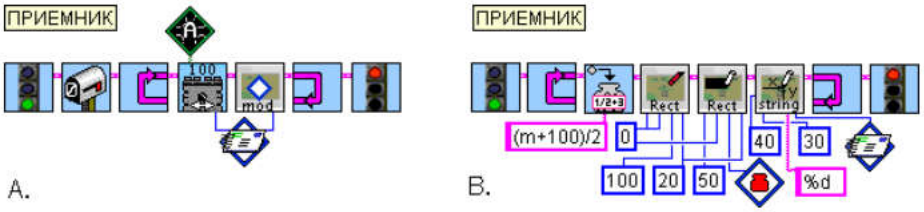


Рис. 8.129. Отображение принимаемого значения: слева — в виде числа, справа — в виде диаграммы.

Усовершенствуем приемник, добавив отображение принимаемого значения на экране в виде числа или даже диаграммы (рис. 8.130).

На рис. 8.129 слева показано, что одновременно с управлением скоростью мотора на экране приемника отображается принимаемое значение. Задержка отсутствует, поскольку вывод на экран сам по себе является довольно существенной задержкой.

На рис. 8.129 справа принимаемое значение в нижней части экрана отображается в виде изменяющегося прямоугольника, а в центральной — в виде числа. Поскольку максимальная ширина прямоугольника на экране 100 точек, что в 2 раза меньше диапазона скоростей мотора –100...100, то полученное по почте значение *m* смещается в неотрицательный диапазон и сокращается вдвое. Координата X одной пары вершин отображаемого прямоугольника находится в позиции 50 (визуальная точка отсчета), а координата противоположной пары (значение красного контейнера) колеблется в диапазоне 0...100.

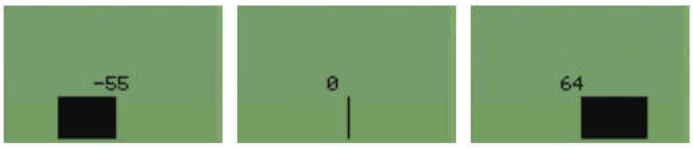


Рис. 8.130. Диаграмма на экране NXT.

Читатель может добавить в программу управление скоростью мотора, показанное на рис. 8.129, самостоятельно.

Теперь рассмотрим удаленное управление положением мотора (рис. 8.131). Такая возможность полезна при взаимодействии с роботом-манипулятором, который находится на удалении от оператора.

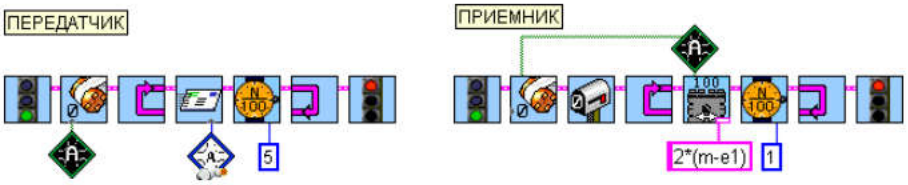


Рис. 8.131. Управление положением удаленного мотора.

Изменяемые вручную показания энкодера передатчика с достаточно высокой частотой меняют содержимое почтового ящика приемника, благодаря чему П-регулятор выводит мотор приемника в соответствующее положение (рис. 8.132). Задержка в цикле приемника (0.01 с) меньше, чем в цикле передатчика (0.05 с), поскольку П-регулятор может работать и со «старым» значением почтового ящика, а его изменение должно быть зафиксировано как можно быстрее.

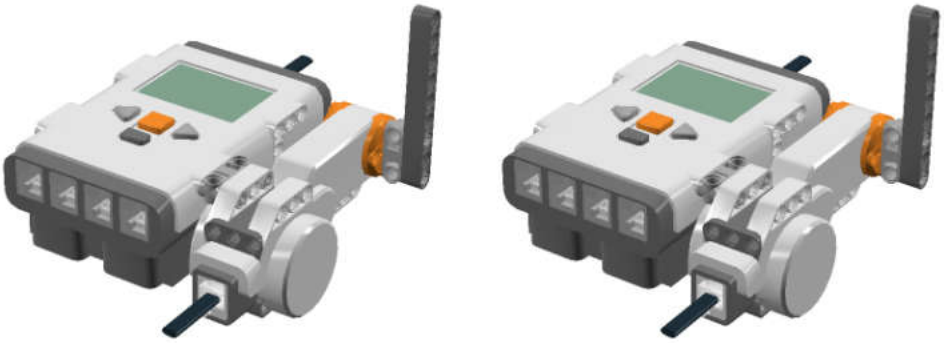


Рис. 8.132. Мотор приемника повторяет движения мотора передатчика.

Кодирование при передаче

Управление двумя моторами — это более сложная задача, поскольку в ней требуется объяснить приемнику, к какому из моторов относится каждое отправляемое значение. Прежде чем браться за полное решение, рассмотрим простейший случай с использованием двух датчиков касания, каждый из которых отвечает за свой удаленный мотор (рис. 8.133).

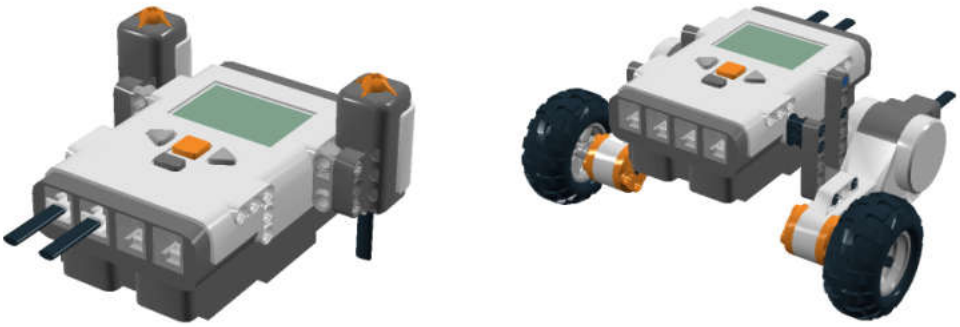


Рис. 8.133. Управление движением тележки с помощью датчиков касания.

Чтобы не увеличивать вероятность ошибок, загромождая эфир вспомогательными сообщениями, закодируем всю информацию в одном. Датчик касания передает контроллеру всего два значения: 0 (отпущен) или 1 (нажат), — т. е. 1 бит информации. Сформируем двоичный сигнал из двух битов, в котором поместим значения датчиков *S*1 и *S*2 на соответствующих портах (рис. 8.134).

Отправленный сигнал будет расшифрован без особых усилий с помощью операций целочисленного деления. Полученные биты необходимо домножить на 100, чтобы получить полную мощность двигателя (рис. 8.135).

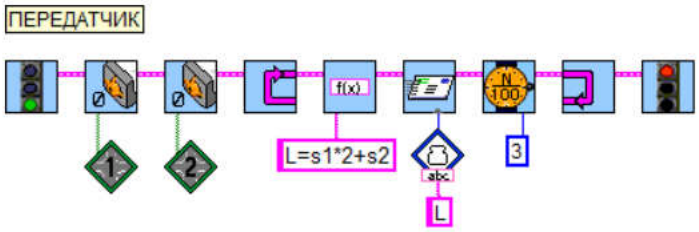


Рис. 8.134. Передача закодированного сигнала с двух датчиков касания.

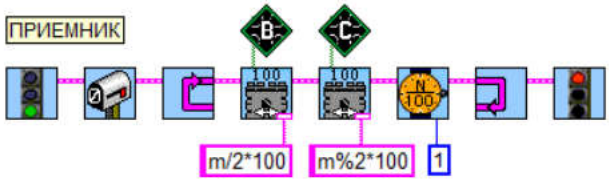


Рис. 8.135. Расшифровка закодированного сигнала и подача его на двигатели.

Надо обратить внимание, что тележка будет ехать только вперед или поворачивать. Движение назад не предусмотрено. Можно было бы добавить третий датчик, но гораздо интереснее научиться передавать полноценный управляющий сигнал сразу на два мотора. Рассмотрим конструкцию самодельного двуручного джойстика, которым будем осуществлять управление (рис. 136, слева). Обратите внимание, что на джойстиках моторы подключены на порты A и B, в то время как на тележке на B и C.

Как изменить алгоритм, изображенный на рис. 8.137, чтобы он работал для двух двигателей на каждом контроллере? Самый простой и неэффективный способ — отправлять четыре сообщения, по два на каждый мотор: сперва служебный сигнал (префикс) — номер мотора, затем значение скорости, которое на него нужно подать (рис. 8.137). При этом следует ставить защиту от потерь сообщений, т. е. заменять скорость моторов, равную префиксам 1 или 2, на значение 3.

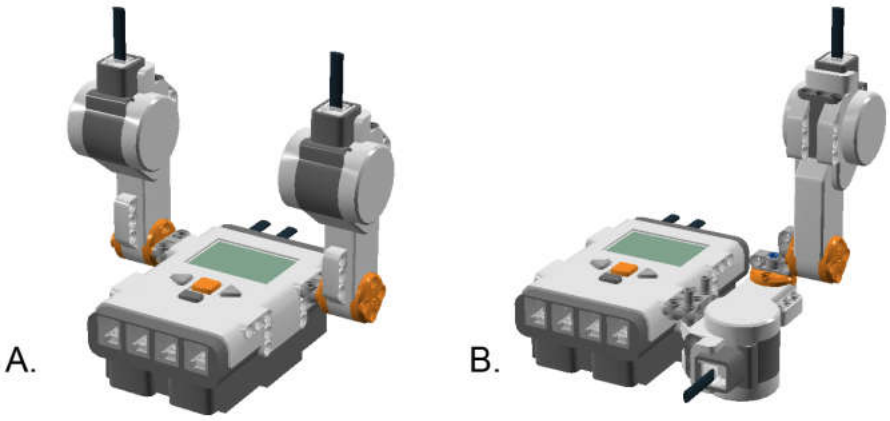


Рис. 8.136. Джойстики: слева — для двух рук, справа — для одной руки.

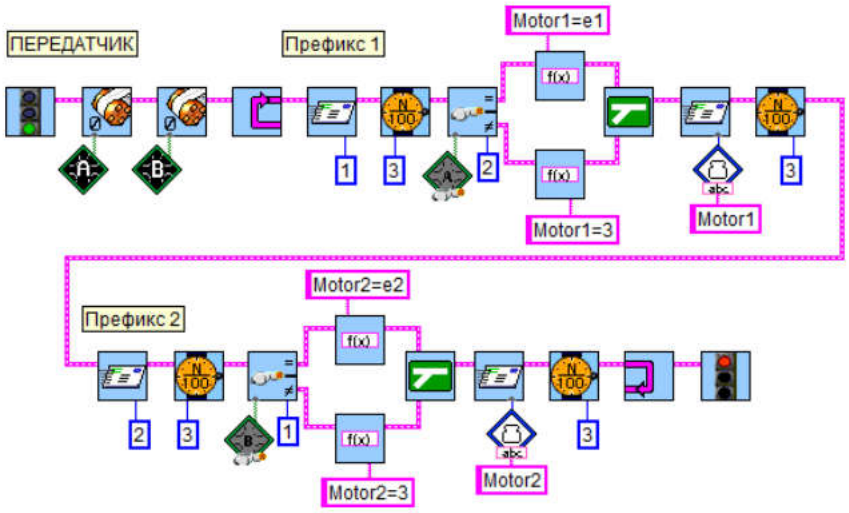


Рис. 8.137. Последовательная передача значений скорости на 2 мотора.

Алгоритм приема сообщений выглядит проще: приходит префикс, обнуляется почтовый ящик и следующее сообщение воспринимается как мощность соответствующего мотора (рис. 8.138). Если по какой-то причине приемник не получил вовремя нужный префикс, сообщение в почтовом ящике быстро обновляется.

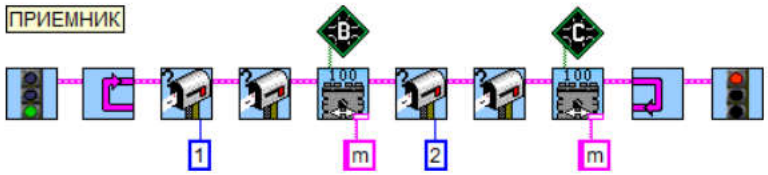


Рис. 8.138. Последовательный прием скоростей моторов.

Решение получилось громоздкое, а скорость передачи оказалась минимум в 4 раза ниже возможной. Гораздо эффективнее было бы передавать всю информацию в одном сообщении, как в предыдущем примере. Для этого придется задействовать весь доступный диапазон значений, передаваемых по Bluetooth, т. е. от –32 768 до 32 767.

В первую очередь придется ограничить диапазон передаваемых значений. Обычно джойстик позволяет повернуть мотор не более чем на 90 градусов вперед или назад. Однако для страховки следовало бы взять небольшой запас. Тем более что ближайший кодируемый диапазон –128...127, т. е. 1 байт или 256 возможных значений. А мощность, подаваемая на моторы приемника, находится в диапазоне –100...100. Таким образом, во всех отношениях диапазон выбран удачно. Поскольку для передачи данных имеется 2 байта, мы имеем возможность передать мощность обоих моторов одним письмом, воспользовавшись кодированием. Осталось только разобраться с отрицательными значениями. Как выясняется, кодировать удобнее беззнаковые числа, а передавать придется знаковые. Вопрос решается несколькими операциями сложения и вычитания.

Итак, $e1$ и $e2$ — значения энкодеров передатчика в диапазоне –128...127. Дополним их до неотрицательного диапазона 0...255:

$$e1new = e1 + 128,$$
$$e2new = e2 + 128.$$

Закодируем два числа в одном со сдвигом первого на 8 битов влево. Можно сказать, воспользуемся 256-ричной системой счисления:

$$Lnew = e1new \cdot 256 + e2new.$$

Получили значение $Lnew$ в диапазоне 0...65 535 (рис. 8.139). Последовательность нулей и единиц в нем не имеет решающего значения, это некоторые показания энкодеров, представленные в двоичной форме по 8 бит на каждый.

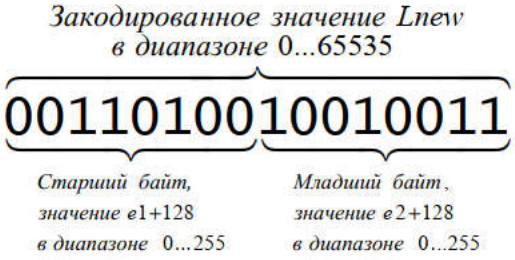


Рис. 8.139. Битовая структура закодированного сообщения.

Теперь переводим его в знаковый диапазон, поддерживаемый Bluetooth:

$$L = Lnew - 32\,768.$$

Получили готовое к пересылке значение, которое полностью можно записать следующей формулой:

$$L = (e1 + 128) \cdot 256 + (e2 + 128) - 32\,768.$$

Теперь рассмотрим приемник. Полученное сообщение m (так названо значение почтового ящика в Robolab) необходимо дополнить до неотрицательного диапазона, после чего раскодировать с помощью операций целочисленного деления и подать соответствующую скорость на моторы:

$mnew = m + 32\,768$, получили диапазон –32 768...32 767;

$e1new = mnew/256$, получили старший байт в диапазоне 0...255;

$mB = e1new - 128$, получили значение $e1$ первого энкодера передатчика, которое в чистом виде подаем на мотор B;

$e2new = mnew \,\%\, 256$, получили младший байт в диапазоне 0...255;

$mC = e2new - 128$, получили значение $e2$ второго энкодера передатчика, которое в чистом виде подаем на мотор C.

Алгоритм, который позволит осуществить полноценное управление тележкой с двумя моторами дан на рис. 8.139.

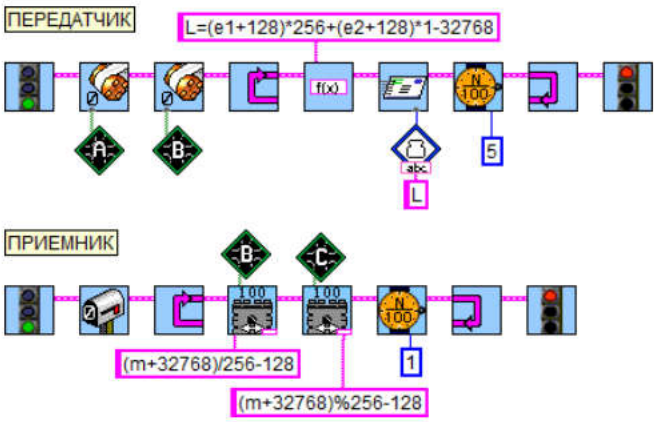


Рис. 8.140. Управление моторами через Bluetooth с помощью самодельного джойстика (конструкция дана на рис. 8.136 слева).

Двуручный джойстик в нашем исполнении будет удобен не каждому. Да и контроллер придется на чем-то устанавливать. Объединив моторы в одноручный джойстик, получим мобильную систему управления

с интуитивно понятным интерфейсом (рис. 8.136, справа). Верхний мотор, подключенный на порт A, будет управлять скоростью робота, а нижний, подключенный порт B, — направлением движения. Для работы с таким джойстиком придется изменить только алгоритм приемника. Сигнал, подаваемый на моторы, пропорционален скорости робота. Обозначим его через v, а корректировку направления движения, т. е. управляющее воздействие, — через u (рис. 8.141):

$$v = (m + 32\,768) / 256 - 128,$$
$$u = (m + 32\,768) \,\%\, 256 - 128.$$

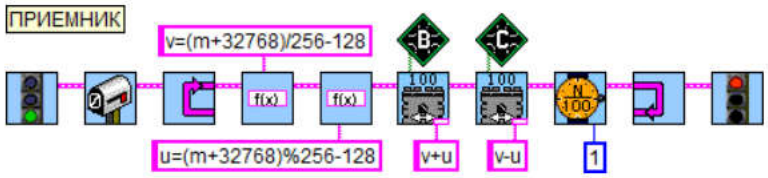


Рис. 8.141. Прием управляющих сигналов скорости и поворота с одноручного джойстика (конструкция дана на рис. 8.136 справа).

Тогда моторы будут управляться следующим образом:

$$mB = v + u,$$
$$mC = v - u.$$

Для повышения или понижения чувствительности джойстика можно ввести усиливающие коэффициенты (рис. 8.142):

$$v = k1 \cdot v,$$
$$u = k2 \cdot u.$$

Это особенно актуально для управляющего воздействия u, поскольку при повороте на месте на максимальной скорости потребуется получить число около 200 или –200 при том, что передаваемые значения ограничены диапазоном –128...127.

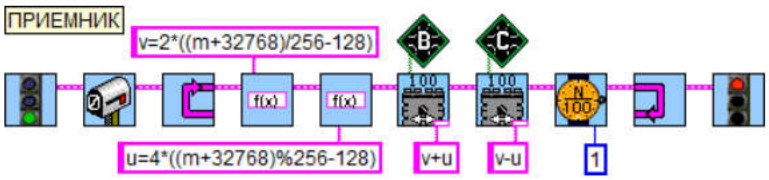


Рис. 8.142. Управляющие сигналы с усиливающими коэффициентами 2 и 4.

Дополнительный режим джойстика

Пальцев на руке пять, а для удержания джойстика требуется два или три. Таким образом, для повышения эффективности управления мы можем ввести дополнительные датчики, которые будут отвечать, например, за изменение скорости («турбо-режим») или за удар по мячу с помощью третьего мотора. Рассмотрим вариант с датчиком касания, закрепленным на верхнем моторе (рис. 8.143). На кнопку удобно нажимать большим пальцем.

Рис. 8.143. Датчик касания на джойстике.

Алгоритм управления следует несколько изменить. До сих пор мы тратили на каждый двигатель по 8 бит информации, что при имеющейся точности движений является избыточным. Займем 1 бит для передачи показания датчика касания, у которого вариантов всего два: 0 (отпущен) и 1 (нажат). Заменив младший бит четности, мы потеряем в точности управления в 2 раза, что будет практически незаметно (рис. 8.144).

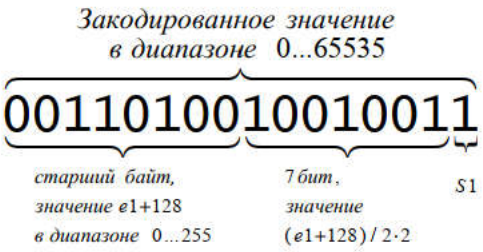


Рис. 8.144. Замена бита четности на показания датчика касания *S*1.

Чтобы избавиться от младшего бита, достаточно выполнить незатейливую процедуру: увеличенные показания энкодера разделить на 2 и умножить на 2. Поскольку деление происходит в целых числах, младший бит неизбежно будет обнулен, а его место займет значение *S*1:

$$L = (e1 + 128) \cdot 256 + (e2 + 128) / 2 \cdot 2 + S1 - 32\,768.$$

На принимающей стороне несколько изменится анализ младшего байта, а старшего останется прежним:

$v = (m + 32\,768) / 256 - 128,$
$u = (m + 32\,768) \% 256 / 2 \cdot 2 - 128,$
$k = (m + 32\,768) \% 2,$
$v = \mathrm{v} \cdot (k + 1),$
$u = \mathrm{u} \cdot (2 - k) / 2.$

В переменную k будет записано переданное значение $S1$. Используя ее, смоделируем режим «турбо», т. е. увеличение скорости v и понижение чувствительности к управлению u (рис. 8.145). Выбрав курс робота в обычном режиме, можно включить «турбо» нажатием кнопки, и робот выполнит рывок в нужном направлении.

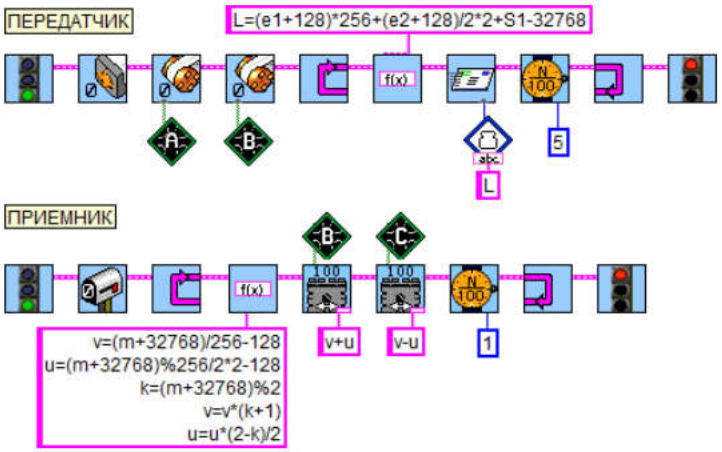


Рис. 8.145. Режим «турбо» включается датчиком касания на передатчике.

Другой пример использования дополнительного бита — это нанесение удара по мячу с помощью третьего мотора. По правилам игры в управляемый футбол, которая с 2011 г. проводится на Математико-механическом факультете Санкт-Петербургского государственного университета (СПбГУ), размер робота в момент удара по мячу не должен превышать цилиндра диаметром 22 см и высотой 22 см.

Для того, чтобы запрограммировать удар, необходимо логически отделить первый полученный сигнал к удару от всех последующих до тех пор, пока «клюшка» не будет приведена в исходное положение. В пылу игры человек может несколько раз нажать на кнопку и продержать ее в нажатом состоянии существенно дольше, чем это необходимо. Задача робота по требованию спокойно от начала до конца выполнить очередной удар и приступить к следующему только в том случае, если после завершения продолжает поступать соответствующий сигнал. Алгоритм приемника приведен на рис. 8.146. Для передатчика подойдет программа из предыдущего примера (рис. 8.145).

Рис. 8.146. Удар по мячу роботом с третьим мотором.

Для управления третьим мотором используется параллельная задача, вход в которую контролируется контейнером-семафором *udar*. На время выполнения удара семафор «запрещает» повторный вызов задачи и обеспечивает корректную работу программы.

Пример робота с ударным механизмом изображен на рис. 8.147. Его разработал ученик физико-математического лицея № 239 Илья Балташов. Игра в футбол с такими роботами происходит в формате 3 × 3 или 5 × 5 на поле размером 4 × 6 м. В качестве покрытия используется ковролин. Мяч для гольфа оказался наиболее подходящим для такого размера роботов, хотя можно использовать и мячик из набора 9797.

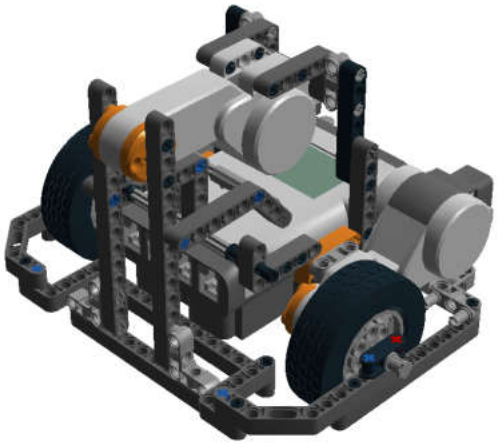


Рис. 8.147. Робот с ударным механизмом для игры в футбол.

Существует множество приложений для ноутбуков и мобильных телефонов, с помощью которых можно управлять таким роботом (например, nxtremote). Их преимущество — высокое быстродействие. Их недостаток — отсутствие возможности развития алгоритма. Если же составлять программу самостоятельно, то, используя методы кодирования, можно достичь интересных результатов во взаимодействии человека с роботом.

Передача данных в RobotC

Некоторые проблемы неизбежно возникнут при программировании тех же алгоритмов в RobotC. Дело в том, что целочисленные значения в нем могут находиться только в диапазоне –32768...32767. А при кодировании нам неизбежно приходится выходить за его пределы. Однако расширенная поддержка стандарта Bluetooth сполна окупает это небольшое неудобство. Одним из преимуществ является возможность пакетной передачи сразу трех 16-битных знаковых чисел:

```
sendMessageWithParm(nMessageID, nParm1, nParm2);
```

Важное замечание: параметр nMessageID не может быть нулевым, ноль означает отсутствие сообщения в почтовом ящике получателя.

При получении сообщения первым рассматривается предопределенная переменная message (в Robolab была *m*), затем MessageParm[1] и MessageParm[2]. Значение MessageParm[0] соответствует message.

Рассмотрим пример программы передачи, в которой первым параметром передается увеличенное на единицу значение датчика касания, а вторым и третьим — показания энкодеров самодельного джойстика.

```
task main()
{
  bFloatDuringInactiveMotorPWM=true;
  nMotorEncoder[mA]=0;
  nMotorEncoder[mB]=0;
  if (nBTCurrentStreamIndex >= 0)
    while(1)
    {
      while(bBTBusy)
        wait1Msec(5);
      sendMessageWithParm(SensorValue[S1]+1,
        nMotorEncoder[motorA], nMotorEncoder[motorB]);
    }
}
```

Программа приема будет управлять двумя моторами и выводить все полученные сообщения на экран NXT. Перед приемом каждого нового сообщения происходит очистка очереди сообщений:

```
task main()
{
  long v,u,k;
  while(1)
  {
    while (bQueuedMsgAvailable()) // Очистка очереди
    {
      word temp;
```

```
      ClearMessage(); // Может не работать в RobotC 3.0
      temp = message;
    }
    if (message!=0)
    {
      k=message-1;    // Сообщение приходит на 1 больше
      v=messageParm[1];
      u=messageParm[2];
      motor[motorA]=v+u;
      motor[motorB]=v-u;
      nxtDisplayTextLine(0, "k=%d", k);
      nxtDisplayTextLine(1, "v=%d", v);
      nxtDisplayTextLine(2, "u=%d", u);
      wait1Msec(10);
    }
  }
}
```

В среде RobotC реализованы команды подключения к другому устройству в программном режиме, а также ряд других возможностей Bluetooth, позволяющих повысить стабильность работы. Примеры находятся в папке Sample Programs\NXT\Bluetooth Communication.

Роботы-манипуляторы

Использование манипуляторов стало обыденным явлением еще в 20-м веке. На сегодняшний день ни одно крупное промышленное производство не обходится без них. Кроме того, манипуляторы устанавливают и на мобильных роботах, чтобы расширить возможности управления в труднодоступных для человека местах. При однообразных и монотонных действиях робот также может заменить человека, например раскладывать пирожные по коробочкам.

Итак, определим манипулятор как управляемое устройство, предназначенное для выполнения сложных действий, аналогичных движениям руки человека. В том числе, это механизм для управления положением предметов.

Стрела манипулятора

Для освоения управления манипулятором следует вернуться к контролю положения двигателя с помощью П-регулятора, которое описано в главе «Алгоритмы управления». Только конструкция будет несколько отличаться от описанной (рис. 8.148—8.149).

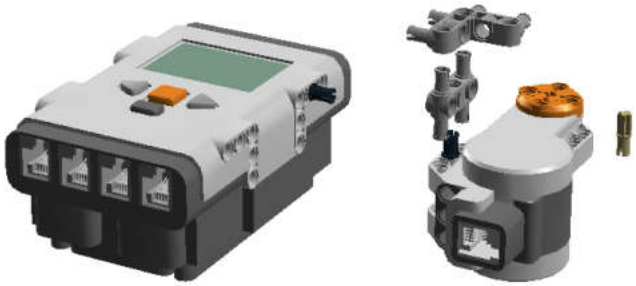


Рис. 8.148. Крепление мотора к корпусу NXT горизонтально на уровне поверхности. Штифт-полуось вставляется в вал мотора снизу.

Рис. 8.149. Первое колено манипулятора.

Стрела манипулятора расположена горизонтально. Алгоритм управления состоит из двух параллельных задач. В первой работает П-регулятор, который удерживает мотор в положении alpha. Во второй положение alpha изменяется со временем (рис. 8.150).

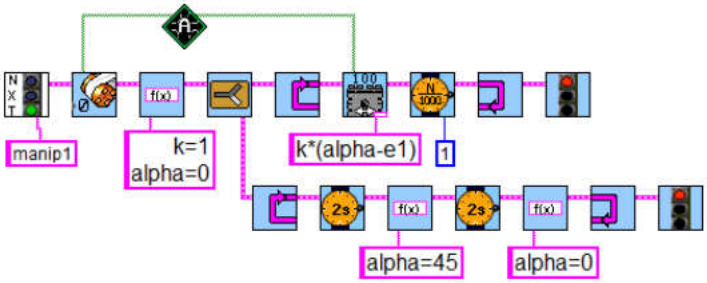


Рис. 8.150. Управление горизонтальным положением стрелы манипулятора.

Учитывая, что нулевое положение определяется на старте, задайте отрицательный угол во второй параллельной задаче и осуществите последовательный переход через нуль. Если все получается, следует приступить к установке второго мотора с захватом.

Манипулятор с захватом

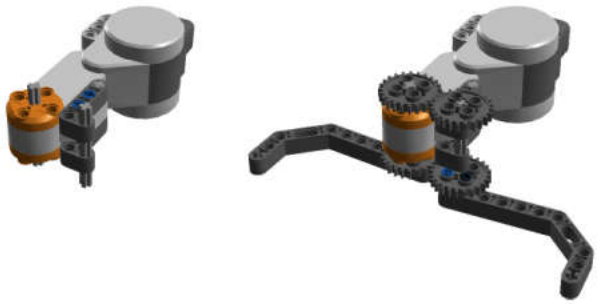


Рис. 8.151. Установка захвата на второй мотор.

Рис. 8.152. Мотор с захватом закрепляется на диске первого мотора.

Программирование робота с двумя степенями свободы (рис. 8.151—8.152) осуществляется аналогично. Новая переменная *beta* будет определять положение второго мотора. Расширьте вторую задачу, подобрав подходящие значения для открывания и закрывания захвата (рис. 8.153). Не забывайте, что стартовое положение определяет все.

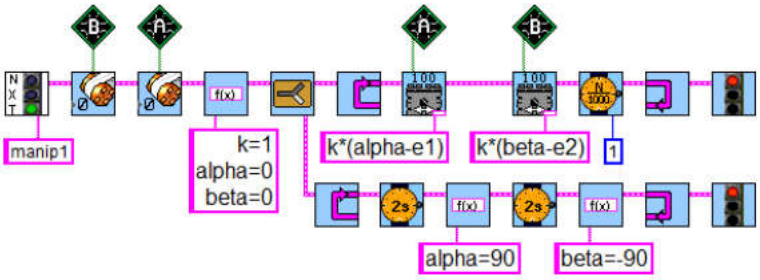


Рис. 8.153. Заготовка для управления двумя моторами на основе П-регуляторов. Необходимо продолжить цикл в параллельной задаче.

Задача для манипулятора с двумя степенями свободы формулируется так. Робот открывает захват и поворачивается в стартовое положение, где уже стоит стаканчик. Захватив стаканчик, робот перемещает его в финишное положение и, открыв захват, возвращается на старт. Задача человека — вовремя менять стаканы.

Дискретный регулятор

Можно заметить, что движения выполняются роботом слишком быстро. Еще один интересный регулятор позволяет разбить перемещение от точки к точки на несколько равных промежутков. Его можно назвать дискретным. Рассмотрим пример дискретного регулятора для двух моторов (рис. 8.154).

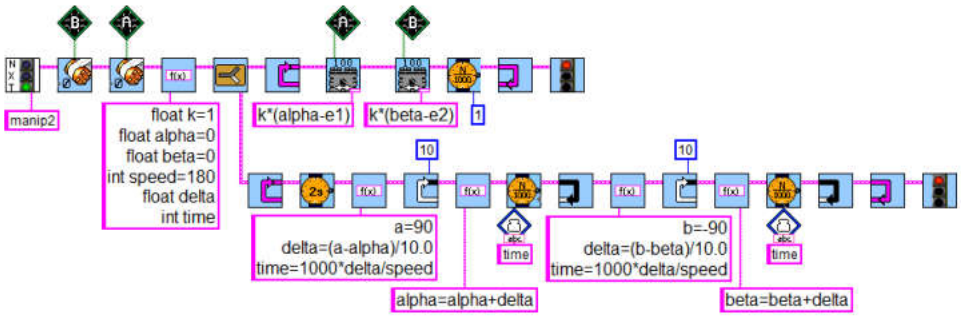


Рис. 8.154. Дискретный регулятор для плавного перемещения манипулятора.

Недостатком данного алгоритма является то, что действия приходится выполнять последовательно. Однако, применив параллельные задачи, можно добиться одновременного выполнения действий разными моторами. Но есть и другие способы.

В приведенном ниже примере на языке RobotC демонстрируется не только возможность одновременного управления моторами с помощью П-регуляторов, но и обеспечивается их плавное движение:

```
int alpha=0, beta=0;
bool flag=true;
task preg()
{
  float ka=1, kb=1, a=0, b=0, da=1, db=1;
  nNxtExitClicks=2;
  while(flag)
  {
    a=a+sgn(alpha-a)*da;
    b=b+sgn(beta-b)*db;
    motor[motorA]=(a-nMotorEncoder[motorA])*ka;
    motor[motorB]=(b-nMotorEncoder[motorB])*kb;
```

```
    if(nNxtButtonPressed==0)
      flag=false;
    wait1Msec(10);
  }
}
task main()
{
  nMotorEncoder[motorA]=nMotorEncoder[motorB]=0;
  StartTask(preg);
  while(flag)
  {
    alpha=90;
    beta=90;
    wait1Msec(2000);
    alpha=0;
    beta=-90;
    wait1Msec(2000);
  }
}
```

В данном примере переменные *a* и *b* показывают изменяющиеся во времени уставки моторов, а переменные *da* и *db* — их приращение. Изменяя приращение, можно добиться ускорения или замедления стрелы манипулятора. Задержка в регуляторе повышена до 10 мс, что естественно, поскольку мы стремимся к понижению скорости. Таким образом, за каждый промежуток времени происходит не только управление моторами в направлении желаемого положения, но и сдвиг этого положения на небольшую величину. Нетрудно рассчитать, что при заданных в программе параметрах угловая скорость моторов будет составлять 100 градусов в секунду.

Изменения углов в основной программе представляют интерес только в качестве демонстрации возможностей одновременной работы. Установив свою последовательность, можно добиться практически идеального выполнения поставленной для данного манипулятора задачи.

Три степени свободы

Если задача с двумя моторами выполнена успешно, можно переходить к следующему уровню сложности: три степени свободы. Следует снять мотор с захватом и на его место поставить другой мотор — промежуточное звено (рис. 8.155—8.156).

Манипулятор получается довольно тяжелым и падает, не выдерживая собственного веса. Чтобы этого не происходило, следует установить опоры, прикрепив их к мотору и контроллеру NXT. Поскольку манипулятор будет переставлять стаканчики, на краях опор следует закрепить специальные стержни (рис. 8.157—8.159).

Рис. 8.155. Замена захвата на промежуточное звено.

Рис. 8.156. Установка мотора с захватом на оранжевый диск второго мотора.

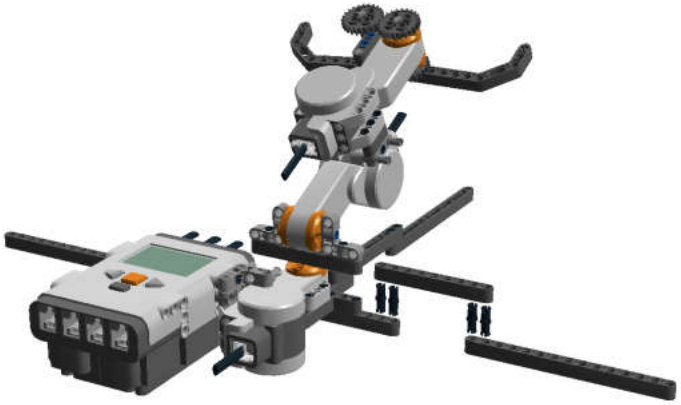


Рис. 8.157. Установка опор для манипулятора.

Рис. 8.158. Установка стержней для стаканчиков в трех позициях.

Рис. 8.159. Конструкция манипулятора завершена.

Управление тремя моторами задается аналогично двум (рис. 8.153). Читателю предоставляется возможность самостоятельно подобрать ключевые значенич *alpha*, *beta* и *gamma*. Пусть моторы будут подключены последовательно и *gamma* соответствует мотору с захватом.

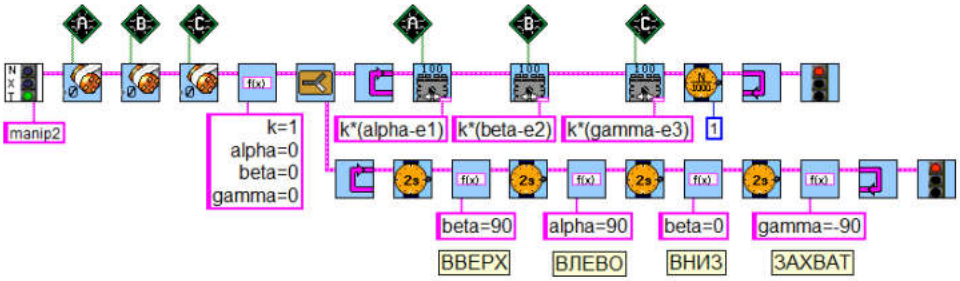


Рис. 8.160. Алгоритм управления тремя моторами требует доработки в нижней параллельной задаче.

Задача управления будет состоять в следующем. Робот открывает захват, поднимает второй мотор, перемещает первый мотор в направлении стаканчика, опускает второй мотор, выполняет захват, поднимает второй мотор, поворачивает первый мотор в сторону свободного стержня, опускает второй мотор, открывает захват. И так далее в цикле.

ПД-регулятор для манипулятора

Движения робота получаются резкими, это связано с особенностью работы П-регулятора. Помимо приведенной ранее дискретной системы есть возможность применить управление моторами на ПД-регуляторе (рис. 8.161), тогда характер движений изменится. Коэффициенты *k*1, *k*2, *k*3, *kd*1, *kd*2 и *kd*3 следует подобрать для каждого мотора отдельно.

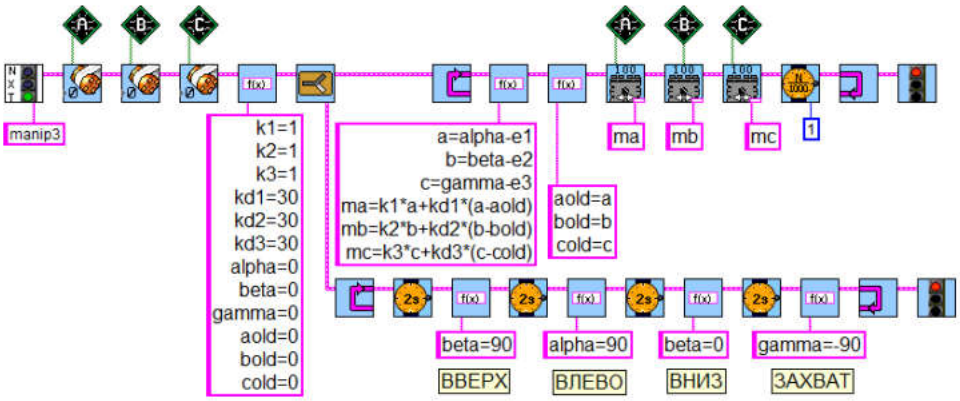


Рис. 8.161. Управление манипулятором с использованием ПД-регулятора.

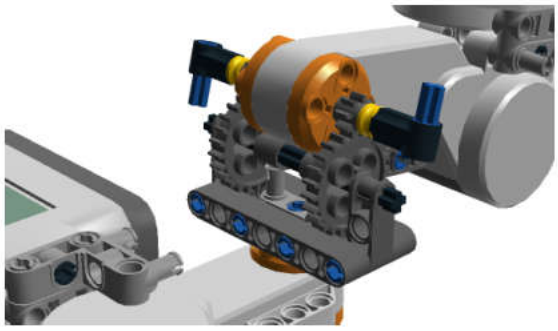


Рис. 8.162. Передача 3 : 1 для разгрузки второго мотора.

Следующим этапом может стать установка передачи на первый мотор, который работает на пределе возможностей, поскольку на него ложится самая большая нагрузка. На рис. 8.162 предложено решение. Должен ли измениться алгоритм управления мотором с передачей?

Шестиногий робот

В главе 2 описаны два четвероногих робота, которые могут передвигаться под управлением одного мотора. Но самостоятельно выполнить поворот они не смогут. Шестиногий робот отличается способностью к повороту. Его движение основано на трех постоянных точках опоры. То есть шесть ног разделены на две тройки, расположенных в вершинах треугольников. Эти «треугольники» поднимаются и опускаются в противофазе. Таким образом, робот постоянно опирается на три точки и ему остается только переносить центр тяжести в направлении движения.

Чтобы реализовать механическую составляющую, приступим к конструкции (рис. 8.163—8.169). Некоторые детали в ней могут быть заменены на аналогичные по функционалу, но самая специфическая часть — это гладкие штифты, которые следует поставить во все подвижные элементы.

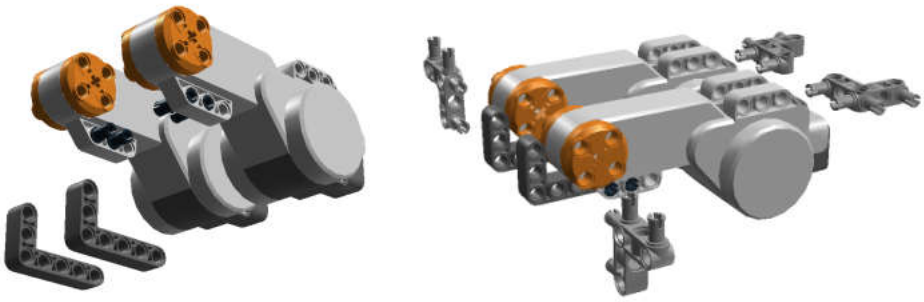


Рис. 8.163. Угловые балки 3 × 5 крепятся к моторам на двумодульные штифты.

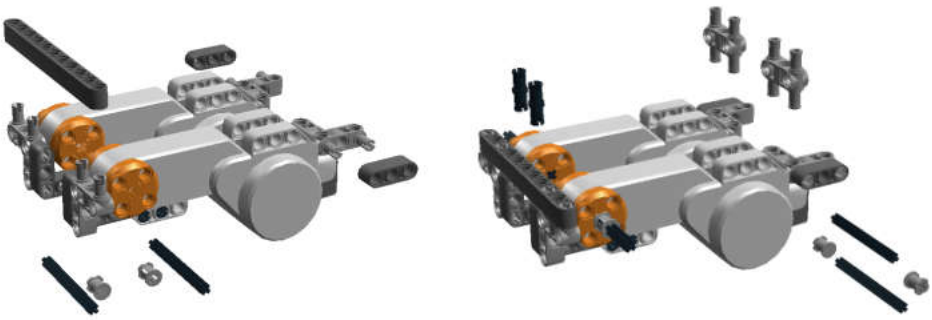


Рис. 8.164. Спереди моторы соединяются 11-модульной балкой. Все оси имеют длину 6 модулей.

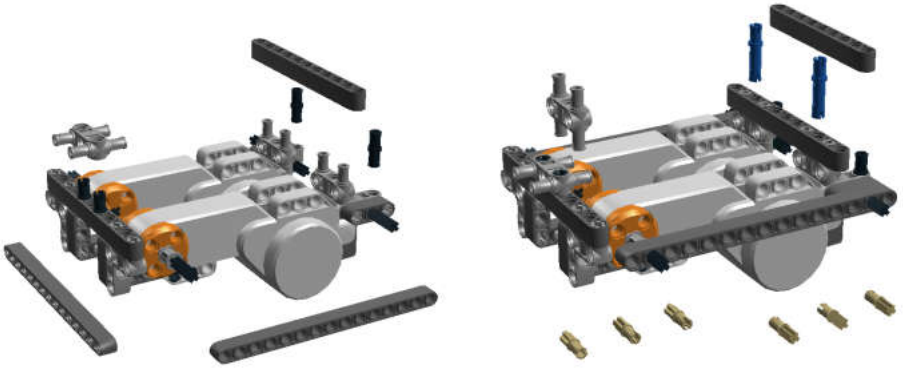


Рис. 8.165. Крепление сзади на 9-ти и 11-модульную балку, а по бокам несущие балки в 15 модулей.

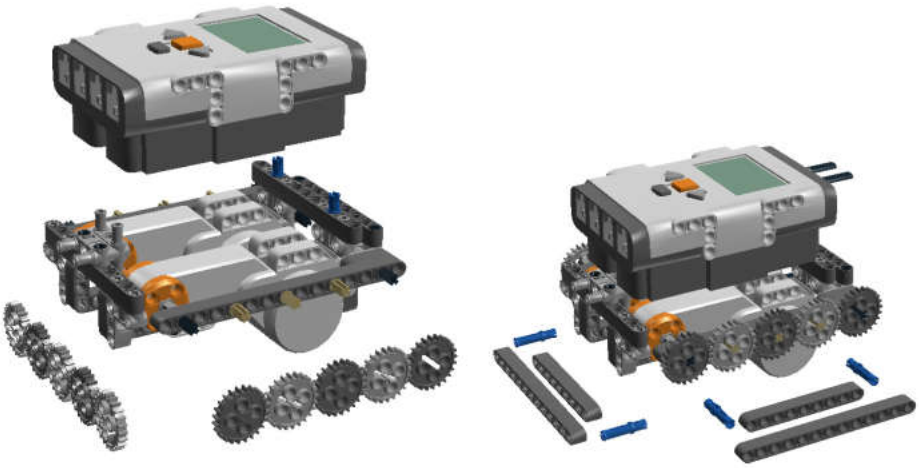


Рис. 8.166. Особое внимание следует уделить повороту нечетных шестеренок.

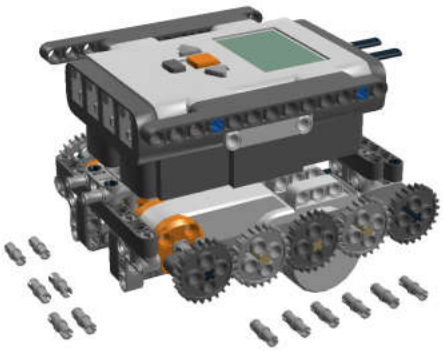


Рис. 8.167. По три гладких штифта следует вставить в боковые балки и нечетные шестеренки с каждой стороны.

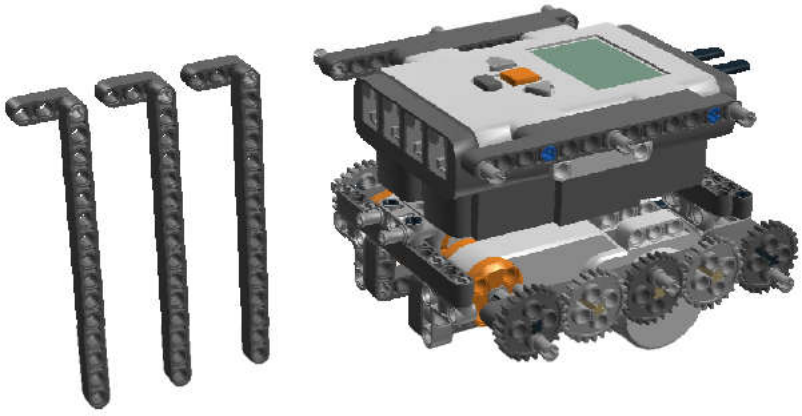


Рис. 8.168. Перед установкой ног следует убедиться, что штифты в шестернях стоят в шахматном порядке.

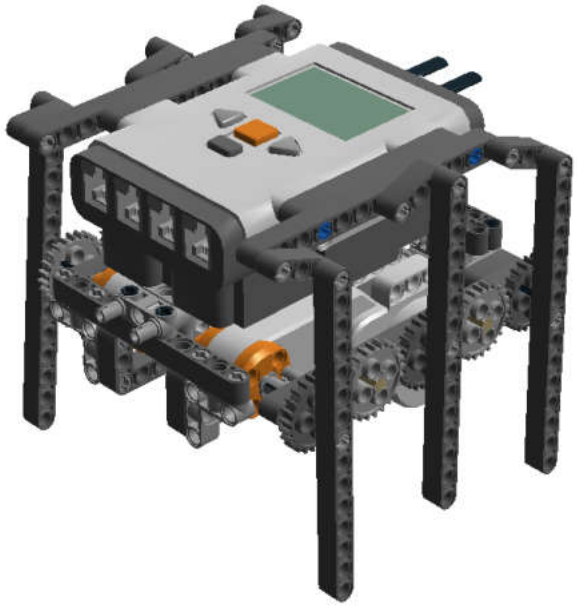


Рис. 8.169. Шестиногий робот готов. Для надежности стоит поставить ему защитные рамки из балок по бокам.

Перед сложным программированием робота можно запустить его со стандартной программой из NXT Program, в которой присутствует синхронизация моторов B и C. Только перед стартом следует убедиться, что ноги робота с правой и левой стороны ориентированы в противофазе (рис. 8.170).

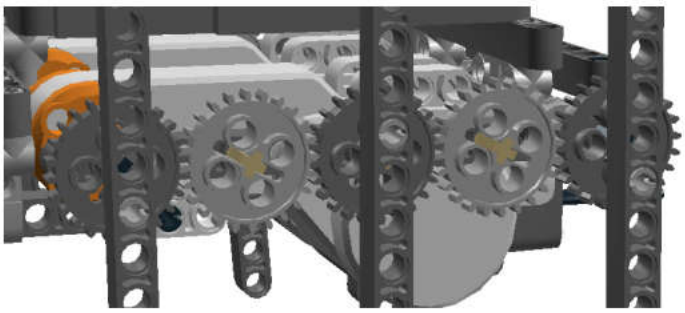


Рис. 8.170. Правильная синхронизация шестеренок на старте — ключ к движению.

Если робот правильно зашагал вперед, можно приступать к программированию поворотов. Алгоритм основан на обычной синхронизации двух моторов с добавлением возмущающего воздействия *delta*, которое будет изменяться из параллельной задачи и тем самым вызывать разнонаправленное движение конечностей, т. е. поворот. При этом величина *delta* должна быть кратна 360 градусам, чтобы не нарушить синхронизацию, заданную на старте. Например, приращение *delta* может составлять $360 \cdot 4$, что обеспечит поворот робота на 90 градусов (рис. 8.171). Правда после 22 поворотов произойдет переполнение значения переменной, но и от этого при желании можно защититься.

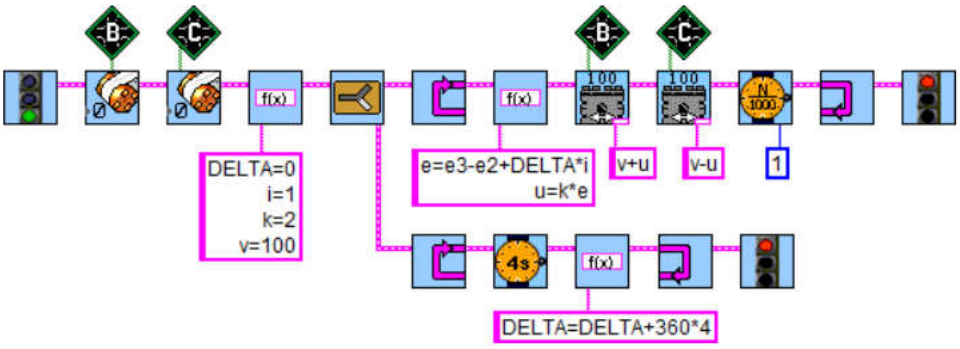


Рис. 8.171. Алгоритм движения «шестинога» по квадрату.

Алгоритм управления роботом основан на П-регуляторе и управляющей задаче, в которой задается изменение возмущающего воздействия через определенные промежутки времени. Получив «толчок» из параллельной задачи, робот некоторое время выполняет поворот, приводя значение отклонения к нулю, а затем продолжает идти прямо. Переменная *i* может пригодиться при использовании передаточного отношения между мотором и шестерней, к которой прикреплена конечность.

Следующим этапом на роботе размещается датчик расстояния. Теперь он готов совершать поворот, увидев препятствие. На выполнение поворота роботу дается 2 секунды, после чего он снова начинает отслеживать объекты (рис. 8.172).

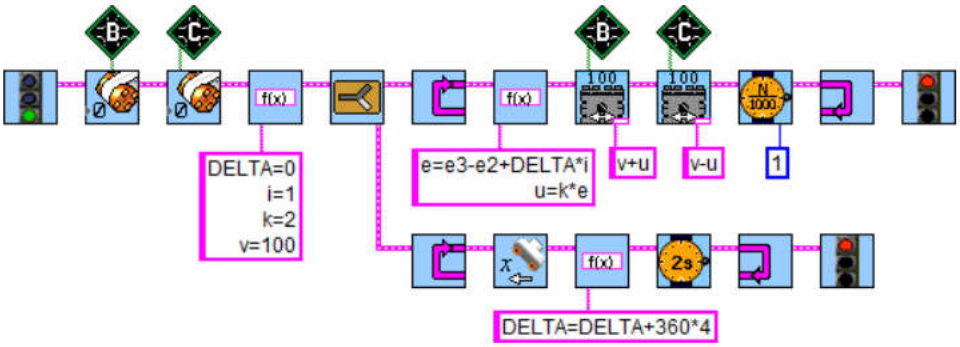


Рис. 8.172. Введение датчика расстояния для контроля препятствий.

```
int delta=0, v=100;
float i=1, k=2;
task preg()
{
  while(true)
  {
    int e=nMotorEncoder[motorC]-
          nMotorEncoder[motorB]+delta*i;
    int u=e*k;
    motor[motorB]=v+u;
    motor[motorC]=v-u;
    wait1Msec(1);
  }
}
task main()
{
  nMotorEncoder[motorC]=nMotorEncoder[motorB]=0;
  StartTask(preg);
  while (true)
  {
    while(SensorValue[S1]>25) wait1Msec(1);
    delta=delta+360*4;
    wait1Msec(2000);
  }
}
```

С датчиком расстояния и защитным корпусом шагающий робот готов стать почти полноценным членом общества. По крайней мере, он сможет участвовать в состязаниях «Гонки шагающих роботов» не без шансов на успех.

Заключение

Если читатель добрался до Заключения, проделав все опыты, изложенные в этой книге, можно быть уверенным, что впереди у него множество собственных находок и изобретений. Мы коснулись лишь малой части замечательной науки, которая все больше становится современной реальностью.

Не останавливайтесь на достигнутом, находите новые задачи и решения, создавайте своих оригинальных роботов. Быть может, эти небольшие открытия сослужат хорошую службу и нашей стране, и всему человечеству.

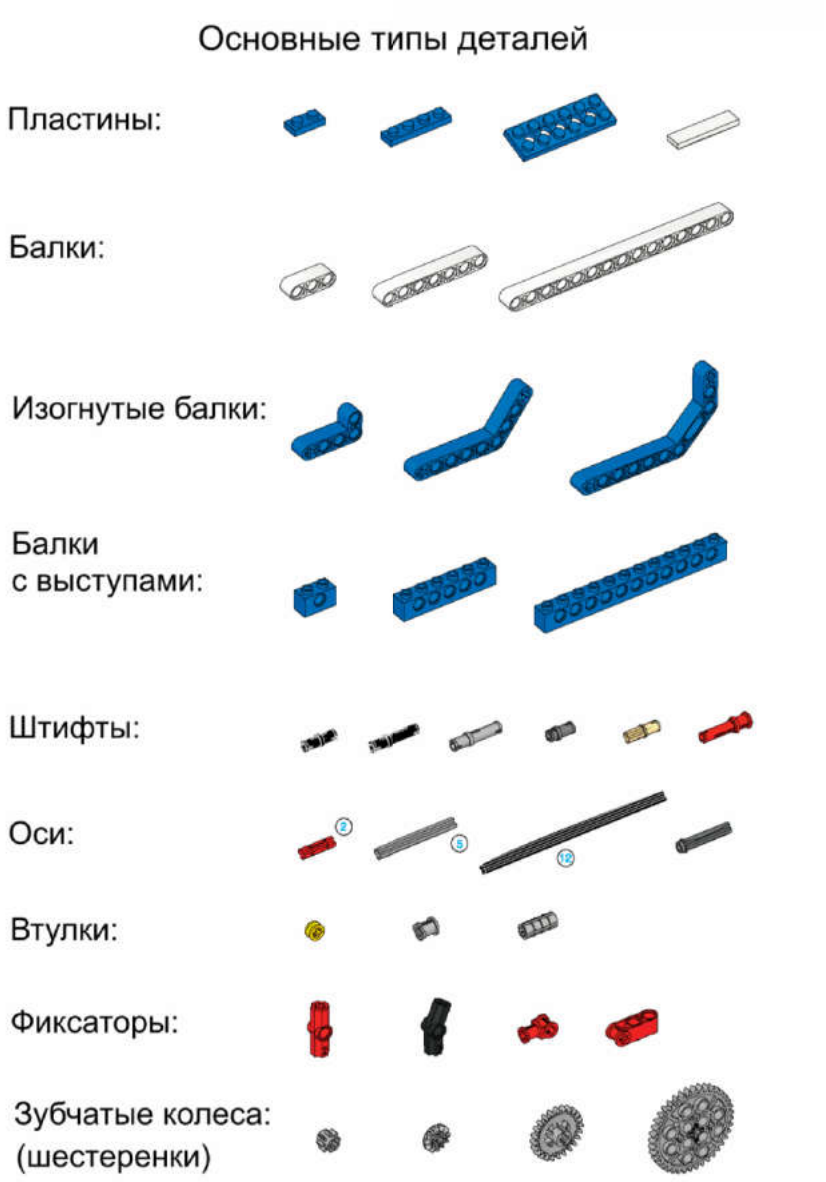
Ваши вопросы и предложения автору отправляйте по адресу robobook@mail.ru.

Литература

1. *Ананьевский М. С., Болтунов Г. И., Зайцев Ю. Е., Матвеев А. С., Фрадков А. Л., Шиегин В. В.* Санкт-Петербургские олимпиады по кибернетике. Под ред. *Фрадкова А. Л., Ананьевского М. С.* СПб.: Наука, 2006.
2. *Boogaarts M., Torok R., Daudelin J.*, et al. The LEGO Mindstorms NXT Idea Book. San Francisco: No Starch Press, 2007.
3. *Isogawa Y.* LEGO Technic Tora no Maki, Version 1.00 Isogawa Studio, Inc., 2007, //Электронный ресурс [*http://www.isogawastudio.co.jp/legostudio/toranomaki/en/*].
4. Constructopedia NXT Kit 9797, Beta Version 2.1, Center for Engineering Educational Outreach, Tufts University, 2008, //Электронный ресурс [*http://www.legoengineering.com/library/doc_download/150-nxt-constructopedia-beta-21.html*].
5. *Kelly J. F.* Lego Mindstorms NXT. The Mayan adventure. Apress, 2006.
6. *Wang E.* Engineering with LEGO Bricks and ROBOLAB. Third edition. College House Enterprises, LLC, 2007.
7. *Perdue D. J.* The Unofficial LEGO MINDSTORMS NXT Inventor's Guide. San Francisco: No Starch Press, 2007.
8. *Белиовская Л.Г., Белиовский А.Е.* Программируем микрокомпьютер NXT в LabVIEW. М: ДМК Пресс, 2010.
9. *Азимов А.* Я, робот. Серия: Библиотека приключений. М: Эксмо, 2002.

Приложения

П.1. Названия деталей



П.2. Правила состязаний

Регламент соревнований роботов «Кегельринг»[1]

(по версии Ассоциации спортивной робототехники)

1. Условия состязания:

— в наиболее короткое время робот, не выходя за пределы круга, очерчивающего ринг, должен вытолкнуть расположенные в нем кегли;

— на очистку ринга от кеглей дается максимум две минуты;

— если робот полностью выйдет за линию круга более чем на 5 секунд, попытка не засчитывается;

— во время проведения состязания участники команд не должны касаться роботов, кеглей или ринга.

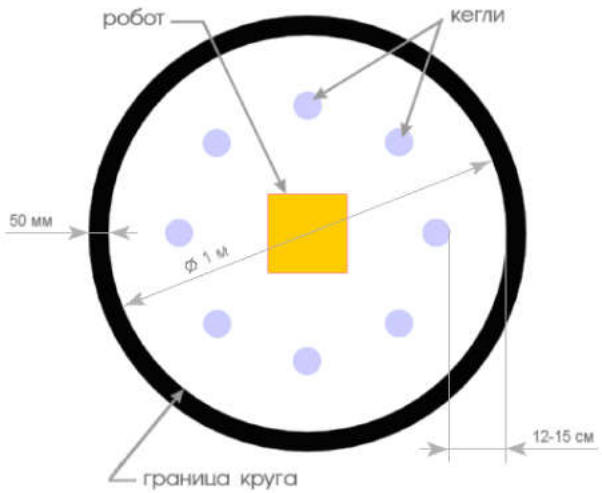


Рис. П.2.1. Поле для кегельринга.

2. Ринг:

— цвет ринга — светлый;

— цвет ограничительной линии — черный;

— диаметр ринга 1 м (белый круг);

— ширина ограничительной линии 50 мм;

3. Кегли:

— кегли — жестяные цилиндры, изготовленые из пустых стандартных жестяных банок, используемых для напитков;

[1] Идея взята с сайта *http://www.myrobot.ru*

— диаметр кегли 70 мм;
— высота кегли 120 мм;
— вес кегли — не более 50 г.

4. Робот:
— максимальная ширина робота 20 см, длина — 20 см;
— высота и вес робота не ограничены;
— робот должен быть автономным;
— во время соревнования размеры робота должны оставаться неизменными и не должны выходить за пределы 20 × 20 см;
— робот не должен иметь никаких приспособлений для выталкивания кеглей (механических, пневматических, вибрационных, акустических и др.);
— робот должен выталкивать кегли только своим корпусом;
— запрещено использование каких-либо клейких приспособлений на корпусе робота для сбора кеглей.

5. Игра:
— робот помещается строго в центр ринга.
— на ринге устанавливается восемь кеглей.
— кегли равномерно расставляются внутри окружности ринга. На каждую четверть круга должно приходиться не более двух кеглей. Кегли ставятся не ближе 12 и не далее 15 см от черной ограничительной линии. Перед началом игры участник состязания может поправить расположение кеглей. Окончательная расстановка кеглей принимается судьей соревнования;
— главная цель робота состоит в том, чтобы вытолкнуть кегли за пределы круга, ограниченного линией;
— кегля считается вытолкнутой, если никакая ее часть не находится внутри белого круга, ограниченного линией;
— один раз покинувшая пределы ринга кегля считается вытолкнутой и может быть снята с ринга в случае обратного закатывания;
— робот должен быть включен или инициализирован вручную в начале состязания по команде судьи, после чего в его работу нельзя вмешиваться. Запрещено дистанционное управление или подача роботу любых команд.

6. Правила отбора победителя:
— каждой команде дается не менее двух попыток (точное число определяется судейской коллегией в день проведения соревнований);
— в зачет принимается лучшее время из попыток или максимальное число вытолкнутых кеглей за отведенное время;
— победителем объявляется команда, чей робот затратил на очистку ринга от кеглей наименьшее время, или если ни одна команда не справилась с полной очисткой ринга — команда, чей робот вытолкнул за пределы ринга наибольшее количество кеглей.

П.3. Интернет-ресурсы по Lego Mindstorms NXT

— *http://www.mindstorms.com* (официальный сайт компании Lego)

— *http://www.mindstorms.su* (неофициальный российский сайт Lego Mindstorms)

— *http://learning.9151394.ru* (содержит вводный курс Lego Mindstorms NXT на русском языке)

— *http://www.lugnet.com* (форум пользователей Lego Mindstorms NXT)

— *http://www.nxtprograms.com* (примеры разработок роботов из Lego Mindstorms NXT)

— *http://www.legoengineering.com* (поддержка пользователей Mindstorms)

— *http://nnxt.blogspot.ru/* (робототехника для школ и вузов Нижнего Новгорода)

— *http://www.isogawastudio.co.jp/legostudio/toranomaki/en/* (LEGO Technic Tora no Maki, энциклопедия конструирования)

Языки и среды программирования для Lego Mindstorms NXT

— RobotC: *http://www.robotc.net*

— NBC/NXC (Next Byte Codes & Not eXactly C): компилятор и документация к NBC
http://bricxcc.sourceforge.net/nbc/

— Интегрированная среда разработки Bricxcc
http://bricxcc.sourceforge.net/

— LEJOS: Java for Lego Mindstorms: *http://lejos.sourceforge.net/*

— Среда LabVIEW для Lego Mindstorms NXT:
www.ni.com/mindstorms

— Обновление для Robolab 2.9 до версии 2.9.4:
http://www.legoengineering.com/patches/RL294PowerPatch_PC.zip

— Обновление для Robolab 2.9.4 с поддержкой новых датчиков сторонних производителей: *http://legoengineering.com/library/cat_view/41-applications-patches-a-firmware/43-robolab.html*

— QReal Robots, среда программирования роботов с 2D-симулятором, разработанная на матмехе СПбГУ: *http://qreal.ru*

Правила состязаний роботов

— *http://www.myrobot.ru/sport* (Мой робот: роботы, робототехника, микроконтроллеры)

— *http://railab.ru/* (лаборатория робототехники и искуственного интеллекта Политехнического музея)

— *http://wroboto.ru/* (Международные состязания роботов)

— *http://www.wroboto.org/ (*Всемирная олимпиада роботов)

— *http://239.ru/robot* (Центр робототехники физико-математического лицея №239 Центрального района Санкт-Петербурга)

Неофициальный гид изобретателя Lego Mindstorms NXT

Интернет-ресурсы по Lego Mindstorms NXT из книги David Perdue, «The Unofficial Lego Mindstorms NXT Inventor's Guide». см. сайт http://nxtguide.davidjperdue.com/

Общие ресурсы

— Обновления программ (*http://mindstorms.lego.com/en-us/support/files/default.aspx*)

— LUGNET (*http://www.lugnet.com*)

— MOC pages (*http://www.mocpages.com*)

— Brickshelf (*http://www.brickshelf.com*)

— Peeron LEGO Inventories (*http://www.peeron.com*)

— Brickset (*http://www.brickset.com*)

— NXT Programs: Fun Projects for your LEGO MINDSTORMS NXT (*http://www.nxtprograms.com/index.html*)

— MINDSTORMS NXT Building Instructions (*http://ricquin.net/lego/instructions/*)

— Technica (*http://isodomos.com/technica/technica.html*)

— Blackbird's Technicopedia (*http://www.ericalbrecht.com/technic*)

Ресурсы для программистов

— Programming Solutions for the LEGO MINDSTORMS NXT: Which approach is best for you? NBC and NXC (*http://bricxcc.sourceforge.net/nbc*)

— NBC Debugger for NXT (*http://www.sorosy.com/lego/nxtdbg*)

— BricxCC (*http://bricxcc.sourceforge.net*)

— Programmable Brick Utilities (*http://bricxcc.sourceforge.net/utilities.html*)

— leJOS NXJ (*http://lejos.sourceforge.net*)

— RobotC (*http://www.robotc.net*)

— Writing Efficient NXT-G Programs: *http://www.firstlegoleague.org/sitemod/upload/Root/WritingEfficientNXTGPrograms2.pdf*

— OnBrick NXT Remote Control (*http://www.pspwp.pwp.blueyonder.co.uk/science/robotics/nxt/)*

— NXTender (*http://www.tau.ac.il/~stoledo/lego/NXTender*)

— NXT Programming Software
(*http://www.teamhassenplug.org/NXT/NXTSoftware.html*)

Ресурсы для Bluetooth

— MINDSTORMS Bluetooth Resources
http://www.mindstorms.com/bluetooth
— NXTBluetoothCompatibilityList:
http://www.vialist.com/users/jgarbers/NXTBluetoothCompatibilityList
— Analysis of the NXT Bluetooth-Communication Protocol:
http://www.tau.ac.il/~stoledo/lego/btperformance.html

NXT-Блоги

— The NXT STEP (*http://www.thenxtstep.com*)
— nxtasy.org (*http://www.nxtasy.com*)

Ресурсы по автоматизированному конструированию
(LEGO computer-aided design resources):

— LEGO Digital Designer (*http://ldd.lego.com*)
— Google SketchUp NXT Parts Library:
http://groups.google.com/group/LegoTechnicandMindstormsNXTParts
— LDraw (*http://www.ldraw.org*)
— Tutorial: Setting up LDraw to Create Virtual NXT Robots: from *http://nxtblog.davidjperdue.com*
— LeoCAD (*http://www.leocad.org*)
— Bricksmith (*http://bricksmith.sourceforge.net*)
— L3P (*http://www.hassings.dk/l3/l3p.html*)
— LDView (*http://ldview.sourceforge.net*)

Методы конструирования (Building techniques)

— NXT-based Creations
(*http://legoengineering.com/library/cat_view/30-building-instructions/38-nxt-based-creations.html*)
— LEGO Education Constructopedia:
http://legoengineering.com/library/doc_details/150-nxt-constructopedia-beta-21.html

Изучаем геометрию Lego:

— *http://www.syngress.com/book_catalog/174_lego_robo/chapter_01.htm*
— LEGO Design (*http://www.owlnet.rice.edu/~elec201/Book/legos*)

— Sergei Egorov's LEGO Geartrains (*http://www.malgil.com/esl/lego/geartrains.html*)

Образовательные ресурсы

— LEGO Education (*http://www.legoeducation.com*)
— MINDSTORMS Education NXT blog: http://www.legoeducation.com/community/9/blogs/nxt/default.aspx
— LEGO ED West (*http://www.legoedwest.com*)
— LEGO Engineering (*http://www.legoengineering.com*)
— FIRST LEGO League (*http://www.firstlegoleague.org*)
— US FIRST Curriculum Collection: *http://www.usfirst.org/community/*
— Robotics Academy (*http://www-education.rec.ri.cmu.edu*)

Наборы Lego, детали Lego и заказные детали (custom hardware)

— LEGO Store (*http://shop.lego.com*)
— LEGO Education Store (*http://www.legoeducation.us*)
— BrickLink (*http://www.bricklink.com*)
— HiTechnic (*http://www.hitechnic.com*)
— Mindsensors.com (*http://www.mindsensors.com*)

Хранение деталей Lego

— Robotics Learning Store (*http://www.roboticslearning.com/store*)
— Plano Molding Company (*http://www.planomolding.com*)

Персональные вебсайты

— David J. Perdue (*http://www.davidjperdue.com*)
— Philippe Hurbain (*http://www.philohome.com*)
— Dave Astolfo (*http://www.astolfo.com*)
— Daniele Benedettelli (*http://daniele.benedettelli.com*)
— Michael Gasperi (*http://extremenxt.com/lego.htm*)
— Matthias Paul Scholz (*http://mynxt.matthiaspaulscholz.eu*)
— Steve Hassenplug (*http://www.teamhassenplug.org*)
— Laurens Valk (*http://www.laurensvalk.com*)
— Jürgen Stuber (*http://www.jstuber.net*)
— Mario Ferrari (*http://www.marioferrari.org/lego.html*)
— Miguel Agullo (*http://miguelagullo.net/technicpuppy/*)

События Lego

- World Robot Olympiad (*http://www.wroboto.org*)
- LEGO World (*http://www.legoworld.nl*)
- BrickFest (*http://www.brickfest.com*)
- NWBrickCon (*http://www.nwbrickcon.org*)
- BrickFair (*http://www.brickfair.com*)

Научное издание

Сергей Александрович Филиппов

РОБОТОТЕХНИКА ДЛЯ ДЕТЕЙ И РОДИТЕЛЕЙ
Издание 3-е, дополненное и исправленное
Утверждено к печати
Ученым советом Института проблем машиноведения РАН

Редактор издательства А. Б. Иванова
Художник О. Скворцова
Книга печатается с оригинал-макета,
подготовленного автором

Санкт-Петербургская издательская фирма «Наука» РАН
199034, Санкт-Петербург, Менделеевская линия, 1
E-mail: main@nauka.nw.ru
Internet: www.naukaspb.spb.ru
Лицензия ИД № 02980 от 06 октября 2000 г.
Подписано к печати 30.01.2013. Формат 70 × 90 1/16.
Бумага офсетная. Печать офсетная. Гарнитура Таймс.
Объем 20 усл. печ. л. Тираж 3000 экз. Стр. 319

Отпечатано в типографии ООО «Дитон»
Санкт-Петербург, Б. Сампсониевский пр., 60, литер М
Тел.: (812) 333-15-42
Факс: (812) 333-15-41

ISBN 978-5-02-038200-8