Изучение робототехники с помощью **PYTHON**

Создайте с нуля автономный мобильный робот ChefBot с использованием метаоперационной системы Robot Operating System (ROS), языка Python и библиотеки алгоритмов компьютерного зрения OpenCV!

Рассматриваемые темы:

- основные понятия ROS;
- принципы работы робота с дифференциальным приводом;
- знакомство со средой моделирования робота Gazebo;
- проектирование оборудования и схем ChefBot;
- согласование приводов и датчиков с контроллером робота;
- введение в OpenCV, OpenNI и PCL;
- исследование работы различных 3D-камер глубины в ROS;
- реализация автономной навигации для ChefBot;
- создание GUI (графического пользовательского приложения) использованием библиотеки Qt и языка Python.

Для работы с книгой вам понадобятся:

- компьютер с установленной системой Ubuntu;
- свободное ПО: ROS, Gazebo, LibraCad, MeshLab, Blender;
- комплектующие (двигатель, кронштейны, колеса, др.) с AliExpress;
- встроенный контроллер Tiva C LaunchPad;
- ультразвуковые датчики, акселерометр и гироскоп, а также динамик и микрофон;
- сенсорный игровой контроллер Kinect либо датчик глубины Orbbec;
- аккумуляторная батарея 12 В, 10 А·ч.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
e-mail: books@alians-kniga.ru

Packt›

ДМК ПРЕСС издательство
www.дмк.рф

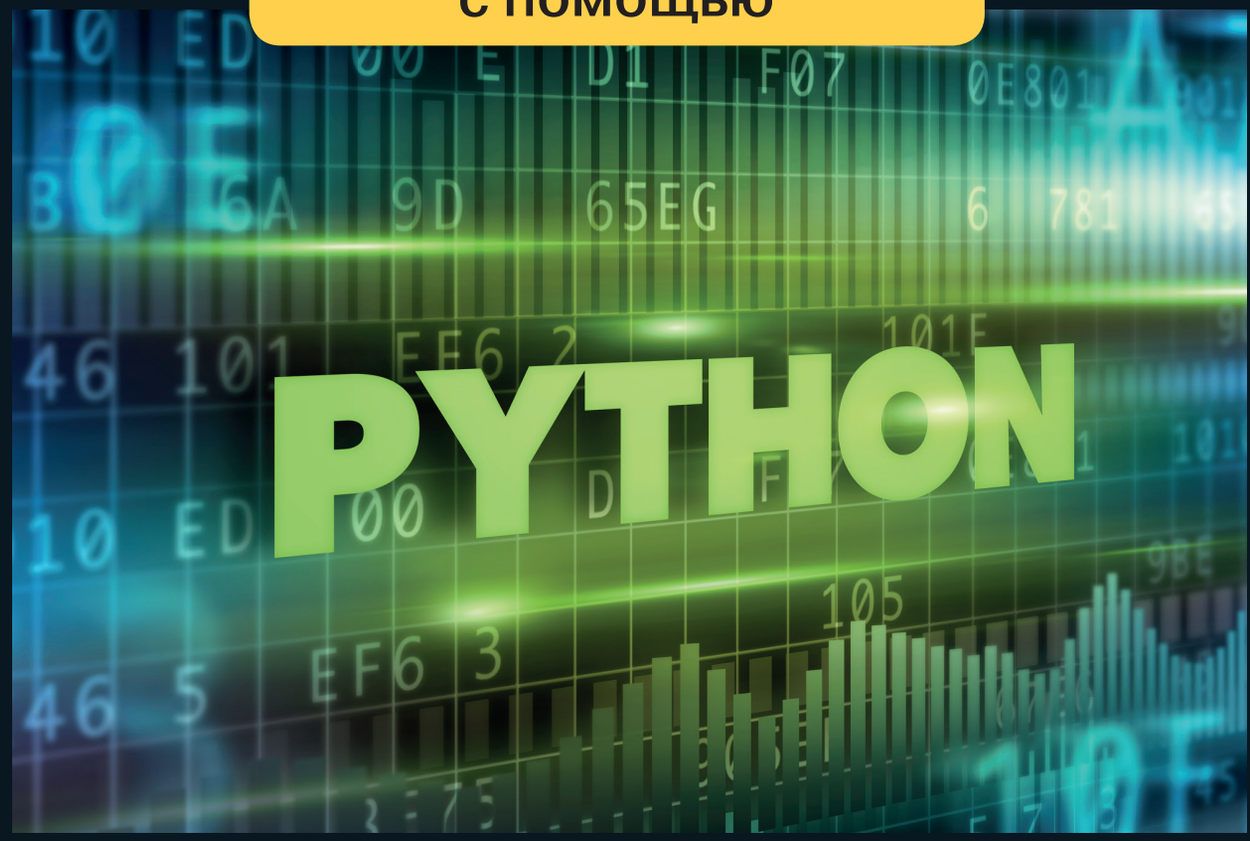
ISBN 978-5-97060-749-7

Изучение робототехники с помощью PYTHON

Джозеф Лентин

Изучение робототехники

С ПОМОЩЬЮ



ДМК ПРЕСС издательство

Лентин Джозеф

Изучение робототехники с использованием Python

Lentin Joseph

Learning Robotics using Python

Second Edition

Design, simulate, program, and prototype an autonomous mobile robot using ROS, OpenCV, PCL, and Python

Packt>
BIRMINGHAM – MUMBAI

Лентин Джозеф

Изучение робототехники с использованием Python

Второе издание

Проектирование, моделирование, программирование и прототипирование интерактивного автономного мобильного робота с нуля с помощью Python, ROS, Open-CV

ДМК Пресс
издательство
Москва, 2019

УДК 004.43, 59.30
ББК 32.81
Д42

Джозеф Л.
Д42 Изучение робототехники с использованием Python / пер. с анг. А. В. Корягина. – М.: ДМК Пресс, 2019. – 250 с.: ил.

ISBN 978-5-97060-749-7

В данной книге рассказывается, как с нуля построить автономный мобильный обслуживающий робот, с помощью которого можно подавать еду в квартире, гостинице и ресторане. Благодаря подробным пошаговым инструкциям читатель узнает весь процесс разработки робота - начиная с теоретической части (принципы дифференциального привода, кинематики и обратной кинематики) и заканчивая практической реализацией (сборка отдельных компонентов, согласование приводов и датчиков с контроллерами). Много внимания уделено программной части - использованию метаоперационной системы ROS, моделированию в Gazebo, обработке изображений в OpenCV, разработке GUI робота на Qt и Python.

Издание предназначено для широкого круга читателей, увлеченных робототехникой, программированием и самостоятельной сборкой различных DIY-устройств.

УДК 004.43, 59.30
ББК 32.81

ISBN 978-1-78862-331-5 (анг.)
ISBN 978-5-97060-749-7 (рус.)

Посвящается моей матери,
Jancy Joseph и моему отцу C. G. Joseph
которые поддержали меня в создании этой книги

Содержание

Составители

Об авторе

Лентин Джозеф – автор этой книги, предприниматель-робототехник из Индии. Он руководит компанией Qbotics Labs. Эта организация находится в Индии, разрабатывает программное обеспечение для роботов и имеет 7-летний опыт работы в сфере робототехники, прежде всего в ROS, OpenCV и PCL.

Лентин Джозеф является автором четырех книг по ROS: «Изучение робототехники с использованием Python», «Освоение ROS для программирования робота», «Проекты робототехники ROS» и «Операционная система робота (ROS) для абсолютных новичков».

В настоящее время автор проводит магистерскую программу по робототехнике в Индии, а также в институте робототехники, CMU, США.

О рецензенте

Ruixiang Du – кандидат технических наук Политехнического университета Вустера. Работает в лаборатории исследования автономии, контроля и оценки и специализируется на планировании движения и управления автономными мобильными роботами в динамичной среде. В 2011 г. получил степень бакалавра автоматизации в Северном Китайском электроэнергетическом университете, а в 2013 г. – степень магистра в области робототехники WPI. Занимался проектами с роботизированными платформами, начиная от медицинских роботов и беспилотных воздушных/наземных транспортных средств и до человекоподобных роботов.

Введение

Книга «Изучение робототехники с использованием Python» состоит из девяти глав, в которых рассказывается, как создать с нуля автономный мобильный робот и, используя язык программирования Python, запрограммировать его. Это устройство разрабатывалось как обслуживающий робот, с помощью которого можно подавать еду в квартире, гостинице и ресторане. В книге представлены подробные пошаговые инструкции, выполняя которые, вы этого робота построите. Сначала рассматриваются основы робототехники. Затем будет создана 3D-модель. После этого будут рассмотрены аппаратные компоненты, необходимые для создания прототипа устройства.

В основном программная часть реализована на языке программирования Python, метаоперационной системе Robot Operating System (ROS) и библиотеке алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения OpenCV. Показано использование Python как для проектирования робота, так и для создания пользовательского интерфейса. Симулятор Gazebo используется для моделирования робота, работы с библиотекой машинного зрения OpenCV, OpenNI и PCL и обработки 2D- и 3D-изображений. Каждая глава начинается с теории, позволяющей понять прикладную часть. Книга анализировалась специалистами в области робототехники.

Для кого предназначена эта книга

Книга «Изучение робототехники с использованием Python» предназначена для инженеров-робототехников, желающих углубить свои знания в этой области и усовершенствовать созданные ранее роботы, предпринимателей, использующих сервисных роботов в своем деле, студентов, изучающих робототехнику, и людей, увлеченных созданием роботов. Книга содержит легковыполнимые пошаговые инструкции.

Краткое содержание

Глава 1 «Начало работы с операционной системой для робота (ROS)»: объясняются основные понятия ROS, являющейся основной платформой для программирования роботов.

Глава 2 «Основные понятия роботов с дифференциальным приводом»: рассматриваются основные принципы роботов с дифференциальным приводом. Обсуждаются фундаментальные концепции дифференциального привода мобильного робота, понятия кинематики и обратной кинематики дифференци-

ального привода. Знания этих принципов помогут вам правильно настроить программное обеспечение регулятора дифференциальной передачи.

Глава 3 «Моделирование робота с дифференциальным приводом»: рассчитываем конструкцию робота и создаем чертежи мобильного робота в 2D и 3D. Создание 2D- и 3D-чертежей является одним из условий разработки мобильного робота. После завершения проектирования и моделирования робота читатель получит расчетные параметры и чертежи, которые будут использованы для создания модели робота.

Глава 4 «Моделирование дифференциального привода робота, управляемого операционной системой ROS»: знакомимся со средой моделирования робота Gazebo и помогаем читателям с помощью Gazebo создать модель собственного робота.

Глава 5 «Проектирование оборудования и схем ChefBot»: выбираем аппаратные компоненты, необходимые для сборки ChefBot.

Глава 6 «Согласование приводов и датчиков с контроллером робота»: рассматриваем взаимодействие регулятора, привода и датчиков робота. Обсуждаем, как взаимодействуют приводы и датчики робота с регулятором Launchpad Tiva C.

Глава 7 «Согласование датчиков зрения с ROS»: обсуждается, как согласовать датчики зрения Kinect и Orbecc Astra, которые могут быть использованы в Chefbot для автономной навигации с ROS.

Глава 8 «Сборка робота ChefBot и интеграция программного обеспечения»: рассказывается, как установить электронное оборудование мобильного робота и настроить его программное обеспечение.

Глава 9 «Разработка графического интерфейса для робота с использованием Qt и Python»: обсуждается, как с помощью GUI передавать команды роботу для его перемещения к столам в гостинице.

Получаем максимальную отдачу от этой книги

Книга содержит подробную информацию, позволяющую создать робот. Для этого вам понадобятся электронные компоненты и материалы (пластик, металл, фанера). Робот может быть построен с нуля, или вы можете купить готовое устройство, оснащенное дифференциальным приводом и энкодером. Для управления роботом вам понадобится плата контроллера, например Texas instruments LaunchPad. Для разработки, моделирования и программирования робота необходим ноутбук/нетбук. В этой книге в качестве бортового компьютера мы используем высокопроизводительный мини-компьютер Intel NUC. Это очень эффективный высокопроизводительный ПК размером 10×10 см. Для компьютерного 3D-зрения используется 3D-датчик, например лазерный сканер Kinect или Orbbec Astra.

Что касается программного обеспечения, вам понадобится знание команд GNU/Linux и Python. Для работы с примерами понадобится операционная си-

стема Ubuntu 16.04 LTS. Быстрее создать робот вам поможет хорошее знание ROS, OpenCV, OpenNI и PCL. Кроме того, вам потребуется установить Ros Kinect Melodic с примерами.

Загрузка файлов с примерами программного кода

Вы можете скачать файлы с примерами программного кода по адресу www.packtpub.com со своего аккаунта. Если вы приобрели эту книгу в другом месте, зарегистрируйтесь на сайте www.packtpub.com/support. Файлы вам будут отправлены по электронной почте.

Чтобы загрузить файлы кода, выполните следующие действия:

1) если вы уже зарегистрированы, зайдите под своим аккаунтом на сайт www.packtpub.com. Если у вас нет учетной записи, сначала зарегистрируйтесь;
2) выберите вкладку **SUPPORT**;
3) щелкните мышью на ссылке на Code Downloads & Errata;
4) введите название книги в поле поиска и следуйте инструкциям на экране.

После того как файл будет загружен, пожалуйста, убедитесь, что вы сможете его распаковать. Это можно сделать с помощью программ:

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

Пакет кодов для этой книги также размещен на GitHub по адресу: https://github.com/PacktPublishing/Learning-Robotics-using-Python-Second-Edition.

Другие программные пакеты и видео для нашего богатого каталога книг вы найдете по адресу: https://github.com/PacktPublishing/.

Загрузка цветных изображений

Вы также можете загрузить PDF-файл с цветными изображениями скриншотов/диаграмм, используемых в этой книге. Данный файл находится по адресу: https://www.packtpub.com/sites/default/files/downloads/LearningRoboticsusingPythonSecondEdition_ColorImages.pdf.

Условные обозначения, принятые в этой книге

В этой книге используется ряд типографических условных обозначений.

`Моноширинный текст`: указывает кодовые слова в тексте, имена таблиц базы данных, названия папок, имена файлов, расширения файлов, имена путей сохранения файлов, фиктивные URL-адреса, пользовательский ввод и маркеры Twitter. Например: «первая процедура – создать файл и сохранить его с расширением `.world`».

Пример блока кода:

```
<xacro:include filename="$(find
  chefbot_description)/urdf/chefbot_gazebo.urdf.xacro"/>
  <xacro:include filename="$(find
  chefbot_description)/urdf/chefbot_properties.urdf.xacro"/>
```

Любой ввод или вывод из командной строки записывается следующим образом:

```
$ roslaunch chefbot_gazebo chefbot_empty_world.launch
```


Так будут оформляться предупреждения и важные примечания.


Так будут оформляться советы или рекомендации.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры, для того чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с неза-

конно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли принять меры.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1

Начало работы с операционной системой для робота (ROS)

Основная цель этой книги – помочь вам создать автономный мобильный робот. Робот оснащается операционной системой ROS, а все операции будут смоделированы с помощью программы-симулятора, называющейся Gazebo. Также в этой главе вы найдете теоретический расчет, проект робота, чертежи деталей, внутреннее программное обеспечение и программное обеспечение высокого уровня, интегрированное с ROS.

Глава начинается с основных понятий о ROS, таких как установка, написание основных программ с использованием ROS и Python. Далее идет знакомство с основами работы в Gazebo. Эта глава станет базовой в вашем проекте автономного робота. Если вы уже знакомы с основами ROS и эта операционная система у вас установлена, то эту главу можете пропустить. Однако вы всегда можете вернуться, чтобы освежить память об основах ROS.

В этой главе будут рассмотрены следующие темы:

- введение в ROS;
- установка Ros Kinetic на Ubuntu 16.04.3;
- знакомство, установка и тестирование Gazebo.

Начинаем программировать робота с помощью Python и операционной системы робота (ROS).

Технические условия

Для получения полного кода, описанного в этой главе, перейдите по следующей ссылке: https://github.com/qboticslabs/learning_robotics_2nd_ed.

Введение в ROS

ROS – это программная платформа, назначение которой – написание программного обеспечения робота. Основная цель ROS – создание и использование робототехнического программного обеспечения по всему миру. ROS состоит из набора инструментов, библиотек и программных пакетов, упрощающих задачу создания программного обеспечения робота.

Официальное определение ROS

ROS – это метаоперационная система с открытым исходным кодом, разработанная для робота. Она предоставляет все необходимые для данной операционной системы службы, включая аппаратные средства визуализации, низкоуровневое управление устройством, реализацию функциональности для общего использования, передачу сообщений между процессами и пакетами управления. Она также предоставляет инструменты и библиотеки для получения, построения, написания и выполнения кода на нескольких компьютерах. ROS похожа в некоторых отношениях на «фреймворки робота», такие как Player, YARP, Orocos, CARMEN, Orca, MOOS и Microsoft Robotics Studio.

Более подробная информация о ROS здесь: http://wiki.ros.org/ROS/Introduction.

Основные функции, которые обеспечивает ROS:

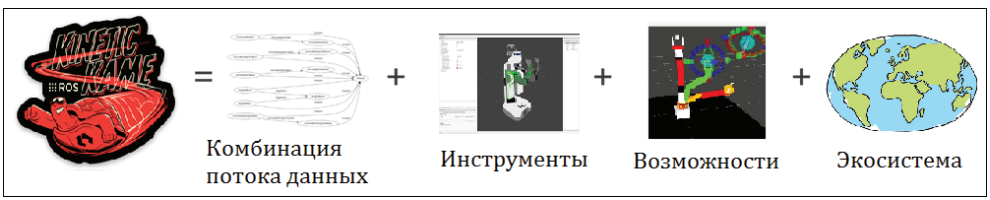
- **интерфейс передачи сообщений**. Это базовая функция ROS, позволяющая организовать межпроцессное взаимодействие. Используя эту функцию, ROS может обмениваться данными между связанными системами. Больше о технических терминах, связанных с обменом данными между ROS-программами/узлами, вы узнаете в этой главе;
- **аппаратная универсальность**. ROS обладает некоторой степенью универсальности, что позволяет разработчикам создавать роботизированные приложения. Они могут быть использованы с любым устройством. Разработчики должны лишь позаботиться о надлежащем аппаратном обеспечении робота;
- **управление пакетами**. Узлы ROS организованы в пакеты и называются пакетами ROS. Они состоят из исходных кодов, конфигурационных файлов типа «build» («строить») и т. д. Мы сначала разрабатываем пакет, собираем его и устанавливаем. В ROS предусмотрена система сборки, которая помогает создавать эти пакеты. Управление пакетами делает развитие ROS более упорядоченным и организованным;
- **дополнительные библиотеки**. В структуру ROS встроено несколько дополнительных библиотек, разработанных сторонними организациями, таких как Open-CV, PCL, OpenNI и др. Данные библиотеки помогают сделать ROS универсальной метаоперационной системой;
- **управление низкоуровневыми устройствами**. Во всех роботах присутствуют устройства низкого уровня, например устройства ввода/вы-

вода, контроллеры последовательных портов и др. Управление такими устройствами тоже осуществляется с помощью ROS;

- **распределенные вычисления**. Объем вычислений, которые потребуются для обработки потока данных от многочисленных датчиков, очень высок. Используя ROS, мы можем легко распределить их в группе вычислительных узлов. Это позволит равномерно распределить вычислительные мощности, и данные будут обрабатываться быстрее, чем на одиночном компьютере;
- **многократное использование кода**. Основной целью ROS является многократное использование кода, способствующее развитию научно-исследовательского сообщества по всему миру. Исполняемые файлы ROS называются узлами. Они сгруппированы в единый объект, называемый пакетом ROS. Группа пакетов – стек. Стеки могут быть разделены и распределены;
- **независимость от языка**. В ROS можно программировать с помощью таких популярных языков, как Python, C++ и Lisp. Узлы могут быть написаны на любом из этих языков, и обмен данными между такими узлами через ROS не вызовет никаких проблем;
- **простота тестирования**. ROS имеет встроенный модуль интеграционного тестирования, называющийся rostest и предназначенный для тестирования пакетов ROS;
- **масштабирование**. ROS после некоторой модификации может выполнять сложные вычисления в роботах;
- **свободный и открытый исходный код**. Исходный код ROS открыт и абсолютно свободен в использовании. Основная часть ROS лицензирована BSD и может быть повторно использована в коммерческих и закрытых продуктах источника.

ROS – комбинация из потока данных (передачи сообщений), инструментов, возможностей и экосистемы. В ROS встроены мощные инструменты для ее отладки и визуализации. Робот, оснащенный ROS, обладает такими возможностями, как навигация, определение местонахождения, составление карты, манипуляции и т. д. Они помогают создать мощные приложения для роботов.

Приведенное ниже изображение показывает уравнение ROS:



Уравнение ROS

Более подробная информация о ROS здесь: http://wiki.ros.org/ROS/Introduction.

Концепции ROS

ROS состоит из трех основных уровней, таких как:

1) файловая система ROS;
2) вычислительный граф ROS;
3) сообщество ROS.

Файловая система ROS

Основная задача файловой системы ROS – организация файлов на диске.

Основные термины файловой системы ROS:

- **пакеты**. Пакеты ROS являются основной единицей в рамках программного фреймворка ROS. Пакет ROS может содержать исполняемые файлы, ROS-библиотеки, принадлежащие программному обеспечению сторонних производителей, конфигурационные файлы и т. д. Пакеты ROS можно использовать повторно и совместно;
- **описания пакета**. В описании пакетов (пакет .XML-файла) вы найдете всю детализацию пакетов, включая имя, описание, лицензию и зависимости;
- **типы сообщений** (msg). Все сообщения хранятся в отдельной папке, в файлах с расширением `.msg`. ROS-сообщения – это структурированные данные, проходящие через ROS;
- **типы служб** (SRV). Описания служб хранятся в папке SRV с расширением `.srv`. SRV-файлы определяют запрос и ответ для структуры данных в сервисе ROS.

Вычислительный граф ROS

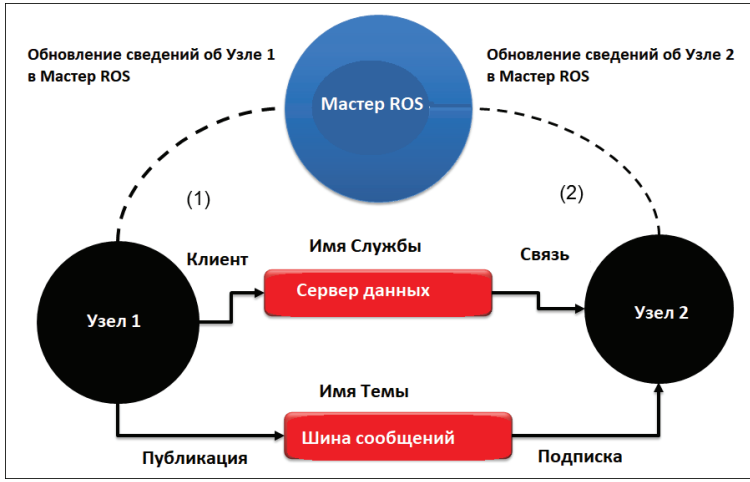
Вычислительный граф ROS – это одноранговая сеть систем ROS, назначение которой – обработка всех данных. Основными понятиями вычислительной графики ROS являются узлы, Мастер ROS, сервер параметров, сообщения и службы.

- **Узлы** – это вычислительные процессы в ROS. Каждый узел выполняет обработку определенных данных. Например, один узел обрабатывает данные с лазерного сканера, другой публикует эти данные и т. д. Узел ROS создается с помощью клиентской библиотеки ROS (например, roscpp и rospy). Более подробно мы познакомимся с этими библиотеками во время создания образца узла.
- **Мастер ROS.** Мастер ROS действует как служба имен в графе вычислений ROS. Он хранит темы и регистрационную информацию служб для узлов ROS. Узлы общаются с Мастером, сообщая свои регистрационные данные. Эти узлы, общаясь с мастером, получают информацию о других зарегистрированных узлах и соответственно устанавливают с ними соединения. Мастер также выполняет обратный вызов этих узлов при изменении регистрационной информации, позволяющей узлам динамически создать соединения при запуске новых узлов. Узлы подключа-

ются непосредственно к другим узлам. Мастер, как и DNS-сервер, обеспечивает только поиск информации. Узлы, подписывающиеся на тему, запрашивают соединения от узлов, публикующих эту тему, и позволяют установить связи по согласованному протоколу подключения. Наиболее распространенный протокол, используемый в ROS, называется TCPROS. Он использует стандартный протокол TCP/IP-сокетов.

- **Сервер параметров**. Статические данные ROS хранятся в общедоступном месте – на сервере данных, от которого узлы и получают доступ к этим данным. Мы можем установить объем сервера данных и определить его как частный или общественный, чтобы ограничить доступ к серверу одним узлом или разрешить доступ всем узлам.
- **Темы ROS**. Узлы обмениваются данными в виде сообщений через систему транспортировки в ROS с конкретным названием темы. Тема – это имя, используемое для идентификации содержания сообщения. Узел, заинтересованный в определенном виде данных, будет подписываться на соответствующую тему. В целом издатель и абонент не знают о существовании друг друга. Идея в том, чтобы отделить производство информации от ее потребления. Логически можно думать о теме как о шине строго типизированных сообщений. Каждая шина имеет имя, и любой может подключаться к ней для отправки или получения сообщений до тех пор, пока они имеют правильный тип.
- **Сообщения**. Узлы взаимодействуют друг с другом с помощью сообщений. Сообщение ROS состоит из примитивных типов данных, таких как целые числа, плавающие точки, логические значения («true» – «истина» или «false» – «ложь»). Сообщения ROS передаются через тему ROS. Тема может получать или посылать один тип сообщения только один раз. Мы можем создать собственное определение сообщения и передать его через тему.
- **Службы.** Выше было показано, что переслать/получить сообщение с использованием тем ROS – это очень простой метод общения между узлами. Этот способ коммуникации («один ко многим» – «one-to-many») позволяет подписываться на одну тему любому количеству узлов. В некоторых случаях может осуществляться частичное взаимодействие, обычно применяемое в распределенных системах. Используя определенное сообщение, можно послать запрос в другой узел, предоставляющий нужную услугу. Узел, пославший запрос другому узлу, обязан ожидать результат обработки данного запроса.
- **Bags** – это формат для сохранения и воспроизведения данных сообщений ROS. Bags – важный механизм для хранения данных, получаемых, например, от датчиков. Эти данные могут быть использованы позже для разработки и тестирования алгоритмов в автономном режиме.

На следующем рисунке показано взаимодействие между мастером, темами, службами и узлами:



Связь между узлами ROS и мастером ROS

На приведенной выше схеме показана связь, осуществляемая с помощью Мастера ROS (ROS Master) и двух узлов ROS: Узла 1 (Node 1) и Узла 2 (Node 2). Прежде чем начать обмен данными между этими двумя узлами, необходимо запустить Мастер ROS. Master ROS является посредником, позволяющим установить связь и обмениваться информацией между узлами. Например, Узел 1 (Node 1) хочет опубликовать тему/xyz с типом сообщения abc. Для этого он обращается к ROS Master и сообщает, что собирается опубликовать тему/xyz с сообщением вида abc и поделиться ею с другими узлами. А другой узел, например Узел 2 (Node 2), желает подписаться на эту тему/xyz с типом сообщения abc. Мастер сообщает Узлу 2, какой узел опубликовал запрашиваемую информацию, и выделяет порт для связи этих двух узлов напрямую, минуя Мастер ROS.

Таким образом, как уже упоминалось ранее, Master ROS представляет собой DNS-сервер, обеспечивающий поиск запрашиваемой узлами информации. Ранее уже упоминалось, что в ROS применяется протокол связи TCPROS (http://wiki.ros.org/ROS/TCPROS), использующий для связи в основном сокеты TCP/IP.

Уровень общения ROS

Сообщество ROS состоит из разработчиков и исследователей, создающих и поддерживающих пакеты ROS. Они обмениваются между собой информацией о существующих и недавно созданных пакетах и другими новостями, связанными со структурой ROS. Сообщество ROS предоставляет следующие услуги:

- **распространение**. В распространяемых дистрибутивах присутствует ряд пакетов, предназначенных для конкретных версий ROS. В этой книге используется дистрибутив ROS Kinetic. Доступны и другие версии дистрибутивов, например такие, как ROS Lunar или Indigo. Зная версию дис-

трибутива, легче подобрать к нему недостающие пакеты, которые будут стабильно работать;

- **репозитории**. Репозитории, или мета-пакеты ROS, находятся на удаленных серверах. Также в едином репозитории может храниться отдельный пакет;
- **Wiki-ROS**. На этом ресурсе доступна почти вся документация о ROS. Здесь вы можете получить информацию от фундаментальных понятиях до самого передового программирования. Любой желающий может зарегистрироваться и предоставить собственную документацию, исправления или обновления, написать учебники и т. д. Адрес ресурса: http://Wiki.ROS.org);
- **списки рассылки**. Для получения обновлений следует подписаться на рассылку ROS (http://lists.ros.org/mailman/listinfo/ros-users). Последние новости о ROS и форум, где обсуждается программное обеспечение ROS, находятся по адресу: https://discourse.ros.org;
- **ответы ROS**. Если у вас возникнут вопросы, связанные с ROS, обращайтесь по адресу: https://answers.ros.org/questions/. Здесь вы получите помощь и поддержку от разработчиков со всего мира.

Существует много функциональных возможностей ROS. Для получения дополнительных сведений о них обращайтесь на официальный сайт ROS по адресу: www.ros.org.

В следующем разделе мы рассмотрим процедуру установки ROS.

Установка ROS на Ubuntu

Как было сказано ранее, ROS – это метаоперационная система, которая устанавливается в ведущей операционной системе (host-системе). ROS полностью поддерживается такой операционной системой, как Linux Ubuntu. Проводились эксперименты с операционными системами и Windows, и OS X. Вот некоторые из последних ROS-дистрибутивов:

Дистрибутив	Дата выпуска
ROS Melodic Morenia	23 мая 2018
ROS Lunar Loggerhead	23 мая 2017
ROS Kinetic Kame	23 мая 2016
ROS Indigo Igloo	22 июля 2014

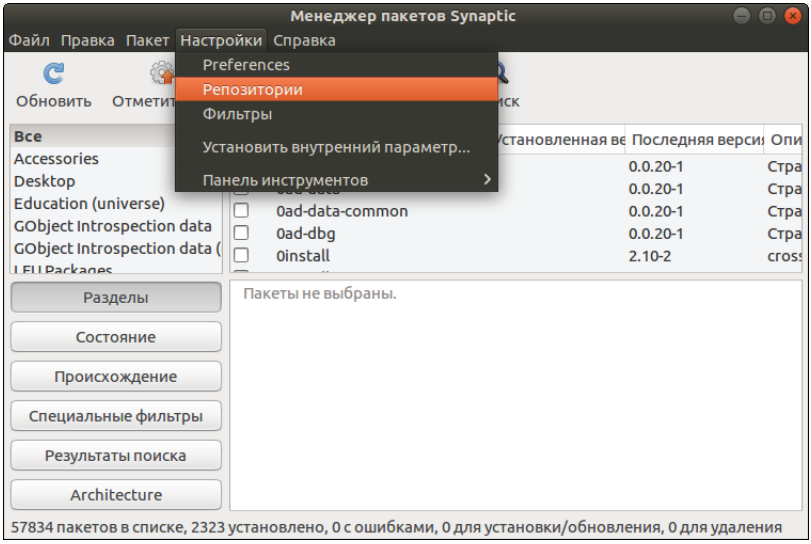
Теперь рассмотрим процедуру установки стабильной версии дистрибутива ROS Kinetic с долгосрочной поддержкой (LTS) на Ubuntu 16.04.3 LTS. Разработчики ROS Kinetic Kame в первую очередь ориентировались на Ubuntu 16.04 LTS. Вы также можете найти инструкции по настройке ROS LTS Melodic Morenia на Ubuntu 18.04 LTS. Если на вашем компьютере операционной системой является Windows или OS X, то сначала нужно установить виртуальную машину Virtual Box, а затем Linux Ubuntu 16.04 LTS.

Загрузить установочный файл виртуальной машины можно, перейдя по ссылке: https://www.virtualbox.org/wiki/Downloads.

Подробные инструкции по установке вы можете найти на сайте http://wiki.ros.org/kinetic/Installation/Ubuntu. Обратите внимание, что ROS, Gazebo, LibraCad, Blender требуют довольно много места на диске. Поэтому при установке Linux Ubuntu 16.04 LTS увеличьте размер виртуального диска как минимум до 20 гигабайт вместо предлагаемых по умолчанию 10 гигабайт.

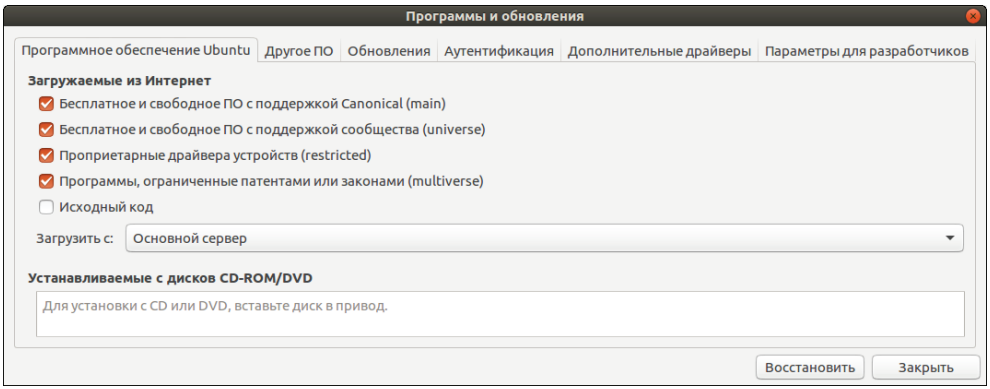
Чтобы установить ROS, выполните следующие действия.

1. В левом верхнем углу экрана выберите команду главного меню **Приложения → Системные утилиты → Менеджер пакетов**. На экране окно приложения, называющегося **Менеджер пакетов**.



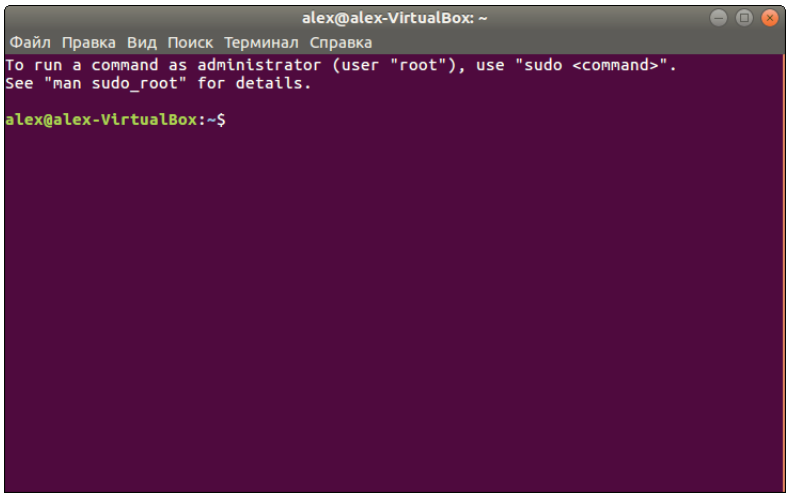
Команда **Настройки → Репозитории**

2. Выберите команду **Настройки → Репозитории**. Откроется окно **Программы и обновления**.



Окно **Программы и обновления**

3. Из открывающегося списка **Загрузить с**: выберите строку **Основной сервер** (Main server).
4. Установите флажки **Проприетарные драйвера устройств (restricted)** и **Бесплатное и свободное ПО с поддержкой сообщества (universe)**.
5. Закройте окно **Программы и обновления**. Теперь вся установка будет проходить с помощью команд, вводимых с клавиатуры.
6. Чтобы запустить приложение **Терминал**, выберите команду главного меню **Приложения** → **Утилиты** → **Терминал**.



Терминал запущен

7. Для авторизации в **Терминале** введите команду `sudo su`, для завершения ввода команды нажмите клавишу **Enter** и введите свой пароль. Те-

перь все остальные команды будут вводиться от имени администратора, и вам не придется начинать каждую команду со слова `sudo`.
Обратите внимание, что в книге в начале каждой команды будет находиться символ `$`. Его вводить не нужно, т. к. этот символ обозначает командную строку.

8. Чтобы при установке ROS не возникло ошибок, сначала желательно обновить список источников. Для этого необходимо открыть список источников `Sources.list`, находящийся в папке `etc/apt/`, и с помощью редактора **Nano** отредактировать источник приложений. Введите команду:

```
$ nano /etc/apt/sources.list
```

9. В конце списка источников добавьте следующие строки:

```
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial main restricted universe multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-security main restricted universe
multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-updates main restricted universe
multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-proposed main restricted universe
multiverse
deb http://mirrors.ustc.edu.cn/ubuntu/ xenial-backports main restricted universe
multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial main restricted universe
multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-security main restricted universe
multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-updates main restricted universe
multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-proposed main restricted universe
multiverse
deb-src http://mirrors.ustc.edu.cn/ubuntu/ xenial-backports main restricted universe
multiverse
```

10. Нажмите комбинацию клавиш **Ctrl**+**O**, а затем клавишу **Enter**. Изменения в списке источников приложений сохранены. Нажмите комбинацию клавиш **Ctrl**+**C**. Окно редактора закроется, и вы снова вернетесь в терминал.
11. Обновите источники. Для этого выполните следующую команду:

```
$ apt-get update
```

Процесс обновления займет некоторое время. После того как обновление завершится, можно приступать к установке ROS Kinetic.

12. Настройте систему для приема пакетов ROS из http://packages.ros.org. ROS Kinetic поддерживается только на Ubuntu 15.10 и 16.04. Следующая команда сохранит http://packages.ros.org в списке `apt` репозиториев Ubuntu:

```
$ sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /
etc/apt/sources.list.d/ros-latest.list'
```

13. Добавьте apt-key. Apt-key нужен для управления списком ключей, которые используются apt для проверки подлинности пакетов. Пакеты, которые с помощью этих ключей пройдут проверку, будут считаться доверенными. Следующая команда добавит apt-keys для пакетов ROS:

```
$ apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9
FF1F717815A3895523BAEEB01FA116
```

14. После добавления apt-ключей требуется обновить список пакетов Ubuntu.
Для совместного обновления пакетов ROS с пакетами Ubuntu выполните следующую команду:

```
$ apt-get update
```

15. После того как пакеты ROS обновятся, мы можем их установить. Приведенная ниже команда установит необходимые инструменты и библиотеки ROS:

```
$ apt-get install ros-kinetic-desktop-full
```

16. Полная установка займет некоторое время. Возможно, после ее завершения потребуется установить дополнительные пакеты. Каждая дополнительная установка будет упомянута в соответствующем разделе.
17. Следующий шаг – инициализация `rosdep`. Это позволит легко установить системные зависимости для исходных пакетов ROS, или скомпилировать некоторые компоненты ROS, необходимые для работы:

```
$ rosdep init
$ rosdep update
```

18. Чтобы получить доступ к инструментам и командам ROS в текущей оболочке bash, добавим переменные среды ROS в файл `bashrc`. Эта операция будет выполняться в начале каждой сессии bash. Ниже приведена команда для добавления переменной ROS `.bashrc`:

```
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
```

Следующая команда выполнит `.bashrc`-скрипт для генерации изменений в текущей оболочке:

```
$ source ~/.bashrc
```

После установки ROS мы обсудим, как создать в ней типовой пакет. Сначала следует создать рабочую область ROS. Мы будем использовать систему сборки catkin, представляющую собой набор инструментов для построения пакетов в ROS. Система catkin генерирует исполняемую или общую библиотеку из исходного кода. Поскольку ROS Kinetic для построения пакетов использует инструменты catkin, познакомимся с ним.
19. Одним из полезных инструментов для установки является rosinstall. Данный инструмент устанавливается отдельно. Rosinstall позволяет

легко скачать многие исходные деревья пакетов ROS с помощью одной команды:

```
$ apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

Инсталляция последней версии ROS – LTS Melodic похожа на инсталляцию ROS Kinetic Kame. Вы можете установить Melodic вместе с Ubuntu 18.04 LTS. Полная инструкция по установке находится по адресу: http://wiki.ros.org/melodic/Installation/Ubuntu.

Введение в catkin

Catkin является официальной версией системы сборки системы ROS. До ее появления использовали систему для сборки пакетов, называющуюся Rosbuild. Ее заменили на catkin в версиях, начинающихся с ROS Indigo. Для обеспечения нормального технологического процесса в Catkin совместно со скриптами Python используется макрос CMake. Catkin обеспечивает лучшие распределение пакетов и кросс-компиляцию. К тому же она более мобильна, чем система Rosbuild. Для получения дополнительной информации обратитесь по адресу: wiki.ros.org/catkin.

Рабочее пространство Catkin – это папка, в которой можно создавать, изменять и устанавливать пакеты catkin.

Давайте посмотрим, как создать рабочее пространство ROS catkin.

Представленная ниже команда создаст родительский каталог `catkin_ws` и вложенную папку с именем `src`:

```
$ mkdir -p ~/catkin_ws/src
```

Переместите созданный родительский каталог в папку `src` с помощью приведенной ниже команды. Мы создадим наши пакеты в папке `src`:

```
$ cd ~/catkin_ws/src
```

Используя следующую команду, инициализируйте рабочую область catkin:

```
$ catkin_init_workspace
```

После инициализации рабочей области catkin у нас появится возможность собрать пакет (даже при отсутствии исходного файла). Для этого используем приведенную ниже команду:

```
$ cd ~/catkin_ws/
$ catkin_make
```

Команда `catkin_make` используется для построения пакетов внутри каталога `src`.

После сборки пакетов в каталоге `catkin_ws` мы увидим папки `build` и `devel`. В папке `Build` будут сохранены исполняемые файлы, а в папке `devel` появятся сценарии оболочки для добавления в рабочую область среды ROS.

Создание пакета ROS

В этом разделе мы расскажем, как создать образец пакета, содержащего два узла Python. Один из узлов опубликует в тему сообщение Hello World, а другой узел на эту тему подпишется.

Пакет catkin ROS создается в ROS с помощью команды `catkin_create_pkg`. Данный пакет появится внутри ранее созданной папки `src`. Перед созданием пакетов перейдите в папку `src` с помощью следующей команды:

```
$ cd ~/catkin_ws/src
```

Приведенная ниже команда создаст пакет `hello_world` с зависимостью `std_msgs`, содержащий стандартные определения сообщений. Rospy является клиентской библиотекой Python для ROS:

```
$ catkin_create_pkg hello_world std_msgs rospy
```

После успешного создания пакета мы получим следующее сообщение:

```
   Created file hello_world/package.xml
   Created file hello_world/CMakeLists.txt
   Created folder hello_world/src
   Successfully created files in /home/имя_пользователя/catkin_ws/src/hello_world.
Please adjust the values in package.xml
```

После того как пакет `hello_world` будет успешно создан, следует добавить два узла со скриптами Python для демонстрации подписки и публикации разделов. Для этого нужно в пакете `hello_world` создать папку для хранения двух текстовых файлов-сценариев и сами пустые файлы `hello_world_publisher.py` и `hello_world_subscriber.py`. Далее, не выходя из Терминала, с помощью любого текстового редактора, например Nano, отредактируем оба файла сценария, введя соответствующие коды. Затем мы сможем изменить права доступа и запустить сценарии на исполнение.

Итак, создадим папку `scripts`:

```
$ mkdir scripts
```

Перейдем во вновь созданную папку:

```
$ cd ~/catkin_ws/src/scripts
```

Создадим два пустых файла и назовем их `hello_world_publisher.py` и `hello_world_subscriber.py`.

```
$ touch hello_world_publisher.py
```

```
$ touch hello_world_subscriber.py
```

С помощью команды `Ls` проверим, были ли созданы в папке `scripts` два файла:

```
$ Ls
```

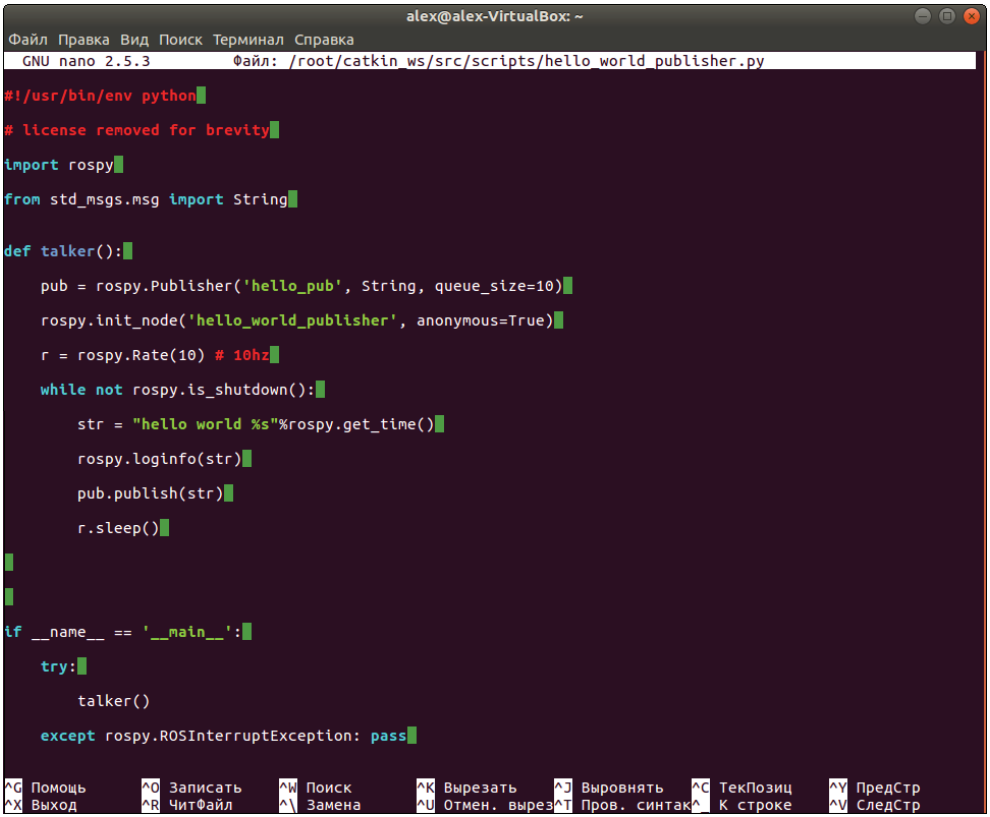
В ответ мы получим список файлов, хранящихся в данной папке:

```
hello_world_publisher.py hello_world_subscriber.py
```

Итак, файлы созданы. Теперь для каждого файла поочередно запустим редактор nano и внесем строки кода. Запускаем nano для файла `hello_world_publisher.py`:

```
$ nano ~/catkin_ws/src/scripts/hello_world_publisher.py
```

Далее нужно ввести скрипт с клавиатуры. На рисунке ниже показан скрипт сценария `hello_world_publisher.py`. При вводе строк скрипта особое внимание нужно обратить на отступы:

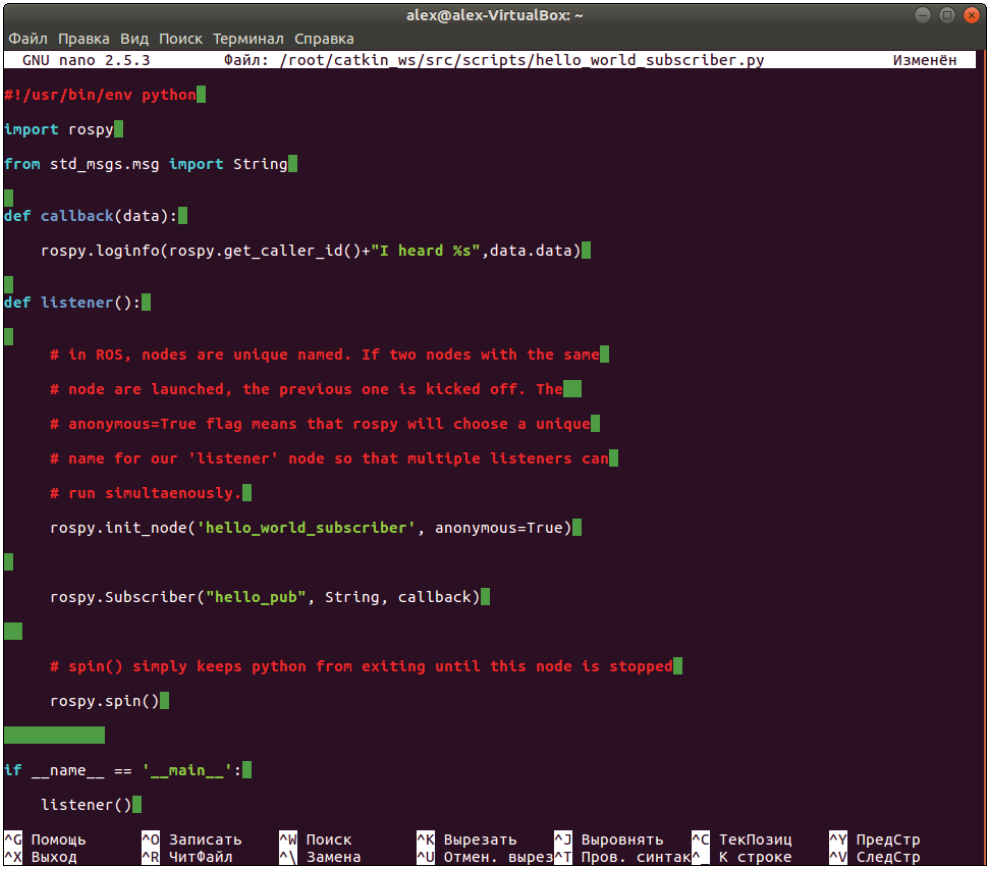


Скрипт `hello_world_publisher.py`

Чтобы сохранить скрипт и закрыть окно редактора, нажмите комбинацию клавиш **Ctrl**+**O**, клавишу **Enter** и комбинацию клавиш **Ctrl**+**X**.

Отредактируйте таким же образом файл `hello_world_subscriber.py`.

```
$ nano ~/catkin_ws/src/scripts/hello_world_subscriber.py
```



Скрипт `hello_world_subscriber.py`

После того как текст скриптов будет внесен в оба файла и редактор nano закроется, следует изменить права доступа к файлам (опубликовать) и запустить скрипты на исполнение.

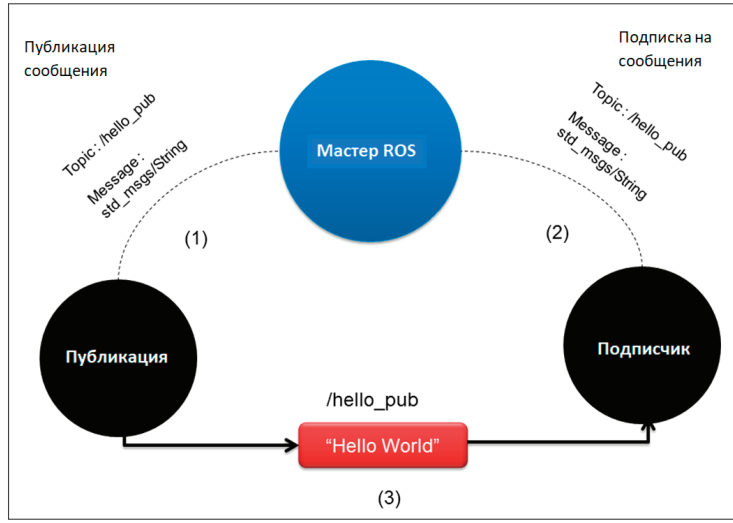
Следующие два раздела дают полное описание скриптов с именами `hello_world_publisher.py` и `hello_world_subscriber.py`.

Далее с помощью введенных в Терминал команд `chmod` будут изменены права доступа к исполняемому файлу. Благодаря команде `catkin_make` мы создадим пакет, после чего запустим скрипты на исполнение.

Hello_world_publisher.py

Узел `hello_world_publisher.py` публикует приветствие, вызывая сообщение `hello world` в теме `/hello_pub`. Приветствие в теме публикуется с частотой 10 Гц.

Вот схема, показывающая взаимодействие между двумя узлами ROS:



Связь между узлом издателя и подписчика

Полный код этой книги доступен по адресу: https://github.com/qboticslabs/learning_robotics_2nd_ed.

Коды двух узлов показаны на снимках с экрана, приведенных выше. Пошаговое описание этого кода выглядит следующим образом.

1. Для написания узла ROS на Python нам требуется импортировать `rospy`. Его назначение – организовать взаимодействие ROS с темами, сервисами и т. д.
2. Чтобы отправить сообщение **hello world**, из пакета `std_msgs` мы импортируем строковый тип данных. Он определяет сообщения для стандартных типов данных. Импорт производится с помощью следующей команды:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
```

3. Следующая строка кода отвечает за создание объекта издателя в разделе с именем `hello_pub`. Тип данных – `String` (строка), `queue_size` (размер очередности) – `10`. Если абонент медленно получает данные, используйте параметр `queue_size` для его настройки:

```
def talker():
    pub = rospy.Publisher('hello_pub', String, queue_size=10)
```

4. Следующая строка кода является обязательной для всех узлов ROS Python. Ее назначение – инициализация узла и назначение ему имени. Пока узел не получит имя, он не может быть запущен. Для взаимодействия с другими узлами он должен использовать свое имя. Если двум узлам будет присвоено одинаковое имя, возникнет конфликт имен и узлы прекратят свою работу. Для запуска обоих узлов следует использовать признак `anonymous=True`, как показано ниже:

   ```
   rospy.init_node('hello_world_publisher', anonymous=True)
   ```

5. Следующая строка создает объект с названием `r`. Используя в объекте `Rate()` метод `sleep()`, мы можем выбрать желаемую частоту цикла. В показанной ниже строке выберем обновление цикла с частотой 10 Гц:

   ```
   r = rospy.Rate(10) # 10hz
   ```

6. Следующий цикл проверяет, создает ли `rospy` признак завершения цикла `rospy.is_shutdown()`. Если этот признак отсутствует, цикл повторяется. Чтобы завершить цикл вручную, достаточно нажать комбинацию клавиш **Ctrl**+**C**. Сообщение `hello_world` внутри цикла публикуется по теме `hello_pub` с частотой 10 Гц:

   ```
   while not rospy.is_shutdown():
       str = "hello world %s"%rospy.get_time()
       rospy.loginfo(str)
       pub.publish(str)
       r.sleep()
   ```

7. Следующий код Python `_main_` наблюдает за выполнением кода и появлением признака `rospy.ROSInterruptException`, вызываемого методом `rospy.sleep()` и `rospy.Rate.sleep()`. Узел продолжает работать до появления данного признака или до нажатия комбинации клавиш **Ctrl**+**C**.

   ```
   If __name___ == '__main__':
       try:
           talker()
       except rospy.ROSInterruptException: pass
   ```

Итак, тема опубликована. Далее будет показано, как на эту тему подписаться. В следующем разделе мы рассмотрим код для подписки на тему `hello_pub`.

Hello_world_subscriber.py

Код подписчика выглядит следующим образом:

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String
```

Данный код – это функция обратного вызова, которая выполняется при по-

явлении сообщения hello_pub тема. Переменная данных содержит сообщение от темы и создаст сообщение rospy.loginfo():

```
def callback(data):
    rospy.loginfo(rospy.get_caller_id()+"I heard %s",data.data)
```

Следующим действием будет запуск узла с именем hello_world_subscriber и запуск подписки на тему /hello_pub.

1. Типом данных сообщения будет String (строка). Даже по прибытии сообщения в тему тип данных не изменится. Этот метод называется методом обратного вызова и запускается следующим образом:

   ```
   def listener():
      rospy.init_node('hello_world_subscriber',anonymous=True)
      rospy.Subscriber("hello_pub", String, callback)
   ```

2. Данный код не позволит узлу выйти до завершения работы самого узла:

   ```
   rospy.spin()
   ```

3. Следующий очень важный шаг – это проверка кода на языке Python. Основной раздел вызывает метод listener(), который, в свою очередь, создаст подписку на тему /hello_pub:

   ```
   if __name__ == '__main__':
       listener()
   ```

   Итак, скрипты двух узлов написаны и сохранены. Теперь, как говорилось ранее, нужно изменить права доступа к исполняемым файлам.

4. Это делается с помощью команды chmod:

   ```
   $ chmod +x hello_world_publisher.py
   $ chmod +x hello_world_subscriber.py
   ```

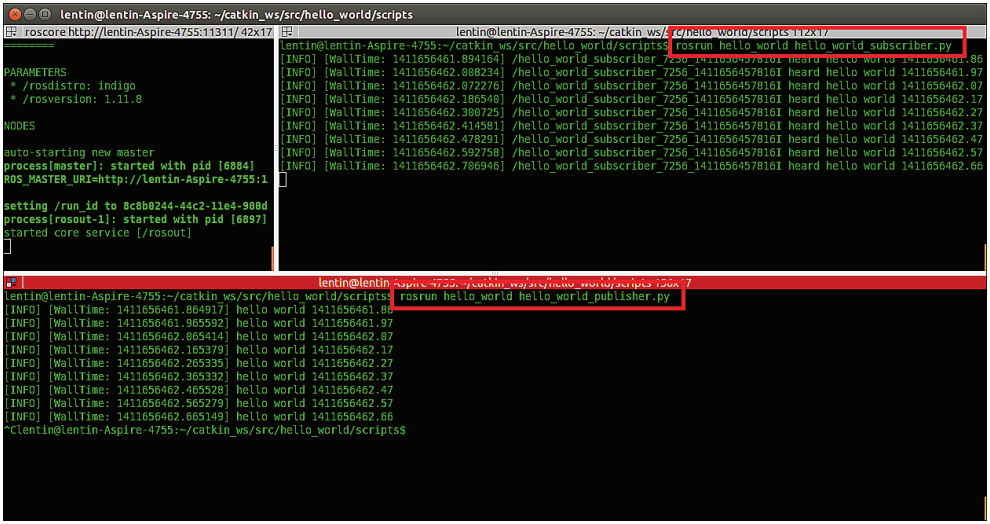
5. После того как права доступа будут изменены, необходимо с помощью команды catkin_make создать пакет:

   ```
   cd ~/catkin_ws/
   catkin_make
   ```

6. Следующая команда добавит текущий путь в рабочей области ROS во всех конечных устройствах так, чтобы у нас появился доступ ко всем пакетам ROS внутри этой рабочей области:

   ```
   echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
   source ~/.bashrc
   ```

Ниже приведены выходные данные узлов подписчика и издателя:



Выходные данные узла hello world

Далее можно запустить созданные узлы.

1. Перед запуском узла сначала следует запустить `roscore`. Команда `roscore`, или Мастер ROS, необходима для установки связи между узлами.
   Для запуска `roscore` выполните команду:

   ```
   $ roscore
   ```

2. Далее запустите каждый из двух узлов. Для этого используйте две приведенные ниже команды:
   - для запуска публикации выполните следующую команду:

     ```
     $ rosrun hello_world hello_world_publisher.py
     ```

   - для управления узлом, который подписывается на тему `hello_pub`, служит команда:

     ```
     $ rosrun hello_world hello_world_subscriber.py
     ```

Узлы запущены, и в Терминале вы увидите сообщение, похожее на приведенное ниже:

```
SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
```

```
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [2316]
ROS_MASTER_URI=http://alex-VirtualBox:11311/

setting /run_id to 0a1de67e-f011-11e8-949a-0800278d5979
process[rosout-1]: started with pid [2329]
started core service [/rosout]
rosrun hello_world hello_world_publisher.py
rosrun hello_world hello_world_subscriber.py
```

В этом разделе мы рассмотрели основные функции ROS. Теперь необходимо познакомиться с Gazebo и узнать, как, используя ROS, работать с этой программой.

Введение в Gazebo

Gazebo – это программа с открытым исходным кодом, моделирующая работу робота. С помощью данного симулятора мы можем не только проверить конструкцию и алгоритмы разрабатываемого робота, но и выполнить тестирование с использованием реалистичных сценариев. Gazebo может точно и эффективно имитировать работу робота в помещении и вне его. Gazebo создана на движке с высоким качеством графики и удобным программно-графическим интерфейсом.

Отметим следующие особенности Gazebo:

- **динамическое моделирование**. Gazebo может имитировать движение робота с помощью такого имитатора движения (кинематики и динамики), как Open Dynamics Engine – ODE (http://opende.sourceforge.net/), Bullet (http://bulletphysics.org/wordpress/), Simbody (https://simtk.org/projects/simbody/) и DART (Dynamic Animation and Robotics Toolkit – Набор инструментов для динамической анимации и робототехники) (http://dartsim.github.io/);
- **улучшенная 3D-графика**. Gazebo с помощью фреймворка OGRE обеспечивает высокое качество визуализации, освещения, тени и http://www.ogre3d.org/;
- **поддержка датчиков**. Gazebo поддерживает широкий спектр датчиков, в том числе лазерные дальномеры, сенсоры kinect style, 2D/3D-камера и т. д. Мы можем имитировать движение как с препятствиями, так и без них;
- **Plug-in**. Мы можем разрабатывать плагины для робота и всех его датчиков, включая датчики контроля окружающей среды. Плагины могут получить доступ к API Gazebo;

- **модели роботов**. Gazebo может создавать модели популярных роботов, таких как PR2, Pioneer 2 DX, iRobot и TurtleBot. Мы можем также построить нестандартные модели роботов;
- **протокол TCP/IP**. С помощью службы передачи сообщений socket-based мы можем запустить на удаленной машине интерфейс Gazebo и моделирование;
- **облако симуляции**. Используя фреймворк CloudSim (http://cloudsim.io/), мы можем запустить моделирование на облачном сервере;
- **инструменты командной строки**. Для проверки используются инструменты командной строки и протоколы моделирования.

Установка Gazebo

Gazebo может устанавливаться в двух вариантах: как автономная программа или как приложение, интегрированное с ROS. В этой главе мы установим Gazebo, интегрированную с ROS, чтобы при моделировании проверить код, написанный с использованием фреймворка ROS.

Последняя версия симулятора Gazebo находится здесь: http://gazebosim.org/download.

Для работы с симулятором Gazebo не нужно устанавливать отдельно от ROS, т. к. при полной установке ROS Gazebo устанавливается автоматически.

Пакет ROS с интегрированным симулятором Gazebo называется `gazebo_ros_pkgs`. Этот пакет с помощью служб сообщений ROS предоставляет интерфейс для моделирования робота в Gazebo.

Для установки полной версии пакета `gazebo_ros_pkgs` в ROS Kinetic выполните следующую команду:

```
$ sudo apt-get install ros-kinetic-gazebo-ros-pkgs ros-kinetic-ros-control
```

Тестирование Gazebo с интерфейсом ROS

Мы предполагаем, что среда ROS настроена правильно.

Прежде чем запустить Gazebo, запустите `roscore`, если он не запущен:

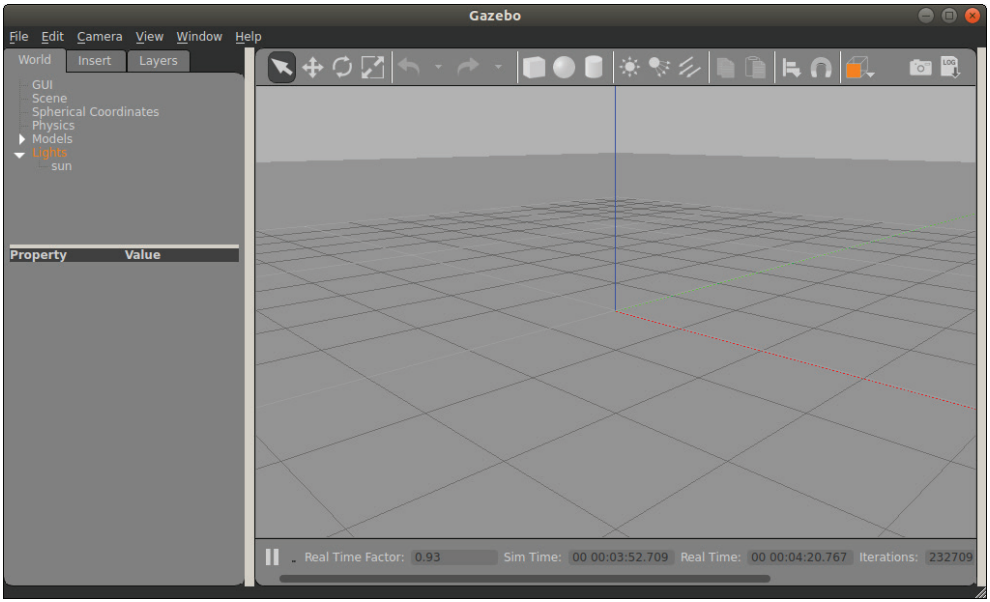
```
$ roscore
```

Чтобы с помощью ROS запустить Gazebo, выполните следующую команду:

```
$ rosrun gazebo_ros gazebo
```

Gazebo работает как два исполняемых файла: сервер Gazebo и клиент Gazebo. Gazebo-сервер выполняет процесс моделирования, а Gazebo-клиент вызывает графический интерфейс. С помощью предыдущей команды `$ rosrun gazebo_ros gazebo` сервер Gazebo и клиент Gazebo будут работать одновременно.

Графический интерфейс Gazebo показан на следующем скриншоте:



Имитатор Gazebo

Итак, Gazebo запущена.

Чтобы отобразить список сгенерированных тем, выполните следующую команду. Возможно, для ее ввода вам придется запустить еще один терминал. Для этого в строке меню терминала выберите команду **File** (Файл) → **Open Terminal** (Открыть терминал). Ниже вы увидите список сгенерированных тем.

```
$ rostopic list
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
```

Сервер и клиент можно запустить отдельно.

- Чтобы запустить сервер Gazebo, используйте следующую команду:

```
$ rosrun gazebo_ros gzserver
```

- Клиент Gazebo запускается с помощью команды:

```
$ rosrun gazebo_ros gzclient
```

Итоги

Это была вводная глава, благодаря которой мы познакомились с основными понятиями операционной системы роботов (ROS), ее установкой и программированием с помощью языка программирования Python. Кроме того, мы рассмотрели, как установить и запустить программу – имитатор роботов Gazebo. В следующей главе мы познакомимся с основными принципами дифференциального привода робота.

Вопросы

1. О каких важных особенностях ROS вы узнали из этой главы?
2. В чем различие уровней концепций в ROS?
3. Что такое система сборки ROS catkin?
4. Что такое темы и сообщения ROS?
5. В чем разница концепций графа вычислений ROS?
6. Какая основная функция ROS Master?
7. О каких важных особенностях Gazebo вы узнали?

Глава 2

Основные понятия роботов с дифференциальным приводом

В предыдущей главе мы познакомились с основными понятиями и установкой операционной системы роботов ROS и с программой – имитатором робота Gazebo. Как ранее уже упоминалось, цель этой книги – помочь читателю создать автономный мобильный робот с нуля. Перемещаться этот робот будет с помощью колес. Проектируемый нами робот будет оснащен двумя ведущими колесами, которые мы установим на противоположных сторонах шасси. Управлять направлением движения мы будем с помощью разности скорости вращения ведущих колес. Например, если скорость вращения правого колеса будет меньше, чем скорость вращения левого, робот будет поворачивать вправо, и наоборот, если правое колесо будет вращаться быстрее левого, робот будет поворачивать влево.

Эта глава поможет вам провести математический анализ движения робота и решить уравнение кинематики робота. Уравнение кинематики даст вам возможность с помощью данных, получаемых от датчиков робота, предсказать его положение во время движения.

В этой главе будут рассмотрены следующие темы:

- математическое моделирование робота;
- введение в систему дифференциального привода и кинематику робота;
- инверсная кинематика.

Математическое моделирование робота

Наиболее важная часть мобильного робота – это система управления. С помощью нее робот ориентируется и перемещается в пространстве. Одной из самых простых и рентабельных систем управления является система дифференциального привода. Дифференциальный привод состоит из двух неза-

висимых друг от друга ведущих колес, установленных на общей оси. Каждое ведущее колесо приводится в движение отдельным двигателем. Дифференциальный привод – это неголономная система, которая имеет ограничения в изменении положения робота. Примером неголономной системы является автомобиль, поскольку он не может изменить положение без изменения позы. Таким образом, имея три степени свободы (3D), автомобиль с помощью газа, тормоза и рулевого управления может использовать только две степени свободы (2D).

Напомним, **голономная система** – это механическая система, все **механические связи** которой можно свести к геометрическим. Такие связи накладывают ограничения только на положение системы, но не на величины скоростей.

Механической связью называют ограничения, накладываемые на координаты и скорости механической системы, которые должны выполняться на любом ее движении.

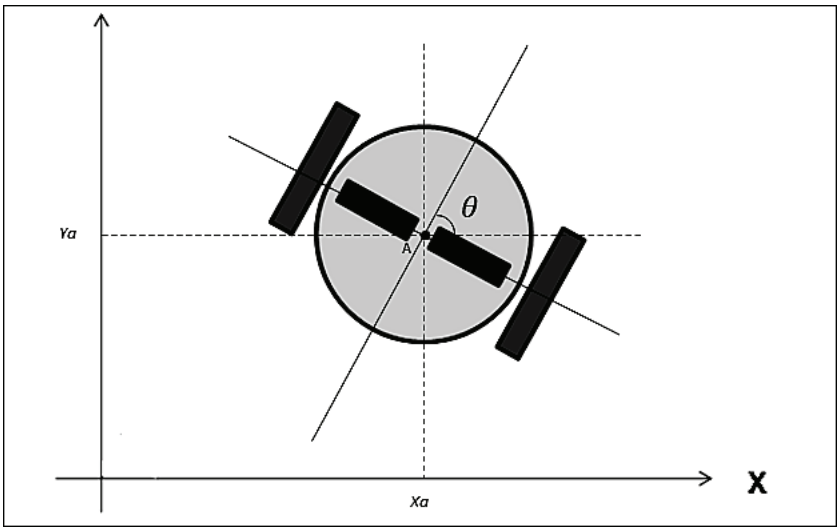
Если все кинематические связи нельзя свести к геометрическим связям, то данная система будет **неголономной**. Таким образом, **неголономная система** – это механическая система, на которую, кроме геометрических связей, наложены еще и дифференциальные (кинематические). Их нельзя свести к геометрическим связям.

Далее мы рассмотрим работу робота с дифференциальным приводом и возможность смоделировать его математически.

Введение в систему дифференциального привода и кинематику робота

Кинематика – изучение движения объекта с точки зрения математики. Она рассматривает движение объекта без учета влияющих на него внешних сил. Здесь в основном внимание привлекает геометрическое отношение деталей, обеспечивающих управление конструкцией. Динамика – это исследование движения объекта с учетом всех сил, влияющих на робота.

Мобильный робот имеет шесть степеней свободы (DOF). Три степени свободы: x, y и z – связаны с тремя координатными осями, позволяющими определить положение объекта в трехмерном пространстве. Остальные три степени свободы относятся к ориентации робота в пространстве. Это такие значения, как крен (боковой наклон, или раскачивание корпуса робота относительно оси движения), тангаж (наклон аппарата относительно горизонтальной поперечной оси, т. е. наклон вниз или подъем вверх передка робота) и рыскание (небольшие изменения направления движения аппарата вправо или влево относительно его курса). Робот с дифференциальным приводом перемещается в двухмерной плоскости (2D), и его положение в любой момент можно описать двумя глобальными координатами X и Y, лежащими в горизонтальной плоскости. При этом курс робота обозначается как θ. Этих данных вполне достаточно, чтобы описать положение робота с дифференциальным приводом.



Положение робота X, Y и θ в глобальной системе координат

Как уже говорилось ранее, дифференциальный привод робота основан на разности между скоростью вращения правого и левого ведущих колес. Если скорость вращения ведущих колес одинакова, робот движется прямо. При изменении скорости вращения одного ведущего колеса относительно другого робот будет выполнять поворот в ту сторону, в которой скорость вращения ведущего колеса меньше по отношению ко второму ведущему колесу. Ниже на рисунке представлена пара популярных роботов с дифференциальным приводом:

iRobot Roomba
(https://en.wikipedia.org/wiki/IRobot)

Pioneer 3-DX
(http://robots.ros.org/ pioneer-3-dx/)

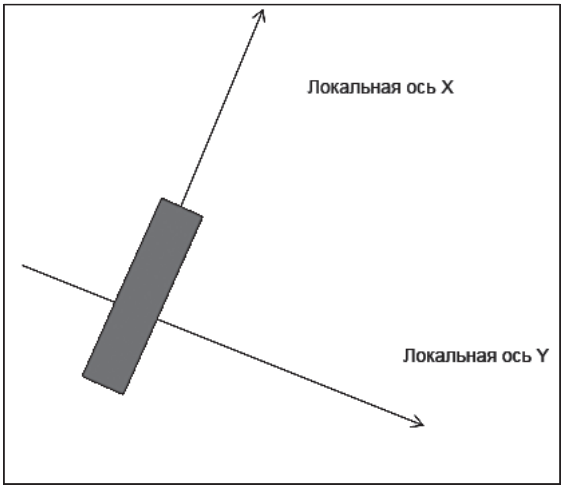
Прямая кинематика дифференциального робота

С помощью уравнений кинематики для робота с дифференциальным приводом определяется положение устройства. Представим, что начальное положение робота в момент времени t – X, Y, θ. Нам требуется определить, какое положение устройство займет (X', Y', θ') за промежуток времени $t + \delta t$. При этом следует учесть следующие параметры: *v-left* – скорость левого и *v-right* – скорость правого колеса.

Эта методика расчета положения используется для проводки робота по требуемой траектории.

Объяснение уравнений прямой кинематики

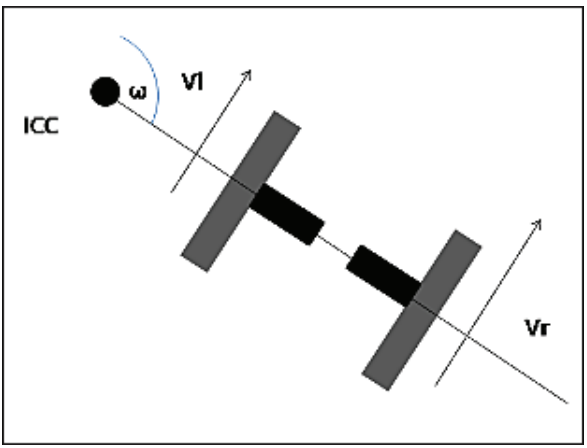
Объяснение уравнения кинематики мы можем начать с формулирования решения для движения робота вперед без учета его массы и действующих на объект сил. Показанный ниже рисунок иллюстрирует кинематику одного из колес робота:



Колесо робота, вращающееся по локальной оси Y

Наклон вправо или влево относительно оси Y называется креном; все остальное можно рассматривать как скольжение. Предположим, что колесо не проскальзывает. В этом случае за один оборот обод колеса пройдет расстояние, равное $2\pi r$, где r – радиус колеса. Будем считать, что движение происходит в двухмерной плоскости с плоской ровной поверхностью. Выполняя поворот,

робот будет поворачиваться вокруг точки, находящейся на оси, совпадающей с осью правого и левого ведущих колес. Эта точка находится за пределами робота и называется **мгновенным центром кривизны** (ICC). На следующем рисунке вы увидите конфигурацию ведущих колес робота с дифференциальным приводом по отношению к точке мгновенного центра кривизны (ICC).



Конфигурация колеса робота с дифференциальным приводом

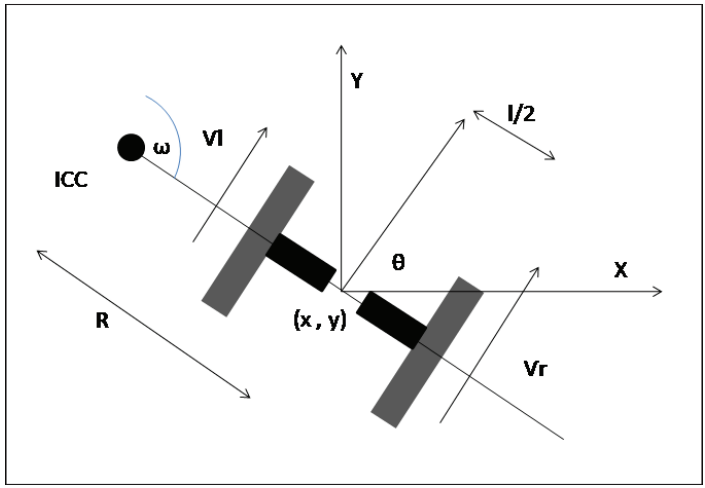
Основное понятие, без которого не получится вывести уравнение кинематического привода, – это угловая скорость робота, обозначаемая буквой ω. При повороте ведущие колеса робота катятся по окружности, центр которой совпадает с мгновенным центром кривизны (ICC). Скорость колеса описывается формулой $v = 2\pi r/T$, где T – время, затраченное на прохождение колесом расстояния, равного длине полной окружности с центром, совпадающим с точкой ICC. Угловая скорость ω определяется как $2\pi/T$ и измеряется в радианах (градусах) в секунду. Радиан – это угол, соответствующий дуге, длина которой равна радиусу этой дуги. 1 радиан равен 57,3°.

Если объединить уравнения для расчета скорости колеса v и его угловой скорости ω, получим уравнение $\omega = 2\pi/T$.

$$v = r\omega. \quad (1)$$

Уравнение линейной скорости

Схема дифференциального привода показана на следующем рисунке:



Детальная схема системы дифференциального привода

Если применить предыдущее уравнение к обоим ведущим колесам, результат будет одинаковым, т. е. ω:

$$\omega(R + l/2) = Vr. \quad (2)$$
$$\omega(R - l/2) = Vl. \quad (2)$$

Уравнение дифференциального привода колес робота

При этом R – расстояние между ICC и центром оси, проходящей через колеса, а l – длина оси колеса. Преобразовав выражения ω и R, мы получим следующий результат:

$$R = l/2(Vl + Vr)/(Vr - Vl); \quad (4)$$
$$\omega = (Vr - Vl)/l. \quad (5)$$

Уравнение для вычисления расстояния от ICC до центра оси робота и расчета угловой скорости робота

Предыдущие уравнения позволяют решать поставленные перед нами задачи кинематики. Предположим, что робот движется с угловой скоростью в течение ωt секунд. В этом случае траектория движения и направление робота изменятся:

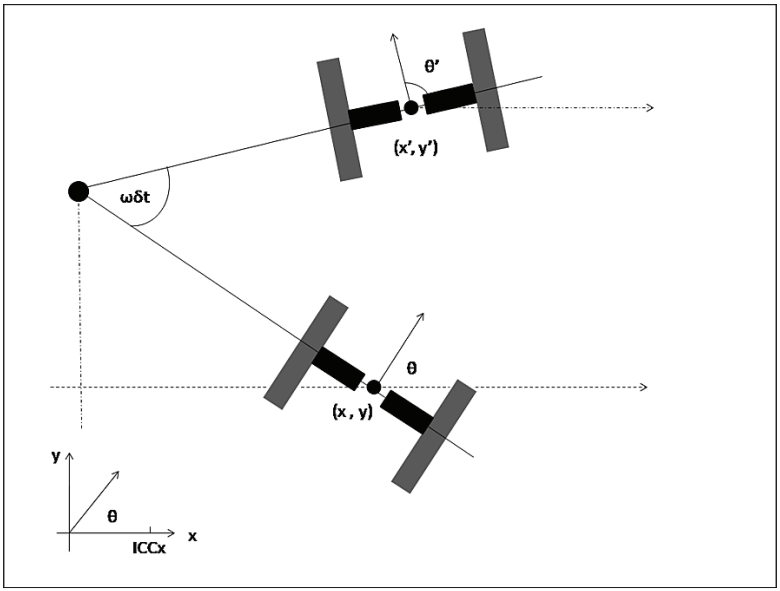
$$\theta' = \omega\delta t + \theta. \quad (6)$$

Уравнение для расчета изменения траектории движения

Центр вращения ICC определяется базовой тригонометрией:

$$ICC = [ICC_x, ICC_y] = [x - R\sin\theta, y + R\cos\theta]. \quad (7)$$

Уравнение для поиска ICC



Поворот робота относительно ICC на угол $\omega\delta t$ в градусах

С учетом стартовой позиции (x, y) новое положение (x', y') может быть вычислено с помощью 2D-матрицы вращения. Учитывая движение робота с угловой скоростью ω в течение δt секунд относительно точки ICC, мы получим следующую позицию для времени $t + \delta t$:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) \\ \sin(\omega\delta t) & \cos(\omega\delta t) \end{pmatrix} \begin{pmatrix} x - ICC_x \\ y - ICC_y \end{pmatrix} + \begin{pmatrix} ICC_x \\ ICC_y \end{pmatrix}. \quad (8)$$

Уравнение для вычисления новой позиции робота

Учитывая ω, δt и R, новое положение робота (x', y' и θ') можно вычислить с помощью уравнений (6) и (8). Угловая скорость ω вычисляется с помощью уравнения (5); точно измерить скорости Vr и Vl затруднительно.

Вместо непосредственного вычисления скорости колеса можно измерить его реальную скорость. Измерение реальной скорости производится с помощью **энкодера**. Энкодер – датчик, преобразующий подконтрольную величину в электрический сигнал. В нашем случае энкодер для определения угла поворота колеса вокруг своей оси фиксирует импульсы. Угол поворота колеса

определяется количеством зафиксированных импульсов. В качестве рабочей величины применяется количество импульсов на один полный оборот колеса. Значениями **одометрии** робота являются данные от колесных энкодеров. Энкодеры установлены на осях колеса и передают двоичные сигналы для каждого шага вращения колеса (каждый шаг (step) равен 0,1 мм). Сигналы от энкодеров, полученные за временной промежуток от t до $t + \delta t$, подаются на счетчик импульсов, в результате чего можно вычислить пройденное расстояние $v\delta t$:

$n \times \text{step} = v\delta t$.

Здесь n – количество импульсов, зафиксированных энкодером за заданное время.

Из этого можно вычислить v:

$$v = n \times \text{step} / \delta t. \quad (9)$$

Уравнение для вычисления линейной скорости с помощью данных энкодера

Если мы подставим уравнение (9) в уравнения (3) и (4), то получим следующий результат:

$$R = l/2(Vl + Vr)/(Vr - Vl) = l/2(nl + nr)/(nr - nl); \quad (10)$$
$$\omega\delta t = (Vr - Vl)\delta t/l = (nr - nl) \times \text{step}/l. \quad (11)$$

Уравнение для вычисления R с помощью данных, полученных от энкодера

Здесь nl и nr – количество импульсов, полученных от энкодеров левого и правого колес. Vl и Vr – это скорости левого и правого колес. Итак, робот перемещается с позиции (x, y, θ) в позицию (x', y', θ'). При этом от датчиков левого и правого колес в течение времени δt соответственно поступит nl и nr импульсов. В этом случае новое положение робота определяется по формуле:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x - \text{ICC}_x \\ y - \text{ICC}_y \\ \theta \end{pmatrix} + \begin{pmatrix} \text{ICC}_x \\ \text{ICC}_y \\ \omega\delta t \end{pmatrix}, \quad (12)$$

Уравнение для вычисления положения робота с помощью данных, полученных от датчиков правого и левого колес

где

$$R = l/2(nl + nr)/(nr - nl); \quad (13)$$
$$\omega\delta t = (nr - nl) \times \text{step}/l; \quad (14)$$
$$\text{ICC} = [x - R\sin\theta, y + R\cos\theta]. \quad (15)$$

Уравнение для вычисления ICC и других параметров с помощью данных, полученных от датчиков правого и левого колес

Полученное кинематическое уравнение в основном зависит от конструкции и размеров деталей ходовой части робота. Изменение конструкции может привести к корректировке уравнений.

Обратная кинематика

С помощью уравнений прямой кинематики мы можем определить новое положение робота, учитывая скорость вращения колеса. Теперь можно приступить к обратной задаче.

Допустим, робот в момент времени t находится в положении (x, y, θ). Определим скорости левого (*V-left*) и правого (*V-right*) колес так, что за промежуток времени $t + \delta t$ робот займет новое положение с координатами (x', y' и θ').

В устройствах с дифференциальным приводом переместить робота в нужную позицию с помощью только простой установки скорости вращения колес нельзя. Как уже упоминалось ранее, дифференциальный привод – это неголономная механическая система, в которой, кроме геометрических, учитываются и кинематические связи. Неголономная система накладывает некоторые ограничения на управление роботами.

Для увеличения степени подвижности (управляемости) неголономного робота следует использовать разность скоростей левого и правого колес. Если вставить значения из уравнений (12) в уравнение (15), то можно определить способы управления, которые в дальнейшем сможем запрограммировать:

- если $V\text{-}left = V\text{-}right$, => $nr = nl$ => $R = \infty$, $\omega\delta T = 0$ (т. е. скорость и направление вращения левого ведущего колеса совпадают со скоростью и направлением вращения правого колеса), значит, робот движется прямолинейно и θ остается неизменной;
- если $V\text{-}left = -V\text{-}right$, => $nr = -nl$, => $R = 0$, $\omega\delta T = 2nl \times \text{step}/l$ и $\text{ICC} = [\text{ICC}_x, \text{ICC}_y] = [x, y]$ => $x' = x$, $y' = y$, $\theta' = \theta + \omega\delta t$ => (т. е. скорости вращения левого и правого колес одинаковы, но направления их вращения не совпадают, колеса вращаются навстречу друг другу), значит, робот вращается вокруг мгновенного центра кривизны (ICC), т. е. $\forall\theta$ может изменяться без ограничений при неизменных координатах [x, y].

Объединяя приведенные ниже способы управления, можно создать алгоритм, позволяющий достичь любого положения робота относительно его начального положения:

1) поворачивайте, пока ориентация робота не совпадет с линией, ведущей из исходного в целевое положение, $V\text{-}right = -V\text{-}left = V\text{-}rot$ (скорость поворота);
2) двигайтесь прямо до тех пор, пока положение робота не совпадет с целевым положением, $V\text{-}right = V\text{-}left = V\text{-}ahead$ (скорость прямого движения);
3) поворачивайте, пока ориентация робота не совпадет с ориентацией цели, $V\text{-}right = -V\text{-}left = V\text{-}rot$. Скорости *V-rot* и *V-ahead* выбираются произвольно.

Для более подробной информации по кинематике смотрите http://www8.cs.umu.se/~thomash/reports/KinematicsEquationsForDifferentialDriveAndArticulatedSteeringUMINF-11.19.pdf.

Итоги

Эта глава была посвящена не только фундаментальным концепциям роботов с дифференциальным приводом, но и кинематическим уравнениям движения таких устройств. В начале главы были рассмотрены основы дифференциального привода роботов. Далее мы вывели использующиеся в таких устройствах уравнения прямой кинематики и объяснили их с помощью кинематических схем. Затем познакомились с уравнениями обратной кинематики. В следующей главе рассмотрим, как с помощью ROS и Gazebo создать модель автономного мобильного робота.

Вопросы

1. Что такое голономные и неголономные системы?
2. Что такое кинематика и динамика робота?
3. Что такое ICC робота с дифференциальным приводом?
4. Что такое уравнение прямой кинематики для робота с дифференциальным приводом?
5. Что такое уравнение обратной кинематики для робота с дифференциальным приводом?

Дополнительная информация

http://people.cs.umu.se/thomash/reports/KinematicsEquationsForDifferentialDriveAndArticulatedSteeringUMINF-11.19.pdf.

Глава 3

Моделирование робота с дифференциальным приводом

В этой главе мы рассмотрим модель робота с дифференциальным приводом и создадим URDF-модель робота в ROS. **URDF** – это универсальный формат XML-файла, используемый для описания всех элементов робота в ROS.

Устройство, которое мы собираемся построить, используется в качестве робота-помощника в отелях и ресторанах и предназначено подавать еду и напитки. Название такого робота – Chefbot.

В этой главе мы полностью смоделируем робот, познакомимся с созданной с помощью системы CAD (САПР – системы автоматического проектирования) конструкцией всех механических элементов и последовательностью сборки. Кроме того, мы рассмотрим конструкцию этого устройства в 2D- и 3D-проекциях и создадим его модель в формате URDF.

В отелях работают роботы больших размеров. Мы же построим робот для тестирования программного обеспечения. Поэтому он будет миниатюрным. Если вы желаете построить его с нуля, т. е. от начала и до конца, значит, эта глава для вас. Если же вы не желаете самостоятельно собирать робот из отдельных деталей, воспользуйтесь роботизированной платформой, например Turtlebot. Эта роботизированная платформа уже есть на рынке.

Прежде чем приступить к созданию робота, следует определиться с техническим заданием. Затем можно начать его проектирование в системе автоматического проектирования САПР (CAD). Сначала мы создадим двухмерную модель (2D), которую впоследствии преобразуем в трехмерную (3D). Трехмерная модель позволит более тщательно и детально проанализировать конструкцию этого устройства. После того как 3D-модель будет создана, преобразуем ее в формат URDF. Это позволит смоделировать робот в ROS.

В этой главе мы рассмотрим следующие темы:

- проектирование параметров робота в заданной спецификации;

- проектирование деталей робота в 2D с использованием LibreCAD;
- проектирование 3D-модели робота с использованием Blender и Python;
- создание URDF-модели Chefbot;
- визуализация модели Chefbot в Rviz (ROS Visualizer).

Технические требования

Для тестирования приложения и кода нам понадобится PC/ноутбук с операционной системой Ubuntu 16.04 LTS и установленной ROS Kinetic.

Требования к сервисному роботу

Прежде чем начать проектирование робототехнической системы, необходимо определить технические требования к разрабатываемому роботу. Ниже представлен список технических требований, предъявляемых к проектируемому устройству:

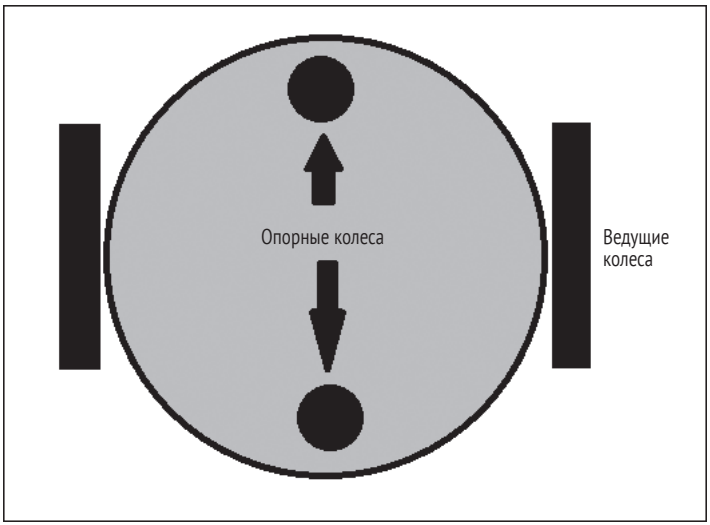
- робот должен иметь устройство для переноски продуктов питания;
- робот должен перевозить груз массой до 2 кг;
- скорость передвижения должна быть в диапазоне от 0,25 до 0,35 м/с;
- дорожный просвет робота должен быть не менее 3 см;
- робот должен непрерывно работать в течение 2 ч;
- робот должен уметь передвигаться, избегая препятствий, и подавать пищу на любой стол;
- высота робота должна быть от 80 до 100 см;
- робот должен быть недорогим (себестоимость не более $500).

Теперь, когда мы определились с техническими требованиями, такими как вес полезной нагрузки, скорость передвижения, дорожный просвет, высота и стоимость робота, можем приступать к его проектированию. Сконструируем корпус и выберем соответствующие компоненты. Давайте обсудим, какой механизм робота будет использован, чтобы устройство соответствовало предъявляемым техническим требованиям.

Приводной механизм ходовой части робота

Одной из наиболее экономически эффективных систем навигации мобильного робота является система дифференциального привода. Это один из наиболее простых механизмов привода, который используется в основном для перемещения домашнего робота. Дифференциальный привод состоит из двух ведущих колес, установленных на общей оси. Каждое ведущее колесо приводится в движение отдельным двигателем. Для большей устойчивости и равномерного распределения веса робот оснащен двумя опорными колесами.

Ниже показана типичная схема системы дифференциального привода.



Дифференциальная система привода

Далее определимся с механическими компонентами дифференциальной системы привода: двигателями, колесами и шасси. Учитывая технические требования, в первую очередь выберем двигатель.

Выбор двигателей и колес

Двигатели выбираются с учетом их технических характеристик. Наиболее важные технические характеристики двигателя – число оборотов в минуту (RPM) и крутящий момент. Эти значения определяются с помощью технических условий, предъявляемых к проектируемому устройству.

Расчет оборотов двигателя

Согласно техническим требованиям скорость передвижения робота должна находиться в диапазоне от 0,25 до 0,35 м/с. Учитывая, что значение дорожного просвета должно быть не менее 3 см, выберем диаметр ведущего колеса, равный 9 см. В этом случае расстояние от поверхности до центра оси колеса составит 4,5 см, требование по дорожному просвету будет выполнено. Используя следующее уравнение, рассчитаем количество оборотов колеса в минуту:

об/мин = (60 × Скорость)/(3,14 × Диаметр колеса);

$$\text{об/мин} = \frac{(60 \times 0{,}35)}{3{,}14 \times 0{,}09} = \frac{21}{0{,}2826} = 74 \text{ об/мин}.$$

Вы также можете использовать для расчетов следующую ссылку: http://www.robotshop.com/blog/en/vehicle-speed-rpm-and-wheel-diameterfinder-9786.

Для устройства, перемещающегося со скоростью 0,35 м/с, расчетное количество оборотов в минуту колеса диаметром 9 см (90 мм) составляет 74. Применив округление, в качестве стандартной величины определим значение, равное 80 об/мин. Учитывая, что колесо будет крепиться прямо на ведущий вал двигателя, необходимо выбрать двигатель со встроенным редуктором и количеством оборотов на ведущем валу, равным 80 об/мин.

Расчет крутящего момента двигателя

Рассчитаем крутящий момент, необходимый для перемещения робота.

1. Количество колес – 4 (2 ведущих плюс 2 опорных колеса).
2. Количество моторов – 2.
3. Предполагаем, что коэффициент трения равен 0,6, а радиус колеса – 4,5 см (45 мм).
4. Определим полную массу робота = Масса робота + груз: = ($W = mg$) = ($\sim 100N + \sim 20N$) $W = \sim 150N$, из этого общая масса = 12 кг.
5. Нагрузка, действующая на четыре колеса, может быть записана как $W = 2 \times N1 + 2 \times N2$, где $N1$ – количество опорных колес и $N2$ – количество ведущих колес.
6. Предположим, что робот находится в неподвижном состоянии. Максимальный крутящий момент требуется в начале движения робота, т. к. ему необходимо преодолеть силу трения.
7. В момент начала движения момент импульса и сила трения уравновешивают друг друга. Зная силу трения, мы получим максимальное значение крутящего момента (момент импульса):

- $\mu \times N \times r - T = 0$, где μ – коэффициент трения, N – средний вес, действующий на каждое колесо, r – радиус колеса, T – крутящий момент;
- $N = W/2$ (в роботе ведущими являются только 2 колеса, поэтому для вычисления максимального крутящего момента мы берем $W/2$). В результате получаем: $0{,}6 \times (120/2) \times 0{,}045 - T = 0$;
- Следовательно, ***T* = 1,62 Н/м**, или **16,51 кг/см**.

Результаты проектирования

После проектирования полученные значения округлим до стандартных и подберем комплектующие из доступной спецификации:

- количество оборотов двигателя = 80 об/мин (приведено к стандартному значению);
- крутящий момент двигателя = 18,0 кг/см (приведено к стандартному значению);
- диаметр колеса = 9 см (90 мм).

Конструкция шасси робота

После расчетов параметров двигателей и колес робота можем спроектировать его шасси. Согласно техническому заданию на шасси робота необходимо смон-

тировать подставку для доставки еды, выдерживающую до 2 кг полезной нагрузки. Дорожный просвет робота должен составлять не менее 3 см (30 мм). Себестоимость робота должна быть минимальной. Помимо этого, в роботе необходимо предусмотреть место для размещения электронных компонентов, таких как персональный компьютер (ПК), датчики и батарея.

Одной из самых простых и удовлетворяющих требованиям технического задания (ТЗ) конструкций является многоуровневая конструкция, например **TurtleBot 2** (робот-черепаха) (http://www.turtlebot.com/) с шасси, состоящим из трех полок – уровней. В качестве ходовой части TurtleBot используется роботизированная платформа **Kobuki** (http://kobuki.yujinrobot.com/about2/). Она оснащена встроенными моторами и датчиками, поэтому нет необходимости дополнительно разрабатывать ходовую часть робота.

На следующем рисунке показана конструкция шасси робота TurtleBot 2:

Робот TurtleBot 2 (http://robots.ros.org/turtlebot/)

Мы конструируем робот, ориентируясь на конструкцию TurtleBot 2. Ходовую часть робота разработаем самостоятельно. Наша конструкция также будет трехуровневой. Прежде чем начинать проектирование, следует определиться с необходимым программным обеспечением.

Для разработки конструкции робота понадобится **система автоматического проектирования САПР**, или **CAD** (computer-aided design/drafting – система автоматизированного проектирования и создания чертежей). Ниже приведен список наиболее популярных систем автоматического проектирования:

- SolidWorks (http://www.solidworks.com/default.htm);
- AutoCAD (http://www.autodesk.com/products/autocad/overview);
- Maya (http://www.autodesk.com/products/maya/overview);
- Inventor (http://www.autodesk.com/products/inventor/overview);
- Google SketchUp (http://www.sketchup.com/);
- Blender (http://www.blender.org/download/);
- LibreCAD (http://librecad.org/cms/home.html).

Конструировать шасси можно в любой удобной для вас системе автоматического проектирования. В этой книге для разработки чертежей и двухмерной модели робота воспользуемся программой LibreCAD. Трехмерное моделирование выполним в программе Blender. Эти программы бесплатны и работают во всех операционных системах. Для просмотра и контроля создаваемой 3D-модели воспользуемся программой MeshLab. Linux Ubuntu будет основной операционной системой. Также мы рассмотрим, как установить нужные нам приложения в Linux Ubuntu 16.04, и предоставим ссылки на инструкции по установке этих приложений в других операционных системах.

Установка LibreCAD, Blender и MeshLab

LibreCad – это бесплатная, с открытым исходным кодом система автоматического проектирования (САПР) для двухмерного (2D) черчения и проектирования. Работает с операционными системами Windows, OS X и Linux. Распространяется под лицензией GNU General Public License версии 2. **Blender** – это бесплатная, с открытым исходным кодом программа для создания трехмерной компьютерной графики со средствами моделирования, анимации, рендеринга, постобработки, монтажа видео и звука и создания интерактивных игр. Приложение распространяется с лицензией GPL, благодаря чему пользователи могут обмениваться программой, изменять и распространять ее. **MeshLab** – редактор с открытым исходным кодом, предназначенный для обработки неструктурированных 3D-моделей.

Ниже приведены ссылки для скачивания и установки в Windows, Linux и OS X:

- зайдите по http://wiki.librecad.org/index.php/Download для скачивания LibreCAD;
- зайдите по https://launchpad.net/~librecad-dev для скачивания и установки LibreCAD на Debian/Ubuntu;
- зайдите по https://apps.fedoraproject.org/packages/librecad для скачивания и установки LibreCAD на Fedora;
- зайдите по https://sourceforge.net/projects/librecad/files/OSX/ для скачивания и установки LibreCAD на OS X;
- зайдите по https://sourceforge.net/projects/librecad/files/Windows/ для скачивания и установки LibreCAD на Windows;
- зайдите по https://github.com/LibreCAD/LibreCAD/releases для разработки и тестирования.

Установка LibreCAD

Следуйте стандартной процедуре установки для всех операционных систем. Если у вас операционная система Ubuntu, можете установить LibreCAD с помощью центра приложений Ubuntu.

Также для установки LibreCAD в Linux Ubuntu можно воспользоваться следующими командами:

```
$ sudo add-apt-repository ppa:librecad-dev/librecad-stable
$ sudo apt-get update
$ sudo apt-get install librecad
```

Если после запуска Терминала вы ввели команду sudo su и прошли авторизацию, то больше команду sudo вводить не надо.

Установка Blender

Для загрузки версии Blender, соответствующей вашей операционной системе, посетите страницу, расположенную по адресу: http://www.blender.org/download/. Здесь вы можете найти последнюю версию Blender. Актуальная документация по Blender находится здесь: https://docs.blender.org/manual/ru/dev/. Если у вас установлена операционная система Linux Ubuntu, то Blender устанавливается с помощью менеджера пакетов Synaptic или команды:

```
$ sudo apt-get install blender
```

Установка MeshLab

Редактор MeshLab доступен для всех операционных систем. Чтобы скачать программу и исходный код MeshLab, перейдите по ссылке: http://meshlab.sourceforge.net/.

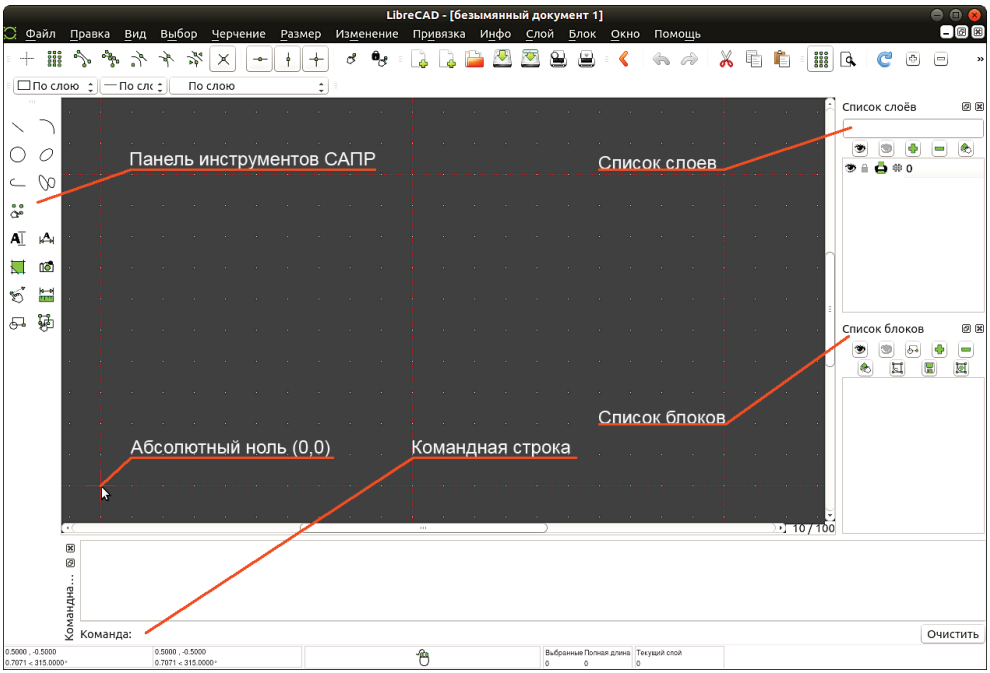
Если у вас установлена Linux Ubuntu, MeshLab можно установить через менеджер пакетов Synaptic или с помощью следующей команды:

```
$sudo apt-get install meshlab
```

Когда программы будут установлены, закройте Терминал, два раза введя команду **Exit**.

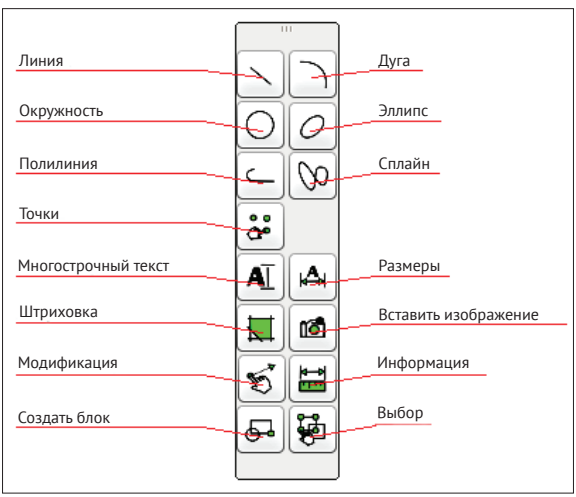
Создание 2D CAD-чертежа робота с помощью LibreCAD

Познакомимся с базовым интерфейсом LibreCAD. Ниже показано окно этой программы:



Рабочее окно программы LibreCAD

Ниже показана панель инструментов САПР, с помощью которых и будут создаваться чертежи:

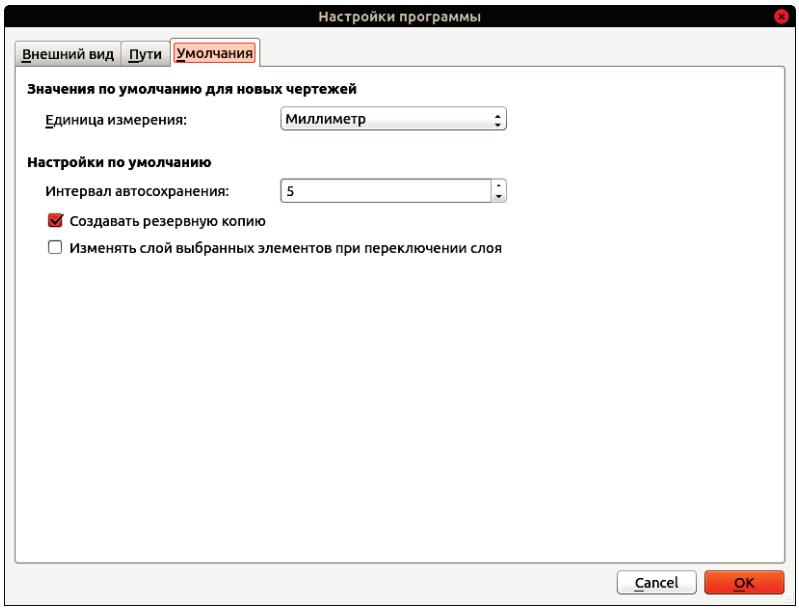


Панель инструментов LibreCAD

Подробное описание инструментов LibreCAD доступно по ссылке: http://wiki.librecad.org/index.php/LibreCAD_users_Manual.

- **Командный блок.** Он включает в себя командную строку, в которую вводятся команды, и поле, в котором отображается история введенных ранее команд. С помощью команд можно создавать чертежи без использования панели инструментов САПР.
  Чтобы построить прямоугольник размером 40 × 30 с координатами начальной точки в левом нижнем углу x = 10; y = 20, щелкните мышью на командной строке и в зависимости от языка интерфейса последовательно введите команды: `line` (линия), **Enter**; `10,20,` **Enter**; `50,20,` **Enter**; `50,50,` **Enter**; `10,50,` **Enter**; `10,20,` **Enter**; 2 раза нажмите **Esc**. Прямоугольник построен. Подробные инструкции об использовании командной строки можно прочитать, перейдя по адресу: http://wiki.librecad.org/index.php/A_short_manual_for_use_from_the_command_line.
- **Список слоев**. Каждый чертеж состоит из нескольких слоев. Например, на одном слое чертится основной чертеж, на другом наносятся размеры с размерными линиями, на третьем – текст и т. д. Стиль, толщину, цвет и т. д. для каждого слоя можно назначить индивидуально. Использование нескольких слоев при создании чертежа – это основная концепция всех САПР. Подробнее об использовании слоев можно прочитать здесь: http://wiki.librecad.org/index.php/Layers.
- Блок – это группа примитивов, которые могут быть несколько раз вставлены в один и тот же чертеж в разных местах и с различными характеристиками (разным масштабом и углом поворота). Подробнее описание блоков можно прочитать, перейдя по следующей ссылке: http://wiki.librecad.org/index.php/Blocks.
- Абсолютный ноль – это начало координат, на рисунке (0, 0).

Итак, начнем чертить. Для этого воспользуемся блоком для черчения. Сначала выберем единицы измерения. В технике используются миллиметры, в строительстве – сантиметры. Назначим единицей измерения миллиметр. Для этого после запуска программы **LibreCAD** следует выбрать команду меню **Правка** → **Настройки программы**, в появившемся окне **Настройки программы** открыть вкладку **Умолчания** и выбрать нужную размерность из открывающегося списка **Единица измерения**.

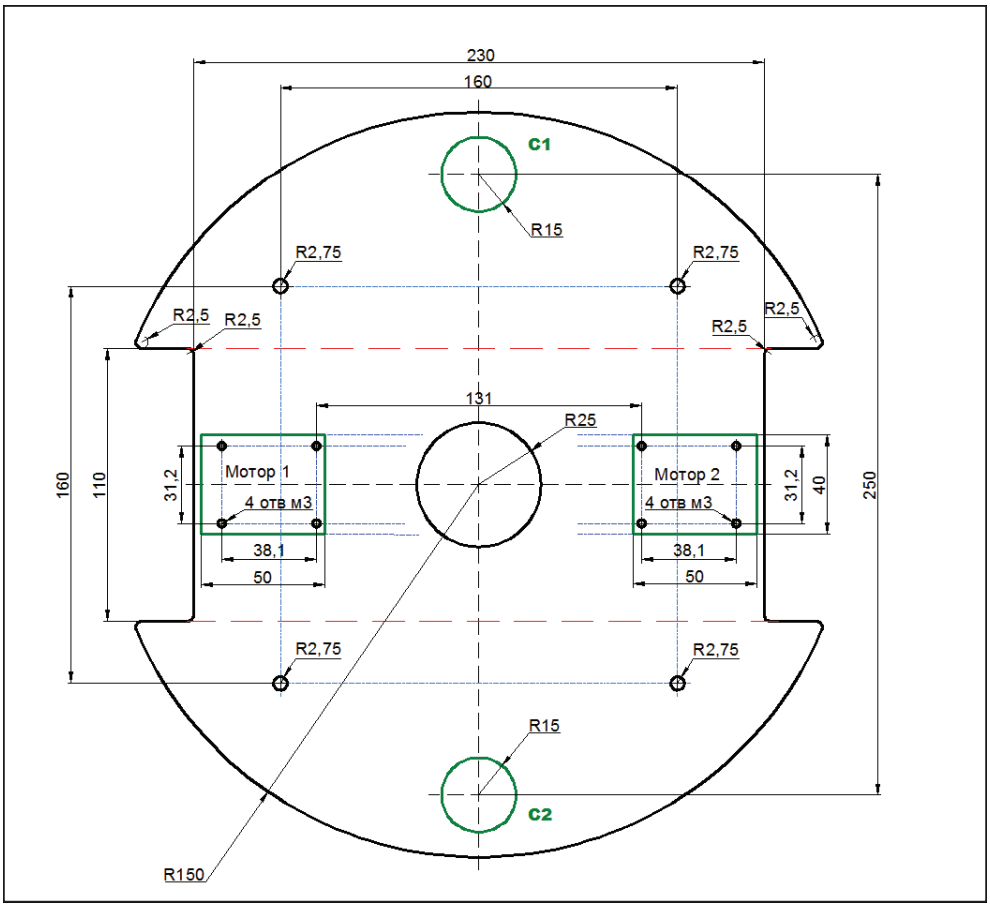


Вкладка **Умолчания** окна **Настройки программы**

Механическая часть робота состоит из трех плит: нижней опорной плиты, или основания конструкции, средней и верхней плит. Плиты соединены друг с другом через опорные стойки, по 4 стойки на ярус. К нижней стороне опорной плиты крепятся 2 кронштейна для моторов и 2 опорных колеса. С верхней стороны устанавливаются аккумулятор и панель управления. Начнем конструирование робота с опорной плиты.

Конструкция опорной плиты робота

Ниже представлен чертеж опорной плиты робота. Плита изготавливается из алюминиевого листа марки АТ (алюминий твердый) толщиной 5 мм. Все размеры, как линейные, так и радиусные, базируются от центра опорной плиты. Как уже упоминалось выше, двигатели для вращения колес крепятся к нижней стороне плиты с помощью двух кронштейнов, конструкция которых будет представлена ниже. Эти двигатели обозначены на чертеже двумя прямоугольниками темно-зеленого цвета и именуются Мотор 1 и Мотор 2. Для крепления каждого кронштейна предусмотрено по 4 отверстия с резьбой М3. Нестандартные расстояния между отверстиями крепления кронштейна (с десятичными значениями) объясняются тем, что эти кронштейны разрабатывались в дюймах (1 дюйм равен 25,4 мм).



Чертеж опорной плиты робота

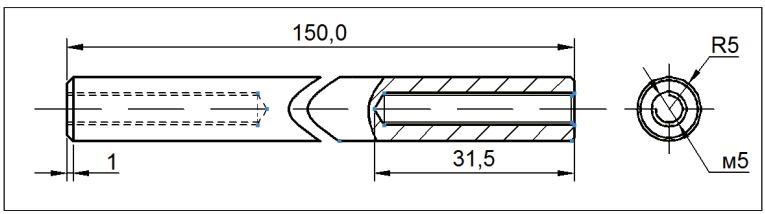
Под прямым углом к воображаемой оси ведущих колес, спереди и сзади платформы, установлены 2 опорных колеса C1 и C2. Расстояние от центра основания до центра опорного колеса составляет 125 мм. Отверстия для крепления опорных колес не обозначены, т. к. это зависит от конструкции узла. Опорное колесо может крепиться с помощью болта (в этом случае в центре, предусмотренном для крепления колеса, придется сверлить отверстие диаметром немного больше диаметра крепежного болта). Также опорное колесо может иметь опорную площадку. В этом случае согласно характеристикам этого узла придется предусмотреть 4 резьбовых отверстия для каждого опорного колеса. Кроме того, с учетом характеристик приобретенного узла (общей высоты) необходимо будет сделать прокладки, компенсирующие разницу дорожного просвета робота и фактической общей высоты поворотного узла. Об опорных колесах мы еще поговорим.

С другой стороны опорной плиты установлены 4 стойки, на которые устанавливается средняя плита. Для крепления этих стоек предусмотрены 4 сквозных отверстия диаметром 5,5 мм. В конструкции для крепления стоек используются винты М5 × 25. В центре плиты просверлено отверстие диаметром 50 мм, через которое будут выведены провода от двигателей. На левой и правой сторонах плиты предусмотрены 2 выреза под ведущие колеса. Радиусы в углах вырезов – технологические, зависящие от диаметра фрезы, которой эти вырезы будут фрезероваться. Радиусы на соединении выреза под колесо и внешнего диаметра плиты предусмотрены, чтобы не было острых углов. Эти углы можно закруглить напильником.

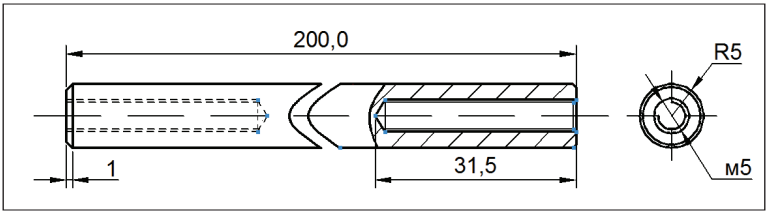
Конструкция нижней и верхней стоек

Как уже упоминалось ранее, средняя плита с опорной плитой соединяется с помощью четырех стоек. Таким же способом мы соединим среднюю и верхнюю плиты. Таким образом, будет собран пакет из опорной, средней и верхней плит со стойками между ними. Мы предлагаем изготовить стойки из стандартного алюминиевого прута (алюминий марки АТ) диаметром 10 мм. Длина нижних стоек, соединяющих основание со средней плитой, равна 150 мм. Длина верхних стоек, устанавливаемых между средней и верхней плитой, – 200 мм. В торцах каждой стойки следует сделать глухие резьбовые отверстия М5 глубиной 25–30 мм. Нижние и верхние стойки (и средняя плита между ними) соединяются с помощью четырех стальных резьбовых стоек М5 × 40. Их можно изготовить из стандартных винтов М6 × 50. Для этого нужно отпилить шляпки винтов и обработать срезы напильником. Ниже приведен чертеж верхней и нижней стоек и резьбовой шпильки.

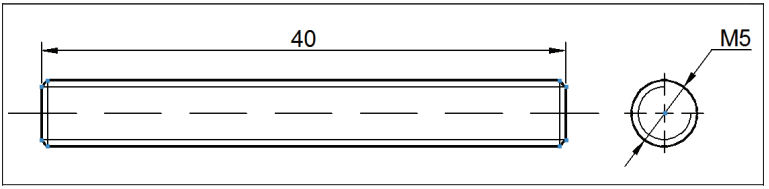
Впрочем, стойки можно изготовить из трубки диаметром 10 мм. Но в этом случае понадобятся 4 длинные шпильки диаметром 6 мм и длиной 370 мм с резьбой М5 × 10 на ее концах (длина шпильки рассчитывалась с учетом высоты всего пакета и запаса под гайки) и гайки.



Чертеж нижней стойки



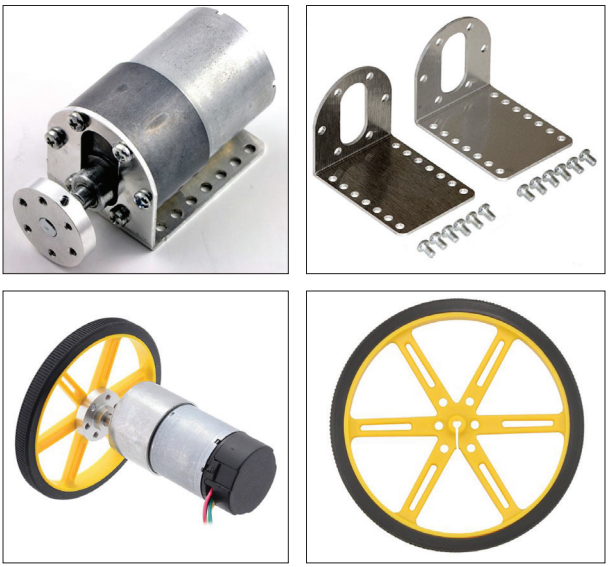
Чертеж верхней стойки



Чертеж шпильки с резьбой

Конструкция колеса и крепления для колеса и мотора

Двигатель, кронштейн и колесо – это комплектующие, которые необходимо приобрести. Заказать их можно на сайте www.pololu.com. Ниже показаны фотографии двигателя в сборке с кронштейном, кронштейна, колеса, установленного на валу двигателя, и самого колеса.

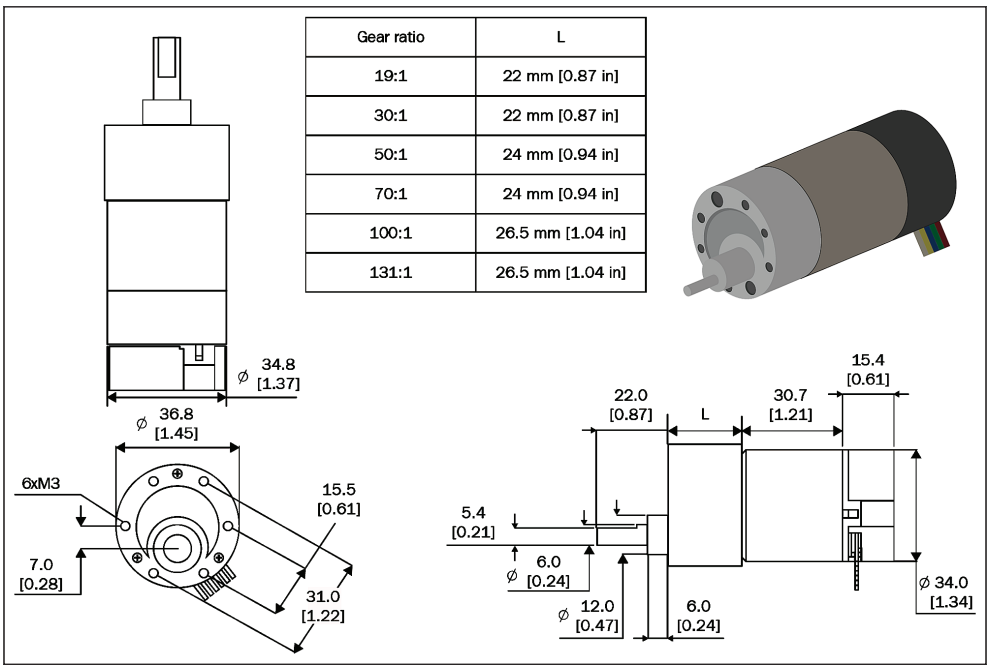


Покупные комплектующие

При расчете количества оборотов и крутящего момента двигателя мы приняли, что диаметр колеса равен 90 мм. Ниже приводятся габаритный чертеж двигателя и таблица с передаточными числами и размерами редукторов, которыми может быть укомплектован двигатель.

Напомним полученные ранее расчетные значения для применяемого в конструкции двигателя:

- количество оборотов = 80 об/мин (приведено к стандартному значению);
- крутящий момент = 18,0 кг/см (приведено к стандартному значению);
- диаметр колеса = 90 мм.



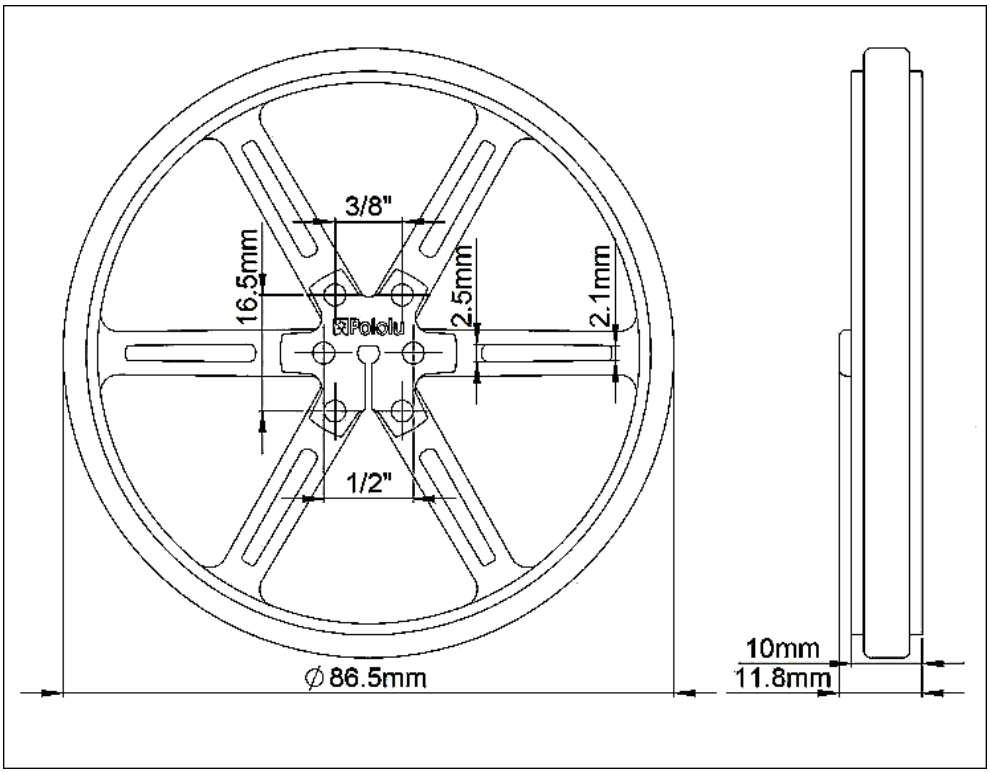
Чертеж двигателя для робота

Конструкция может варьироваться в зависимости от того, какой двигатель будет выбран. Главные требования к выбору двигателя заключаются в том, чтобы его крутящий момент был не меньше расчетного и количество оборотов на ведущем валу тоже приближалось к расчетному. При моделировании мы протестируем, подходит ли этот двигатель к данной конструкции или нужно подбирать другой. Кроме вышеперечисленных характеристик, очень большое значение имеют такие характеристики, как габариты двигателя, способ его крепления и крепежные размеры. В левом нижнем углу рисунка вы увидите крепежные размеры выбранного двигателя. В правом верхнем углу рисунка находится таблица с характеристиками редуктора, которым может быть осна-

щен двигатель. В левой колонке Gear ratio (передаточное число) показаны передаточные числа редукторов. В зависимости от передаточного числа меняется и размер самого узла. На чертеже двигателя и в таблице длина редуктора обозначена латинской буквой L.

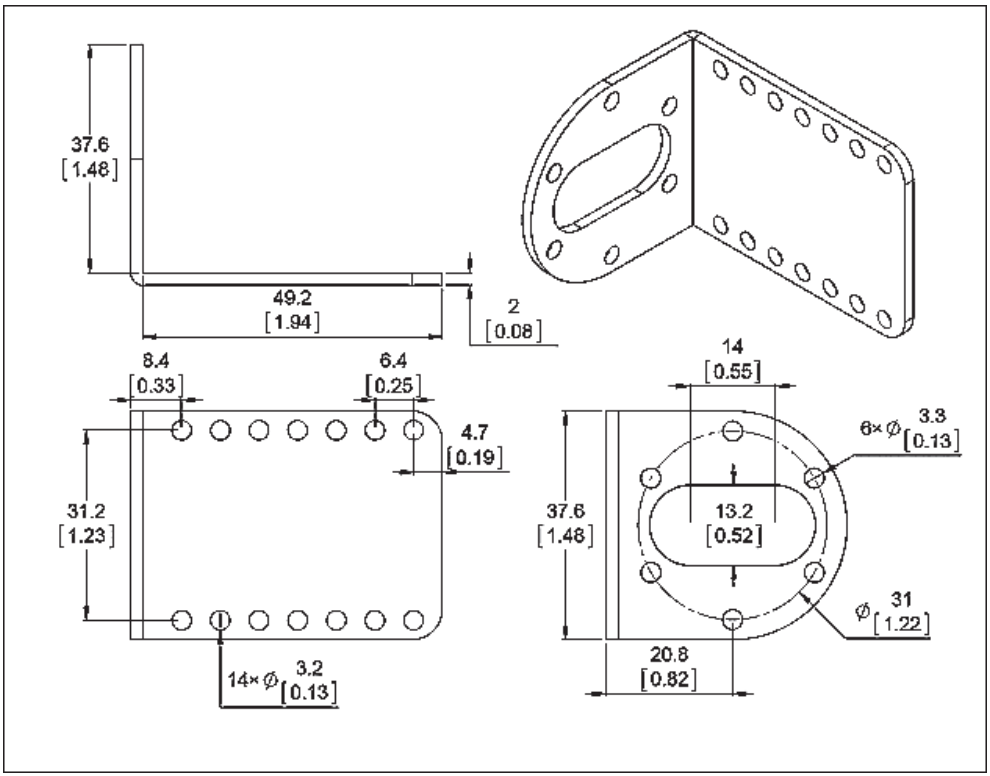
Двигатель, примененный в конструкции, вы можете заказать, зайдя по адресу: https://www.pololu.com/product/2827.

На следующем рисунке показано стандартное колесо диаметром 90 мм, которое предлагается использовать в конструкции. Колесо имеет 6 дополнительных монтажных отверстий для крепления универсальных ступиц. Колеса продаются парами. Ознакомиться и заказать пару колес можно по адресу: https://www.pololu.com/product/1437.



Чертеж колеса для робота

Двигатели устанавливаются на нижней стороне опорной плиты робота с помощью кронштейнов. На следующем рисунке показан чертеж такого кронштейна. Это тоже покупное изделие, и заказать его можно на странице, расположенной по адресу: https://www.pololu.com/product/1084.



Типичная конструкция кронштейна робота

Конструкция опорного колеса

Следующее покупное изделие – это опорные колеса. Опорные колеса могут быть разных конструкций: поворотное колесо со смещенной осью или устройство с опорным шариком (шариковая опора). Устройство с опорным шариком по конструкции похоже на пишущий узел шариковой авторучки. Во всех опорных колесах, кроме грузоподъемности, важным параметром является его высота. Каждый узел имеет свои плюсы и минусы. Шариковое опорное колесо хорошо работает только на ровной поверхности. Если на полу будет лежать, например, ковер, для робота с шариковыми опорами это будет непреодолимым препятствием. А обычное поворотное колесо на крутой разворот робота может среагировать с небольшой задержкой, но хорошо проходит препятствия в виде трещин на полу или ковра. Конструкция двигателя и кронштейна позволяет выбрать необходимый дорожный просвет. Ведущий вал двигателя смещен на 6,86 мм (можно округлить до 7 мм) по отношению к центру двигателя. Поэтому с учетом конструкции двигателя, кронштейна и размера колеса существуют два варианта установки двигателя на кронштейне. Если установить двигатель

ведущим валом ближе к нижней плоскости опорной плиты, расстояние от низа опорной плиты до поверхности, по которой робот будет передвигаться, составляет 59 мм. Если же двигатель установить ведущим валом ближе к полу, расстояние от пола до нижней плоскости опорной плиты составит 62,5 мм. Вариант установки двигателя зависит от условий, в которых будет работать устройство, и от применяемых опорных колес. Если поверхность, по которой предстоит передвигаться роботу, будет ровной, дорожный просвет лучше уменьшить и применить шариковые опоры. Коллекцию шариковых опор мы можем найти, перейдя по ссылке: http://www.pololu.com/category/45/pololu-ball-casters.

Если же на поверхности будут препятствия, лучше применить поворотное колесо, например такое, как показано на рисунке. Высота этого узла – 59 мм, диаметр/ширина колеса – 40/18 мм, смещение оси – 24 мм, размер платформы – 42 × 42 мм, межцентровое расстояние крепежных отверстий – 31 × 31 мм. Ниже приведена ссылка, по которой это колесо можно найти в интернете: https://stella-tech.ru/catalog/kolesa/proizvedeno-v-italii/apparatnyie-kolesa/37-seriya/kolesa-apparatnye-37/povorotnoe-pod-4-bolta-37/koleso-seroe-374100/.

Шариковая опора, показанная на правом рисунке, имеет максимальную высоту 35 мм. Таким образом, при дорожном просвете 59 мм устанавливать колесо придется через прокладку толщиной 24 мм. Данную шариковую опору можно купить, зайдя на страницу по адресу: https://www.pololu.com/product/66.

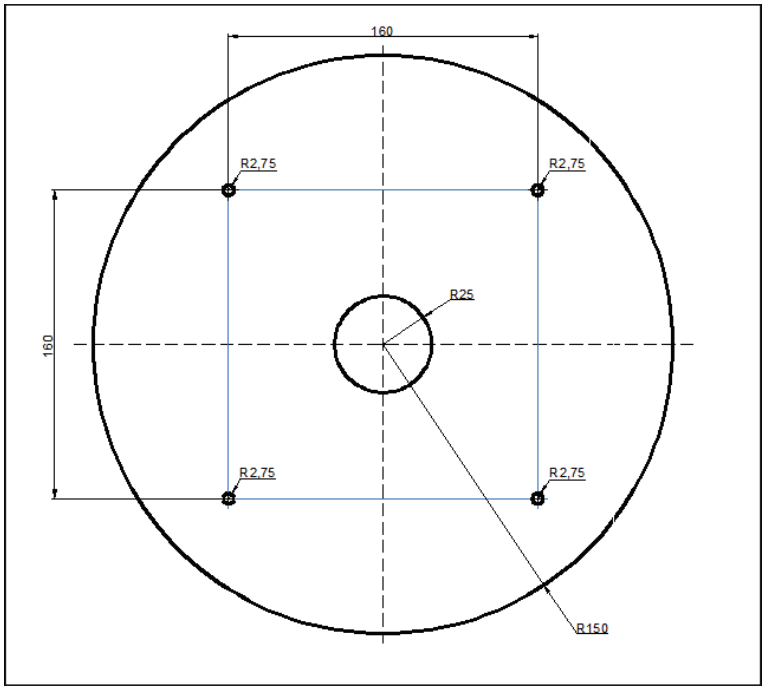
Колесо поворотное

Шариковая опора

Конструкция средней плиты

Внешний диаметр средней плиты равен внешнему диаметру опорной плиты робота. Остаются на своих местах центральное отверстие под провода и крепежные отверстия. Толщина плиты – 2 мм. Она соединяется с опорной и верх-

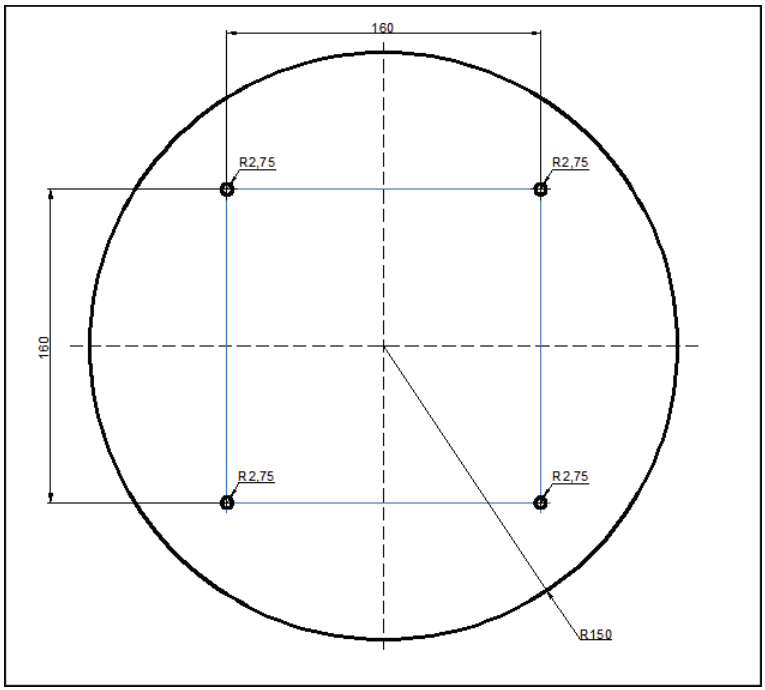
ней плитами посредством стоек, чертежи которых были приведены выше. В стойки со стороны средней плиты вворачиваются стальные шпильки (чертеж шпильки приведен выше), дальше на них через крепежные отверстия насаживается средняя плита и зажимается верхними стойками, которые наворачиваются на шпильки. Среднюю плиту предлагаем изготовить из твердого листового алюминия марки АТ.



Чертеж средней плиты

Конструкция верхней плиты

Конструктивно верхняя плита похожа на среднюю плиту. Только отсутствует центральное отверстие. Крепится верхняя плита к стойкам винтами M5 × 15 через крепежные отверстия. Если с верхней стороны плиты крепежные отверстия зенкеровать, можно применить винты с потайной головкой. В этом случае шляпки винтов не будут выступать над поверхностью плиты, благодаря чему она будет ровной, без выступов.



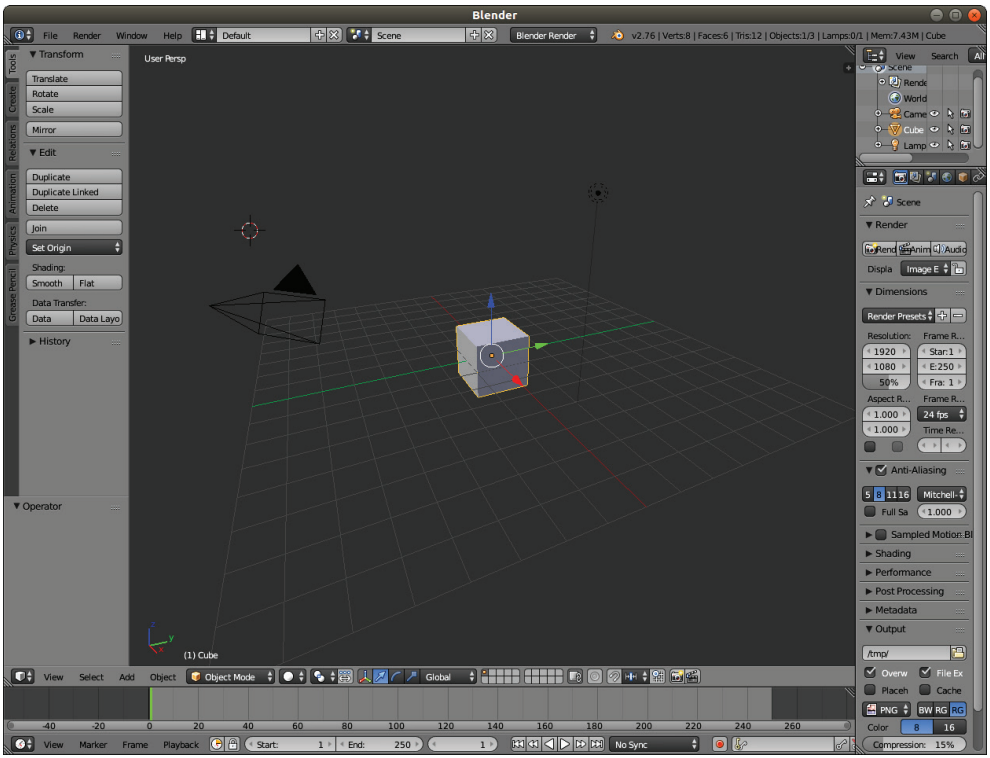
Чертеж верхней плиты

Итак, чертежи всех механических деталей робота созданы. Теперь нам ничего не мешает с помощью программы Blender создать 3D-модель нашего робота. Трехмерная модель будет создана из двухмерных чертежей, необходимых для изготовления деталей на производстве.

Работа с 3D-моделью робота с использованием Blender

В этом разделе мы создадим 3D-модель робота. В основном она будет использоваться для моделирования устройства с помощью программы Blender. Версии данной программы должны быть не меньше чем 2.6, т. к. все чертежи были проверены на них.

На показанном ниже снимке экрана вы увидите окно этой программы с рабочей областью и инструментами, предназначенными для работы с 3D-моделями:



Окно программы Blender

Основная причина использования Blender заключается в том, что мы можем создать модель робота, написав скрипты (программы) на языке программирования Python. Blender имеет встроенный интерпретатор Python и текстовый редактор Python для программирования. Мы не будем рассматривать пользовательский интерфейс Blender. Хороший учебник по Blender можно найти на официальном сайте. Чтобы подробнее узнать о пользовательском интерфейсе программы, перейдите по ссылке: http://www.blender.org/support/tutorials/.

Русскоязычный читатель может воспользоваться следующей ссылкой: http://blender3d.org.ua/book.

А теперь настало время программировать в Blender с помощью Python.

Скрипты Python в Blender

В основном Blender написан на языках C, C++ и Python. Пользователи могут писать свои собственные скрипты на Python и получить доступ ко всем функциональным возможностям Blender. Если вы хорошо знакомы с Blender API Python, то можете вместо ручного моделирования создать модель робота с помощью скриптов Python.

Blender использует Python 3.x. Хоть Python API в целом стабилен, он до сих пор частично перерабатывается и улучшается. Документация по Blender API Python находится по адресу: http://www.blender.org/documentation/blender_python_api_2_69_7/.

Кратко рассмотрим интерфейсы Blender Python API Blender, которые будут использованы при создании модели робота.

Введение в API Blender Python

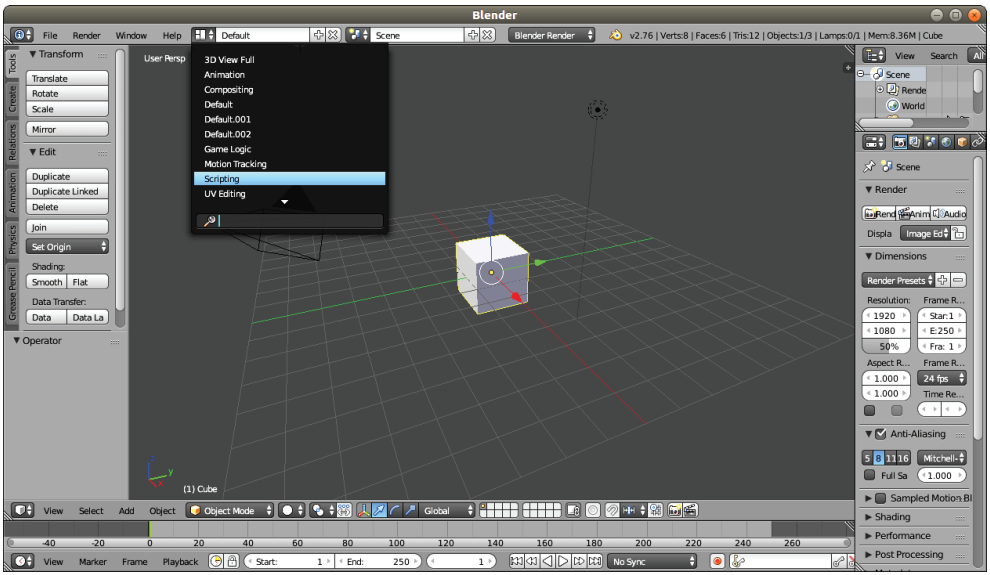
API Python в Blender может использовать большинство функций Blender. Ниже перечислены основные задачи, которые может выполнять API:

- редактирование внутри Blender любых данных, таких как сцены, сетки, точки и т. д.;
- изменение настроек пользователя, горячих клавиш и тем интерфейса Blender;
- создание новых инструментов Blender;
- рисование 3D-изображения с помощью имеющихся в Python команд OpenGL.

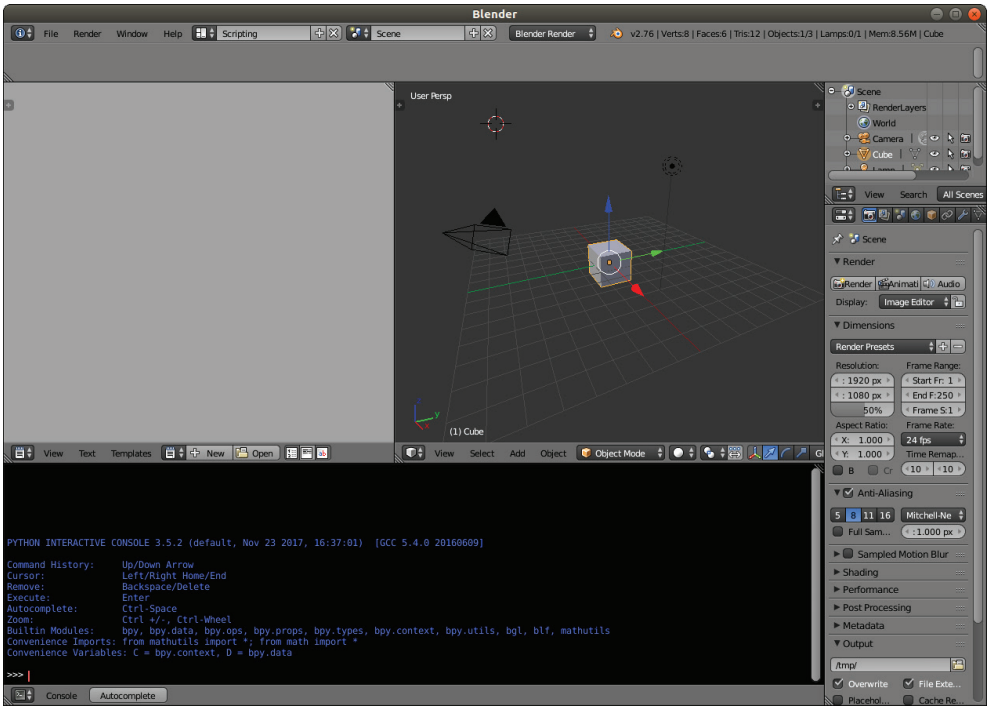
Blender предоставляет интерпретатору Python модуль **bpy**. Этот модуль может импортироваться в скрипт и предоставлять доступ к данным, классам и функциям Blender. Его модуль нужно импортировать скриптам, которые работают с данными Blender. Ниже приведен список основных модулей Python, которые мы будем использовать в bpy:

- **контекстный доступ**. Скрипт `bpy.context` – обеспечение доступа к функциям пользовательского интерфейса Blender;
- **доступ к данным**. Скрипт `bpy.data` – обеспечение доступа к внутренним данным Blender;
- **операторы**. Скрипт `bpy.ops` – обеспечение доступа Python к вызывающим операторам, включая написанные на C, Python или макросы.

Для написания скрипта в Blender необходимо открыть вкладку **Scripting**. Для этого щелкните мышью на кнопке в левой части открывающегося списка выбора готовых рабочих пространств и выберите строку **scripting**. Вид рабочего окна программы изменится, появятся текстовый редактор и консольное окно Python в Blender.



Выбор рабочего пространства **Scripting**



Вкладка **Scripting** открыта

Как уже говорилось ранее, после выбора вкладки **Scripting** мы увидим текстовый редактор и консольное окно Python в Blender. В текстовом редакторе мы можем писать коды с помощью API Blender, а также проверять команды Python через консоль.

Создайте новый скрипт на Python и назовите его `robot.py`. Для этого нажмите в нижней части текстового редактора кнопку **New** (Создать), введите в поле ввода, расположенное правее кнопки **New**, имя скрипта `robot.py` и нажмите клавишу **Enter**. Скрипт создан. Теперь данный скрипт нужно сохранить. Выберите команду меню **File** → **Save As** (Сохранить как), папку сохранения, введите имя, под которым файл будет сохранен, и нажмите кнопку **Save As Blender File** (Сохранить как файл Blender). Теперь, используя только Python-скрипты, мы будем создавать 3D-модель робота. Следующий раздел содержит полный скрипт для создания такой модели. Перед запуском желательно изучить код. Мы надеемся, что вы прочитали инструкции по API Python Blender на официальном сайте. Код в следующем разделе разделен на 6 Python-функций. Это сделано для того, чтобы нарисовать 3 плиты робота, двигатели, колеса и сделать 4 стойки. После запуска мы экспортируем скрипт в формат **STereoLithography** (STL) – 3D-формат файла для моделирования.

Скрипт Python модели робота

Ниже приведен скрипт Python разрабатываемой нами модели робота. Знак решетки # обозначает, что отмеченная этим знаком строка является комментарием и в выполнении скрипта участия не принимает. Строка из множества значков ####### служит для визуального разделения кода на разделы и больше никакой смысловой нагрузки не несет. Данные строки, как и строки с комментариями, можно и не вводить. Это не повлияет на работу скрипта.

1. Перед запуском скрипта Python в Blender необходимо импортировать модуль `bpy`. Модуль `bpy` содержит все функциональные возможности Blender и получает доступ в самом приложении Blender. Щелкните мышью в окне текстового редактора и введите с клавиатуры первую команду:

```
import bpy
```

2. Следующая функция предназначена для создания опорной плиты робота. С ее помощью будет нарисован цилиндр радиусом 15 см:

```
#Создание опорной плиты:
def Draw_Base_Plate():

##################################################
bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.09))

# Изменение рабочего пространства
bpy.ops.object.select_pattern(pattern="Cube")
bpy.data.objects['Cylinder'].select = False
bpy.ops.object.delete(use_global=False)
```

3. С помощью следующей функции мы нарисуем моторы и колеса, установленные на опорной плите:

```
#Функция для создания двигателей и колес
def Draw_Motors_Wheels():
```

4. Показанные ниже команды нарисуют цилиндр радиусом 0,045 и толщиной 0,01 м. Этот цилиндр будет обозначать колесо. После создания колесо будет перемещено на свое место:

```
#Создать первое колесо
bpy.ops.mesh.primitive_cylinder_add(radius=0.045, depth=0.01, location=(0,0,0.07))
#Поворот
bpy.context.object.rotation_euler[1] = 1.5708
#Перемещение
bpy.context.object.location[0] = 0.135

#Создать второе колесо
bpy.ops.mesh.primitive_cylinder_add(radius=0.045, depth=0.01, location=(0,0,0.07))
#Поворот
bpy.context.object.rotation_euler[1] = 1.5708
#Перемещение
bpy.context.object.location[0] = -0.135
```

5. Следующие команды добавят к опорной плите 2 макета двигателей. Размеры двигателей указаны в 2D-чертеже опорной плиты. Модель мотора будет выполнена в виде цилиндра, который после поворота соединится с опорной плитой:

```
# Добавление двигателей
bpy.ops.mesh.primitive_cylinder_add(radius=0.018, depth=0.06,
location=(0.075,0,0.075))
bpy.context.object.rotation_euler[1] = 1.5708
bpy.ops.mesh.primitive_cylinder_add(radius=0.018, depth=0.06,
location=(-0.075,0,0.075))
bpy.context.object.rotation_euler[1] = 1.5708
```

6. Чтобы модели двигателей были похожи на моторы, добавим валы. Вал будет выглядеть как соединенный с двигателем цилиндр:

```
bpy.ops.mesh.primitive_cylinder_add(radius=0.006,
depth=0.04, location=(0.12,0,0.075))

bpy.context.object.rotation_euler[1] = 1.5708
bpy.ops.mesh.primitive_cylinder_add(radius=0.006,
depth=0.04, location=(-0.12,0,0.075))
bpy.context.object.rotation_euler[1] = 1.5708
#########################################################
```

7. Добавим к основанию 2 опорных колеса и изобразим их в виде 2 цилиндров:

```
#Добавление опорного колеса
bpy.ops.mesh.primitive_cylinder_add(radius=0.015, depth=0.05,
location=(0,0.125,0.065))
bpy.ops.mesh.primitive_cylinder_add(radius=0.015, depth=0.05,
location=(0,-0.125,0.065))
```

8. Следующая команда добавит макет датчика Kinect:

```
#Добавление Kinect
bpy.ops.mesh.primitive_cube_add(radius=0.04, location=(0,0,0.26))
```

9. Теперь нарисуем среднюю плиту робота:

```
#Нарисовать среднюю плиту
def Draw_Middle_Plate():
bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.22))

# Нарисовать верхнюю плиту
def Draw_Top_Plate():
bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.37))
################################################################
```

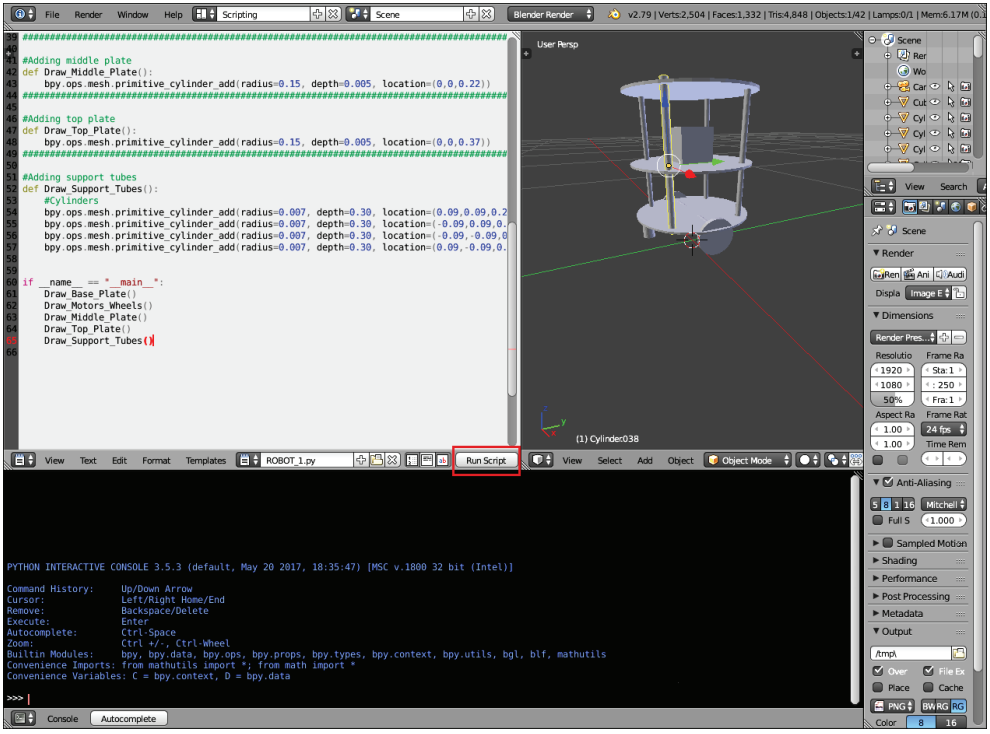
10. Следующие команды нарисуют все стойки для всех 3 пластин:

```
# Добавление стоек
def Draw_Support_Tubes():

# Трубки
bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30,
location=(0.09,0.09,0.23))
bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30,
location=(-0.09,0.09,0.23))
bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30,
location=(-0.09,-0.09,0.23))
bpy.ops.mesh.primitive_cylinder_add(radius=0.007,depth=0.30,
location=(0.09,-0.09,0.23))
```

11. Теперь добавим основной код, без которого наш скрипт не будет работать.

```
# Основной код
if __name__ == "__main__":
Draw_Base_Plate()
Draw_Motors_Wheels()
Draw_Middle_Plate()
Draw_Top_Plate()
Draw_Support_Tubes()
```

12. После ввода кода в текстовый редактор выполните сценарий, нажав кнопку **Run Script** (Запустить скрипт) или комбинацию клавиш **Alt**+**P**. Кнопка **Run Script** на рисунке обведена красной рамкой. Если скрипт не содержит ошибок, 3D-модель отобразится в окне **3D-вид** Blender.

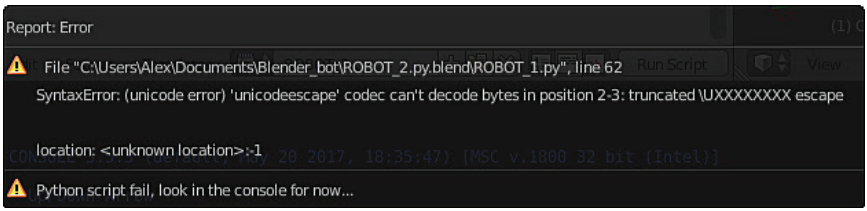


Запуск скрипта в Blender

```
import bpy

def Draw_Base_Plate():
    #Create Base Plate
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.09))
################################################################################
    #Create Working Space
    bpy.ops.object.select_pattern(pattern="Cube")
    bpy.data.objects['Cylinder'].select = False
    bpy.ops.object.delete(use_global=False)
################################################################################

def Draw_Motors_Wheels():
    #Create first wheel
    bpy.ops.mesh.primitive_cylinder_add(radius=0.045, depth=0.01, location=(0,0,0.07))
    #Rotate
    bpy.context.object.rotation_euler[1] = 1.5708
    #Transalation
    bpy.context.object.location[0] = 0.135
    #Create second wheel
    bpy.ops.mesh.primitive_cylinder_add(radius=0.045, depth=0.01, location=(0,0,0.07))
    bpy.context.object.rotation_euler[1] = 1.5708
    bpy.context.object.location[0] = -0.135
    #Adding motor
    bpy.ops.mesh.primitive_cylinder_add(radius=0.018, depth=0.06, location=(0.075,0,0.075))
    bpy.context.object.rotation_euler[1] = 1.5708
    bpy.ops.mesh.primitive_cylinder_add(radius=0.018, depth=0.06, location=(-0.075,0,0.075))
    bpy.context.object.rotation_euler[1] = 1.5708
    #Adding motor shaft
    bpy.ops.mesh.primitive_cylinder_add(radius=0.006, depth=0.04, location=(0.12,0,0.075))
    bpy.context.object.rotation_euler[1] = 1.5708
    bpy.ops.mesh.primitive_cylinder_add(radius=0.006, depth=0.04, location=(-0.12,0,0.075))
    bpy.context.object.rotation_euler[1] = 1.5708
    #Adding caster wheel
    bpy.ops.mesh.primitive_cylinder_add(radius=0.015, depth=0.05, location=(0,0.125,0.065))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.015, depth=0.05, location=(0,-0.125,0.065))

    #Adding Kinect
    bpy.ops.mesh.primitive_cube_add(radius=0.04, location=(0,0,0.26))
################################################################################

#Adding middle plate
def Draw_Middle_Plate():
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.22))
################################################################################

#Adding top plate
def Draw_Top_Plate():
    bpy.ops.mesh.primitive_cylinder_add(radius=0.15, depth=0.005, location=(0,0,0.37))
################################################################################

#Adding support tubes
def Draw_Support_Tubes():
    #Cylinders
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007, depth=0.30, location=(0.09,0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007, depth=0.30, location=(-0.09,0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007, depth=0.30, location=(-0.09,-0.09,0.23))
    bpy.ops.mesh.primitive_cylinder_add(radius=0.007, depth=0.30, location=(0.09,-0.09,0.23))
################################################################################


if __name__ == "__main__":
    Draw_Base_Plate()
    Draw_Motors_Wheels()
    Draw_Middle_Plate()
    Draw_Top_Plate()
    Draw_Support_Tubes()

```

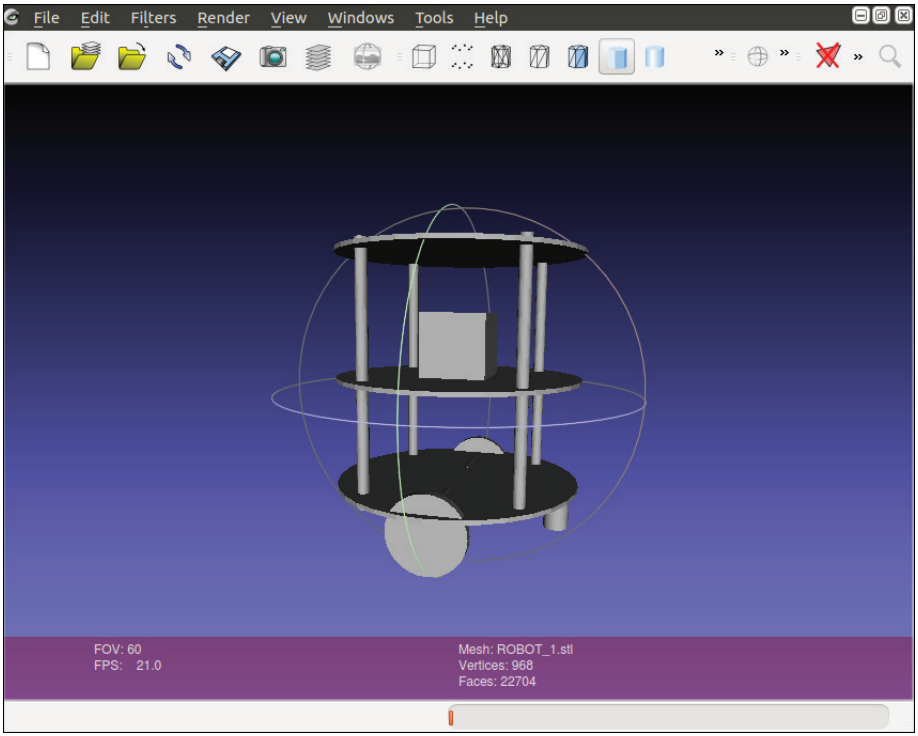
Скрипт, запущенный в Blender

Если же в скрипте будет обнаружена ошибка, на экране появится сообщение с номером строки, в которой она содержится.



Сообщение об обнаруженной ошибке

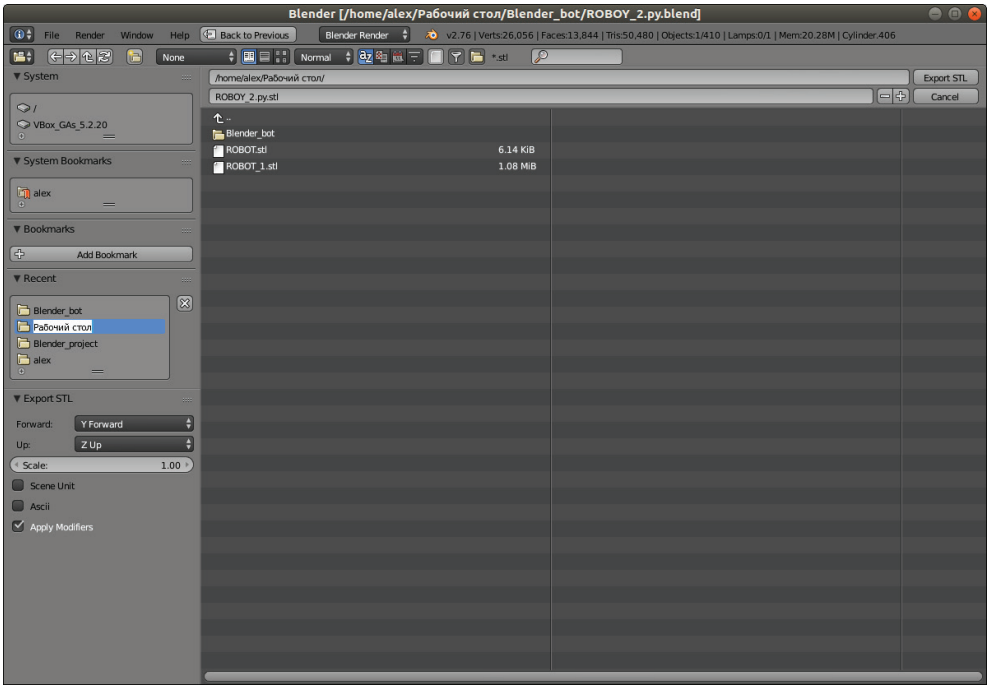
Теперь данный скрипт нужно, во-первых, сохранить, а во-вторых, экспортировать в формат `*.stl`. Файл, сохраненный в данном формате, может быть открыт в программе MeshLab.



Модель в программе MeshLab

Для сохранения скрипта выберите команду меню **File** → **Save** или нажмите комбинацию клавиш **Ctrl**+**S**.

Чтобы экспортировать файл в формат `*.stl`, выберите команду меню **File → Export → Stl(.stl)**. На экране появится окно экспорта файла программы Blender.



Выбор параметров экспорта файла в формат *.stl

В верхнем поле ввода показан путь к целевой папке. Чтобы выбрать другую папку, в которой требуется сохранить экспортируемый файл, достаточно в списке папок, расположенном в большом окне ниже поля ввода пути сохранения и имени файла, щелкнуть на нужной папке. Для открытия корневого списка папок щелкните мышью на значке в первой строке списка папок. Если целевая папка недавно открывалась, выберите ее в списке недавно открывавшихся папок **Recent**. Чтобы впоследствии файл было легче найти, сохраните его на рабочем столе. Далее введите в поле ввода имени файла, расположенного ниже поля ввода пути сохранения, имя, под которым файл будет сохранен. Затем следует выбрать параметры экспорта файла. Это делается в группе элементов управления **Export STL**, расположенной в левом нижнем углу окна. Из открывающегося списка **Forward** выберите строку **Y Forward**, а из открывающегося списка **Up** – строку **Z Up**. Впрочем, эти параметры выбираются по умолчанию. Флажки **Selection Only**, **Scene Unit**, **Ascii** должны быть сброшены. Если флажок **Apply Modifiers** сброшен, установите его, а в списке значения масштаба **Scale** выберите значение **1.00**. После того как параметры экспорта

определены, нажмите кнопку **Export STL**, расположенную в правом верхнем углу. Модель создана, файл сохранен и экспортирован в нужный нам формат.

Создание модели URDF-робота

Модель робота ROS содержит пакеты для моделирования различных аспектов, указанных в формате описания робота XML Robot Description Format. Базовый пакет – это стек URDF, который анализирует файлы URDF и создает объектную модель.

Unified Robot Description Format (Унифицированный формат описания робота URDF) – это XML-спецификация для описания модели робота. С помощью URDF можно представить следующие возможности устойства:

- кинематическое и динамическое описание робота;
- визуальное представление робота;
- моделирование физических процессов робота.

Судя по описанию, робот состоит из совокупности взаимодействующих элементов, соединенных набором связей. Приведенный ниже код демонстрирует типичное описание робота:

```
<robot name="chefbot">
<link> ... </link>
<link> ... </link>
<link> ... </link>

<joint> .... </joint>
<joint> .... </joint>
<joint> .... </joint>
</robot>
```

Для получения дополнительной информации об URDF можно обратиться к следующим ссылкам:
http://wiki.ros.org/urdf,
http://wiki.ros.org/urdf/Tutorials.

Xacro (XML-макросы) – язык макросов формата XML. С xacro мы можем создать короткие, хорошо читаемые XML-файлы. Для упрощения файла URDF мы можем использовать xacro вместе с URDF. При этом следует вызвать дополнительную программу – анализатор, который преобразует xacro в URDF.

Чтобы больше узнать, что такое xacro, перейдите по ссылке: http://wiki.ros.org/xacro.

Одним из основных пакетов в ROS является пакет `tf`, с помощью которого `robot_state_publisher` позволяет публиковать состояние робота (http://wiki.ros.org/tf). Как только публикация подтверждается, она становится доступной для всех компонентов системы, использующих `tf`.

Этот узел читает параметр вызова URDF, называемый `robot_description` (описание робота), в теме `joint_states` (состояние соединения) считывает совместные углы робота и, используя кинематическую модель в дереве робота,

публикует трехмерную ориентацию связей. Другими словами, пакет в качестве входных данных получает совместное положение частей робота и публикует 3D-положение робота, связывая с помощью кинематической древовидной модели все эти данные. Пакет может быть использован в качестве не только библиотеки, но и узла ROS. Он хорошо протестирован, код стабилен. В ближайшее время никаких крупных изменений в пакете не планируется.

- **World files**. Данные файлы представляют среду Gazebo, которая должна быть загружена наряду с моделью робота. Примерами таких файлов могут служить файлы `empty.world` и `playground.world`. Файл `empty.world` содержит просто пустое место. Файл `playground.world` содержит статические объекты. В Gazebo можно создать собственный `*.world`-файл. Подробнее об этом в следующей главе.
- **CMakeList.txt** и **package.xml**. Эти файлы создаются одновременно с пакетом. Файл `CmakeList.txt` помогает построить узлы ROS C++ или библиотеки пакета. Файл `package.xml` содержит список всех зависимостей этого пакета.

Создание пакета описания ChefBot в ROS

Пакет `chefbot_description` содержит URDF-модель нашего робота. Перед тем как создать этот пакет, можно просмотреть загруженные пакеты Chefbot из `chapter3_codes`. Это ускорит процесс создания файла.

Давайте рассмотрим, как создать пакет `chefbot_description`. Для этого следует выполнить следующие действия:

1) сначала нам нужно перейти в папку `src`:

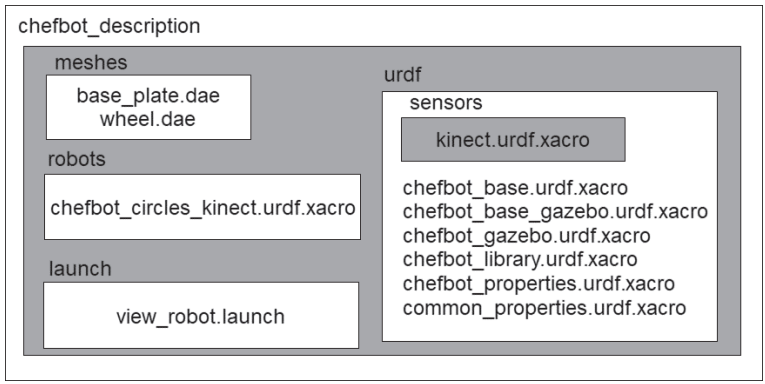
```
$ cd ~/catkin_ws/src;
```

2) следующая команда создаст пакет описания робота и зависимости `urdf` и `xacro`. Также в папке `catkin_ws/src` будет создан пакет `chefbot_description`:

```
$ catkin_create_pkg chefbot_description urdf xacro.
```

Рассмотрим структуру пакета `chefbot_description`. Папка meshes содержит трехмерные детали робота, а в папке `urdf` хранятся URDF-файлы кинематической и динамической моделей робота. Модель робота разбита на несколько макросов, которые облегчают отладку и улучшают читаемость.

Давайте посмотрим на функциональные возможности каждого файла, находящегося внутри этого пакета. Вы можете проверить каждый из файлов внутри `chefbot_description`. На следующей схеме показаны файлы внутри этого пакета:



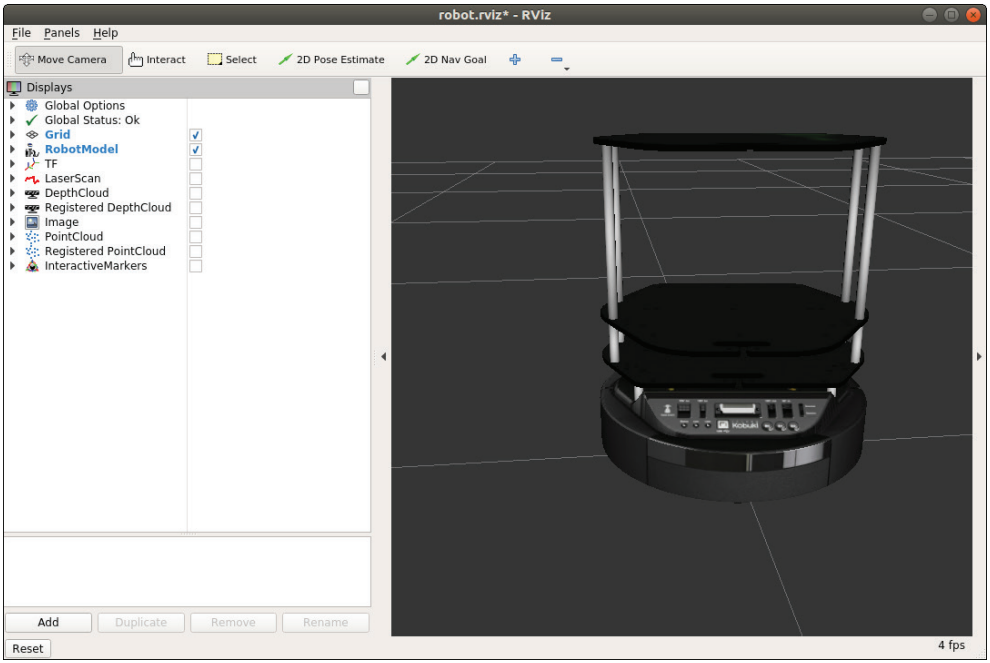
Структура пакета `shefbot_description`

- `urdf/chefbot.xacro`: это основной файл `xacro`, содержащий кинематическое и динамическое описания робота;
- `urdf/common_properties.xacro`: данный файл `xacro` содержит некоторые свойства и их значения, используемые в самой модели робота. Это такие значения, как, например, различные цветовые определения ссылок робота и некоторые константы;
- `gazebo/chefbot.gazebo.xacro`: этот файл содержит параметры моделирования робота. В основном здесь находятся параметры Gazebo и плагины для создания модели, которые будут активны только при запуске моделирования данной модели робота;
- `launch/upload_model.launch`: данный файл запуска имеет узел, предназначенный в основном для анализа файла `xacro`. Файл загружает проанализированные данные в параметр ROS. Этот параметр называется `robot_description`. Далее `robot_description` используется в Rviz для визуализации и применяется в Gazebo для моделирования робота. Если в модели xacro содержатся ошибки, файл запуска об этом сообщит;
- `launch/view_model.launch`: этот файл предназначен для загрузки модели робота URDF и ее просмотра в Rviz;
- `launch/view_navigation.launch`: показывает модель URDF и связанные с навигацией шрифты в Rviz;
- `launch/view_robot_gazebo.launch`: данный файл запускает в Gazebo не только URDF-модель, но и Gazebo-плагины;
- `meshes/`: эта папка содержит все узлы, необходимые для моделирования робота;
- для создания рабочей области следует воспользоваться командой `catkin_make`.

После того как пакеты будут собраны, воспользуйтесь следующей командой для запуска моделирования робота Chefbot:

```
$ roslaunch chefbot_descriptionview_robot.launch
```

Ниже показан снимок с экрана с моделью робота в Rviz:



URDF-модель Chefbot в Rviz

Файл `view_robot.launch` создает образ робота в Rviz:

```
<launch>

<!-- This launch file will parse the URDF model and create
robot_description parameter - ->

<include file="$(find chefbot_description)/launch/upload_model.launch" />
<!-Publish TF from joint states -- >

<node name="robot_state_publisher" pkg="robot_state_publisher"
type="robot_state_publisher" />

<!-Start slider GUI for controlling the robot joints -- >
<node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" args="_use_gui:=True" />

<!-Start Rviz with a specific configuration -- >

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find
chefbot_description)/rviz/robot.rviz" />

</launch>
```

Определим upload_model.launch. Эта xacro-команда проанализирует файл chefbot.xacro и сохранит его в robot_description:

```
<launch>
<!-- Robot description -->
<param name="robot_description" command="$(find xacro)/xacro --inorder
'$(find chefbot_description)/urdf/chefbot.xacro'" />
</launch>
```

Рассмотрим файл udf/chefbot.xacro. Это основной файл URDF. Мы можем изучить каждый элемент внутри файла xacro.

Следующий фрагмент кода показывает заголовок модели робота xacro. Этот XML-код содержит имя робота и включает в себя некоторые другие файлы xacro, такие как _properties.xacro и chefbot.gazebo.xacro. Далее показаны определенные в заголовке свойства камеры:

```
<?xml version="1.0"?>
<robot name="chefbot" xmlns:xacro="http://ros.org/wiki/xacro">

<xacro:include filename="$(find
chefbot_description)/urdf/common_properties.xacro" />

<xacro:include filename="$(find
chefbot_description)/gazebo/chefbot.gazebo.xacro" />

<xacro:property name="astra_cam_py" value="-0.0125"/>
<xacro:property name="astra_depth_rel_rgb_py" value="0.0250" />
<xacro:property name="astra_cam_rel_rgb_py" value="-0.0125" />
<xacro:property name="astra_dae_display_scale" value="0.8" />
```

В следующем фрагменте кода содержится определение связей и соединений в модели:

```
<link name="base_footprint"/>

<joint name="base_joint" type="fixed">
<origin xyz="0 0 0.0102" rpy="0 0 0" />
<parent link="base_footprint"/>
<child link="base_link" />
</joint>
<link name="base_link">
<visual>
<geometry>
<!-- new mesh -->
<mesh filename="package://chefbot_description/meshes/base_plate.dae" />
<material name="white"/>
</geometry>

   <origin xyz="0.001 0 -0.034" rpy="0 0 ${M_PI/2}"/>
</visual>
<collision>
```

```
<geometry>
<cylinder length="0.10938" radius="0.178"/>
</geometry>
<origin xyz="0.0 0 0.05949" rpy="0 0 0"/>
</collision>
<inertial>
<!-- COM experimentally determined -->
<origin xyz="0.01 0 0"/>
<mass value="2.4"/><!-- 2.4/2.6 kg for small/big battery pack -->

<inertia ixx="0.019995" ixy="0.0" ixz="0.0"
iyy="0.019995" iyz="0.0"
izz="0.03675" />
</inertial>
</link>
```

В этом коде показано определение двух ссылок: `base_footprint` и `base_link`. Ссылка `base_footprint` – это формальная ссылка, т. е. она имеет любые свойства. Ее назначение – показать происхождение робота. `Base_link` является источником робота со свойствами `visual` и `collision`. Эта ссылка отображается как файл с расширением `*mesh`.

Мы также можем увидеть инерционные параметры связи в определении. Совместное соединение двух ссылок. В URDF имеются разные виды соединений: фиксированные, шарнирные, непрерывные и призматические. В этом фрагменте кода будет создано фиксированное соединение.

В данной главе мы рассмотрели основы Chefbot URDF. Подробнее о моделировании Chefbot будет рассказано в следующей главе.

Итоги

В этой главе мы обсудили, как создать модель робота Chefbot. Моделирование включает в себя создание двухмерных чертежей и проектирование деталей робота в 3D. Далее создается URDF-модель, которая впоследствии используется в ROS.

В начале главы были определены технические требования, которым должен соответствовать робот, и произведены расчеты параметров. Затем, основываясь на технических требованиях и расчетных параметрах, мы приступили к разработке двухмерных чертежей деталей конструкции робота. Проектирование выполнялось в бесплатной системе автоматического проектирования LibreCAD. С помощью скрипта, написанного на языке Python, в программе Blender была создана 3D-модель URDF-робота. После создания URDF-модели мы узнали, как представить робота в Rviz.

В следующей главе обсудим, как смоделировать робота, составить карту среды и локализацию.

Вопросы

1. Что такое модель робота, и зачем она нужна?
2. Зачем создавать двухмерную модель робота?
3. Какова цель создания трехмерной модели робота?
4. Почему лучше создавать модель с помощью скриптов Python, а не использовать ручное моделирование?
5. Что такое URDF-файл, и как его использовать?

Дополнительная информация

Чтобы узнать больше о URDF, Xacro и Gazebo, обратитесь книге «Освоение ROS для программирования роботов» (второе издание): https://www.packtpub.com/hardware-and-creative/mastering-ros-robotics-programming-second-edition.

Глава 4

Моделирование дифференциального привода робота, управляемого операционной системой ROS

В предыдущей главе мы рассмотрели, как создать модель Chefbot. Эта глава будет посвящена визуализации работы робота с помощью тренажера Gazebo в ROS. Нами будет создана имитационная модель Chefbot. Кроме того, в Gazebo будет создана гостиничная среда, где и будет протестировано приложение, запрограммированное на автоматическую доставку еды клиенту. Также мы предоставим подробное объяснение каждого выполняемого шага. Список тем, рассматриваемых в этой главе, приведен ниже:

- начало работы с симулятором Gazebo;
- работа с симулятором TurtleBot 2;
- создание симуляции Chefbot;
- URDF-теги и плагины для моделирования;
- начало работы с одновременным определением местоположения и составлением карты местности;
- реализация SLAM в среде Gazebo;
- создание карты с помощью SLAM;
- начало работы с адаптивной локализацией Monte Carlo;
- реализация AMCL в среде Gazebo;
- Gazebo: автономная навигация Chefbot в отеле.

Технические условия

Для тестирования приложения и кодов понадобится операционная система Ubuntu 16.04 LTS PC/laptop с установленным Ros Kinetic.

Начало работы с симулятором Gazebo

В первой главе мы ознакомились с основными понятиями симулятора Gazebo и процедурой его установки. В этой главе будет показано, как с помощью симулятора Gazebo сымитировать работу робота с дифференциальным приводом. Сначала мы рассмотрим графический интерфейс пользователя и элементы управления этой программы. Как было сказано ранее, Gazebo состоит из двух разделов: Gazebo-сервер и Gazebo-клиент. Моделирование выполняется на сервере Gazebo, который действует как программно-аппаратная часть сервиса (back-end). Графический интерфейс пользователя является клиентской стороной пользовательского интерфейса к программно-аппаратной части сервиса (front-end) и действует как клиент Gazebo. Мы также рассмотрим Rviz (ROS Visualizer) – инструмент графического интерфейса ROS. ROS Visualizer используется для визуализации различных видов роботов и данных, получаемых от датчиков физического робота или его виртуальной модели, например таких, как Gazebo.

Мы можем использовать Gazebo как имитатор для моделирования робота или использовать интерфейсы ROS и Python для программирования роботов в имитаторе Gazebo.

Если Gazebo будет использоваться в качестве независимого имитатора, то по умолчанию имитация робота будет нужна для написания плагинов на основе C++ (http://gazebosim.org/tutorials/?tut=plugins_hello_world). Мы можем написать плагины на C++. Они будут моделировать поведение робота, создавать новые датчики, новые миры и т. д. По умолчанию моделирование роботов в среде Gazebo осуществляется с помощью SDF-файла (SDF – это формат XML, описывающий объекты и среды для моделирования роботов, визуализации и управления: http://sdformat.org/). Если мы используем интерфейс ROS в Gazebo, нам следует создать URDF-файл, содержащий все параметры робота, и теги Gazebo, содержащие свойства модели. В начале моделирования с использованием URDF-файла преобразуем его в файл SDF и с помощью некоторых инструментов отобразим робота в Gazebo. Интерфейс ROS Gazebo называется gazebo-ros-pkgs. Это набор оболочек и плагинов, позволяющих сымитировать датчик, контроллер робота и выполнить другое моделирование и взаимодействие Gazebo с ROS. В основном в данной главе мы сфокусируемся на взаимодействии ROS-Gazebo, позволяющем смоделировать Chefbot. Преимущество взаимодействия ROS-Gazebo заключается в том, что мы можем запрограммировать робота, используя фреймворк ROS и такие популярные языки программирования, как C++ и Python.

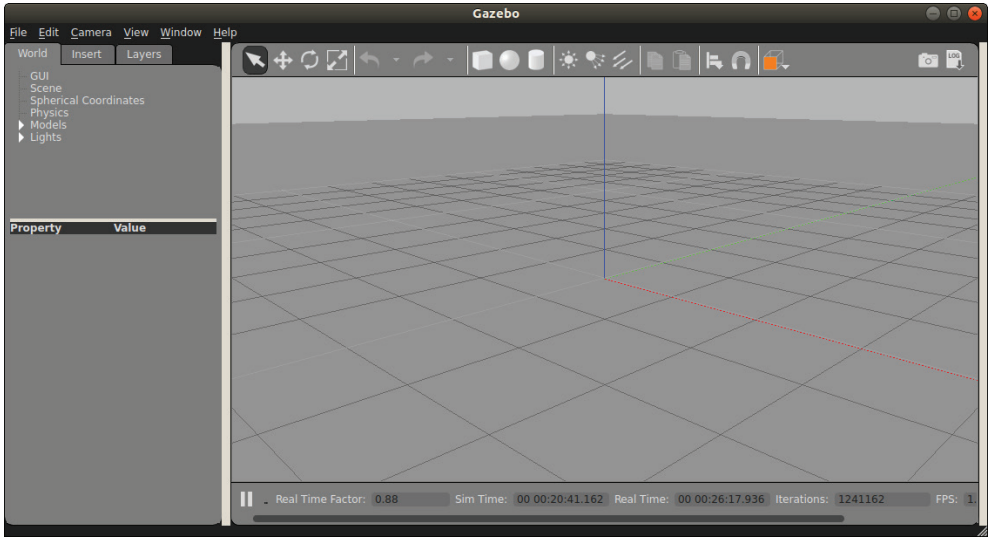
Если для программирования робота вы вместо ROS желаете использовать Python, следует воспользоваться интерфейсом **pygazebo** (https://github.com/jpieper/pygazebo). Это фреймворк, связывающий Python и Gazebo. В следующем разделе мы познакомимся с графическим интерфейсом Gazebo и некоторыми из важных элементов управления.

Графический интерфейс пользователя Gazebo

Запустить Gazebo можно несколькими способами. Мы это уже делали в главе 1 «Начало работы с операционной системой для робота (ROS)». Сейчас для запуска Gazebo и создания необитаемого мира, в котором отсутствует как робот, так и окружающая среда, мы используем следующую команду:

```
$ roslaunch gazebo_ros empty_world.launch
```

Данная команда запустит сервер и клиент Gazebo и загрузит в него необитаемый мир. Ниже показан интерфейс Gazebo с загруженным необитаемым миром:



Пользовательский интерфейс Gazebo

Пользовательский интерфейс Gazebo можно разделить на три части: сцену, левую и правую панели.

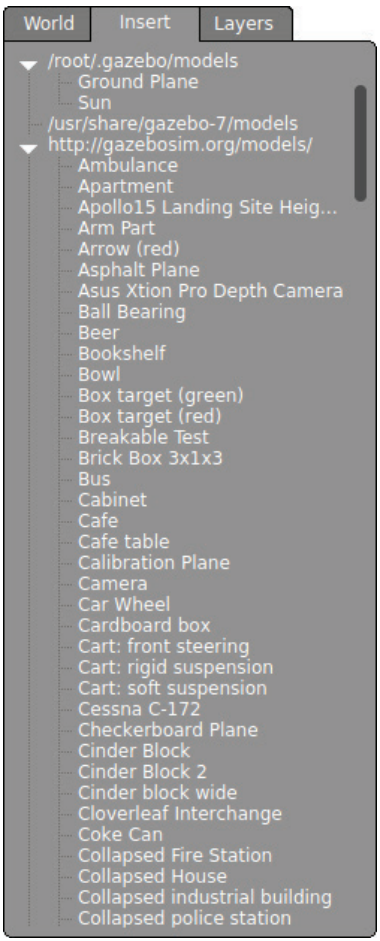
Сцена

Сцена – это место, где происходит моделирование объекта. Сюда можно добавить различные объекты и взаимодействовать с роботом с помощью мыши и клавиатуры.

Левая панель

Левая панель появляется при каждом запуске Gazebo. Она состоит из трех основных вкладок:

- **мир**. Данная вкладка содержит список моделей активной сцены Gazebo. Здесь можно доработать параметры модели и изменить положение камеры;
- **вставка**. Эта вкладка позволяет добавить на сцену новую модель. Модели можно загрузить как из локальной системы, так и с удаленного сервера. На компьютере (в локальной системе) файлы модели находятся в папке `/home/<user_name>/.gazebo/mode`. Адрес удаленного сервера – http://gazebosim.org/models (адрес показан на вкладке **Insert** (Вставка) левой панели):



Вкладка **Insert** левой панели

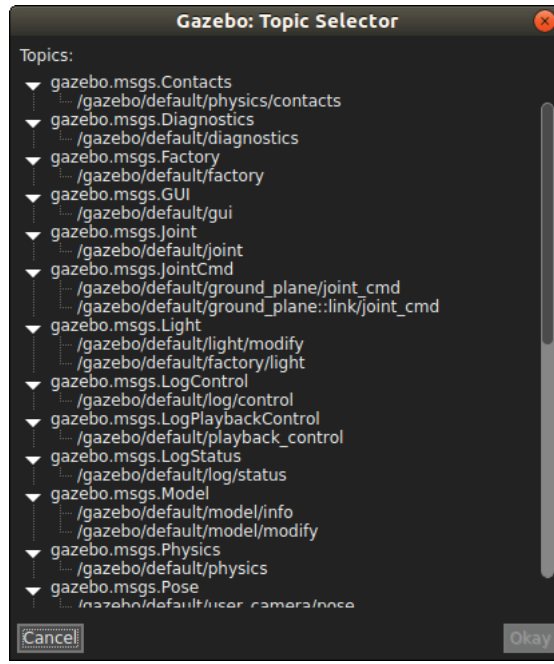
На приведенном рисунке в верхней части вкладки **Insert** показаны файлы, расположенные на данном компьютере. Ниже вы увидите названия моделей, хранящихся на удаленном сервере.

После первого запуска Gazebo или при создании мира, содержащего модель с удаленного сервера, вместо сцены некоторое время будет черный экран или предупреждение в терминале. Причина – низкая скорость загрузки модели с удаленного сервера, поэтому Gazebo ждет некоторое время. Время ожидания может варьироваться в зависимости от скорости вашего интернет-соединения. Сразу после загрузки модель будет сохранена в локальной папке **Модели**. В следующий раз при загрузке не будет никакой задержки;

- **слои**. В основном эта вкладка не используется и предназначена для организации слоев при моделировании. Отключая или включая нужный слой, можно скрыть или показать отдельные части модели.

Информационная панель

По умолчанию информационная панель скрыта и появляется только по команде меню, например **Window → Topic Visualization**.



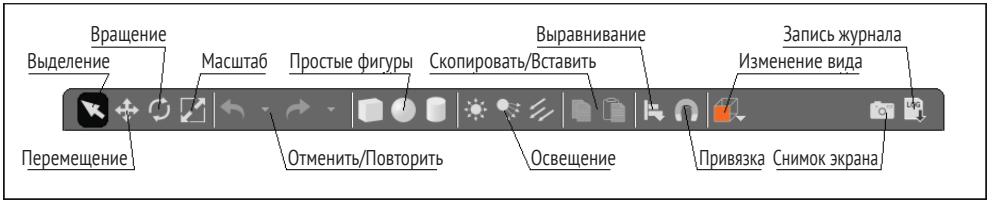
Информационная панель Gazebo

Панели инструментов Gazebo

Gazebo оснащена двумя панелями инструментов, которые установлены в верхней и нижней частях сцены.

Верхняя панель инструментов

На верхней панели инструментов находятся инструменты управления сценой, такие как выделение модели, изменение масштаба, поворот, добавление новой простой фигуры и т. д.



Верхняя панель инструментов

Ниже приведен полный список инструментов с их полным описанием:

- **Select Mode** (Выделение) – данный инструмент позволяет выделять модель, находящуюся на сцене;
- **Translate Mode** (Перемещение) – этот инструмент позволяет перемещать модель с помощью нажатия и удерживания левой кнопки мыши;
- **Rotate Mode** (Вращение) – изменение ориентации модели;
- **Scale Mode** (Масштаб) – изменение масштаба модели;
- **Undo/Redo** (Отменить/Повторить) – позволяет отменить или повторить действие на сцене;
- **Simple Shapes** (Простые фигуры) – с помощью этого инструмента мы можем вставлять примитивные фигуры на сцену (цилиндр, куб или сферу);
- **Lights** (Освещение) – выбор различных источников и вариантов освещения сцены;
- **Copy/Paste** (Скопировать/Вставить) – данные инструменты позволяют копировать и вставлять различные модели и целые части сцены;
- **Align** (Выравнивание) – выравнивание моделей относительно друг друга;
- **Snap** (Привязка) – привязка одной модели и перемещение ее внутри сцены;
- **Change view** (Изменение вида) – изменяет вид сцены, в основном выбирается перспектива или ортогональный вид;
- **Screenshot** (Снимок экрана) – снимок экрана текущей сцены;
- **Record Log** (Запись журнала) – сохраняет логи (журнал) Gazebo.

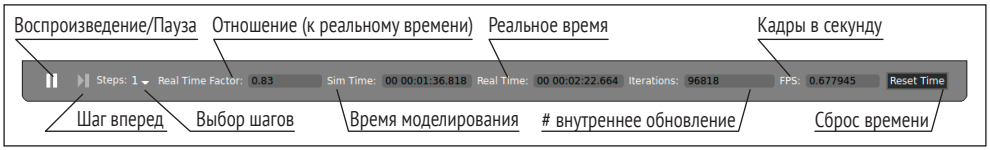
Нижняя панель инструментов

Основная задача нижней панели инструментов – дать нам представление о визуализации. На этой панели отображается время, в течение которого происходит моделирование. Процесс моделирования можно ускорять и замедлять. Это зависит от вычислений, необходимых для текущего моделирования.

Отображение в реальном времени относится к фактическому времени, за которое объект совершает действие при моделировании.

Коэффициент реального времени (RTF) – это соотношение между временем моделирования и скоростью реального времени. Если RTF равно 1, это означает, что моделирование происходит со скоростью, тождественной скорости реального времени.

Состояние мира в Gazebo может меняться с каждым циклом. Каждый цикл может вызвать изменения в Gazebo за равные временные промежутки. Эти равные временные промежутки называются размером шага. Размер шага равен 1 мс. Размер шага и циклы отображаются на нижней панели инструментов:



Нижняя панель инструментов

Мы можем приостановить моделирование и с помощью кнопки **Step** (Шаг) рассмотреть каждый шаг в любой момент времени.

Более подробную информацию об интерфейсе Gazebo можно найти по адресу: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b2.

Прежде чем перейти к следующему разделу, можно попробовать поработать с Gazebo и подробнее узнать о том, как он устроен.

Работа с симулятором TurtleBot 2

После того как вы познакомились с Gazebo и самостоятельно опробовали его инструменты, пришло время запустить моделирование и поработать с моделью робота. Одним из популярных роботов, доступных для изучения, является TurtleBot. Программное обеспечение TurtleBot было разработано в рамках ROS. Оно содержит хорошую симуляцию, работающую в Gazebo. Популярными версиями TurtleBot являются TurtleBot 2 и 3. В этом разделе познакомимся с TurtleBot 2, потому что конструкция Chefbot очень напоминает конструкцию этого робота.

Установить пакеты моделирования TurtleBot 2 в Ubuntu 16.04 просто. Для этого можно использовать следующую команду:

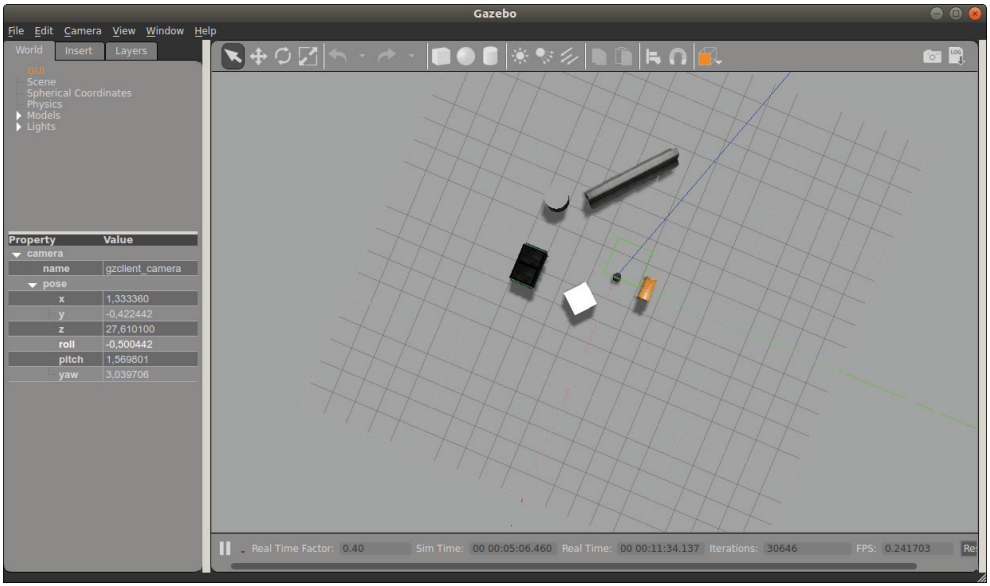
```
$ sudo apt-get install ros-kinetic-turtlebot-gazebo
```

После того как пакеты будут установлены, можно запустить симуляцию. Внутри пакетов turtlebot-gazebo находится несколько исполняемых файлов, содержащих разные файлы мира в Gazebo. Мир Gazebo представляет собой SDF-файл (`*.world`), состоящий из свойств моделей в среде. При изменении файла привязки в Gazebo загрузится другая среда.

Следующая команда запустит мир, который имеет определенный набор компонентов:

```
$ roslaunch turtlebot_gazebo turtlebot_world.launch
```

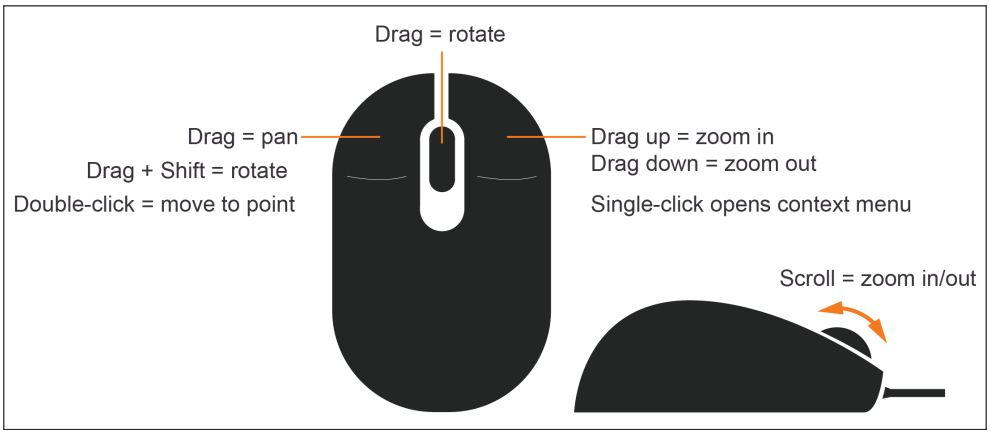
Загрузка модели займет некоторое время. По окончании загрузки на сцене Gazebo появится загруженная модель:



Моделирование в Gazebo робота Turtlebot

Возможно, после загрузки камера будет смотреть на модель сверху. Поэтому сцену со всеми находящимися на ней моделями следует развернуть и изменить масштаб.

Прежде чем развернуть сцену, познакомимся с базовыми операциями, выполняемыми с помощью мыши:

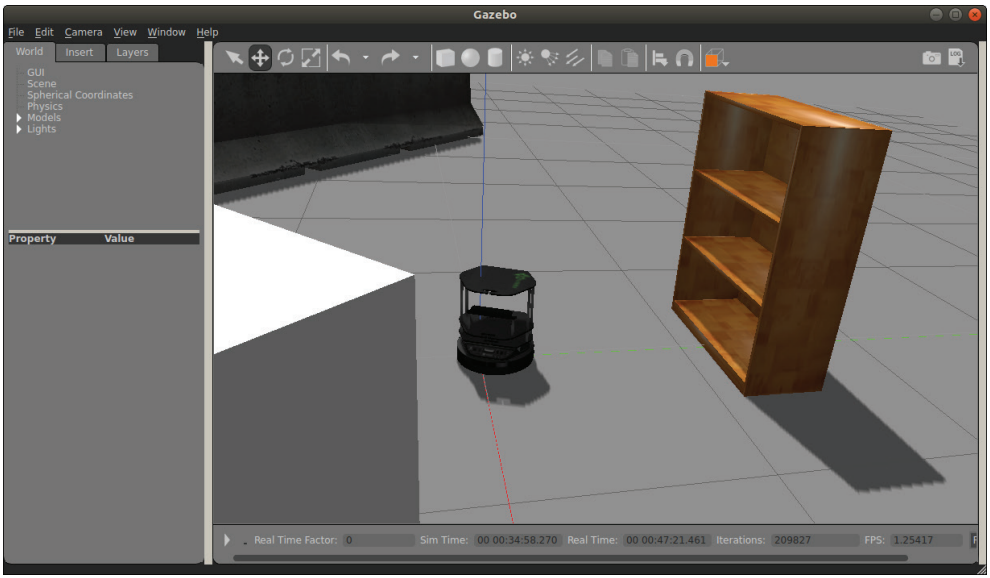


Основные операции, выполняемые в Gazebo мышью

С помощью мыши вы можете выполнять следующие операции:

- **Drag = rotate** – для вращения сцены нажмите колесико мыши и перемещайте мышь;
- **Drag = pan** – для панорамирования (перемещения всей сцены) нажмите левую кнопку мыши и, удерживая ее, перемещайте мышь в направлении, в котором требуется сместить сцену;
- **Drag + Shift = rotate** – для вращения сцены нажмите и удерживайте левую кнопку мыши и клавишу **Shift**. Удерживая нажатыми левую кнопку и клавишу **Shift**, перемещайте указатель мыши в нужном направлении;
- **Double-click = move to point** – переместить в точку;
- **Drag up = zoom in** – для увеличения масштаба нажмите правую кнопку мыши и, удерживая правую кнопку нажатой, переместите указатель мыши вверх;
- **Drag down = zoom out** – для уменьшения масштаба нажмите правую кнопку мыши и, удерживая правую кнопку нажатой, переместите указатель мыши вниз;
- **Single-click opens context menu** – для вызова контекстного меню щелкните на сцене правой кнопкой мыши;
- **Scroll = zoom in/out** – изменение масштаба с помощью вращения колесика мыши.

Разверните сцену. Для этого нажмите колесико мыши и, не отпуская его, проведите примерно на 1/3 оборота по дуге в направлении против часовой стрелки. Изменяя масштаб и поворачивая сцену в нужном направлении, добейтесь того, чтобы робот был хорошо виден.

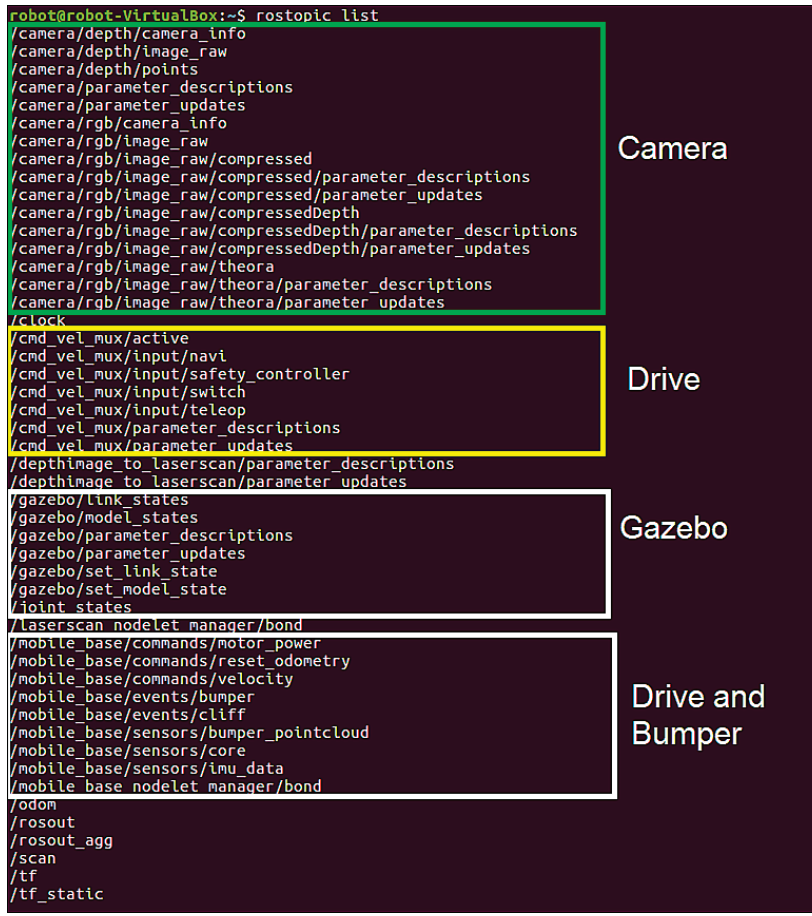


Сцена развернута

При загрузке модели робота в Gazebo будут загружены плагины, необходимые для взаимодействия с ROS. TurtleBot 2 состоит из таких компонентов, как:

- мобильная платформа с дифференциальным приводом;
- датчик глубины для создания карты;
- выключатель бампера для обнаружения столкновения.

При загрузке модели будут загружены плагины ROS-Gazebo, позволяющие сымитировать управление мобильной платформой с дифференциальным приводом, плагины датчика глубины (Kinect или Astra) и плагины для переключателей бампера. Если после загрузки модели ввести в терминале команду `$ rostopic list`, то в терминале получим информацию о загруженных плагинах. Список этих плагинов показан на представленном ниже снимке экрана.



Темы ROS для моделирования TurtleBot 2

В этом списке показаны файлы плагинов дифференциального привода, датчика глубины и переключателей бампера. В дополнение показаны файлы плагинов ROS-Gazebo, содержащих в основном сведения о текущем состоянии робота и других моделей в симуляции.

Датчики Kinect/Astra раскладывают изображение на RGB составляющие и обеспечивают глубину изображения. Плагин дифференциального привода отправляет одометрические данные робота в тему `/odom` (`nav_msgs/Odometry`) и публикует трансформацию робота в темах `/tf` (`tf2_msgs/TFMessage`).

Мы можем визуализировать модель робота и данные датчика в Rviz. Для этих целей предусмотрен пакет TurtleBot. Для визуализации робота можно установить следующий пакет данных:

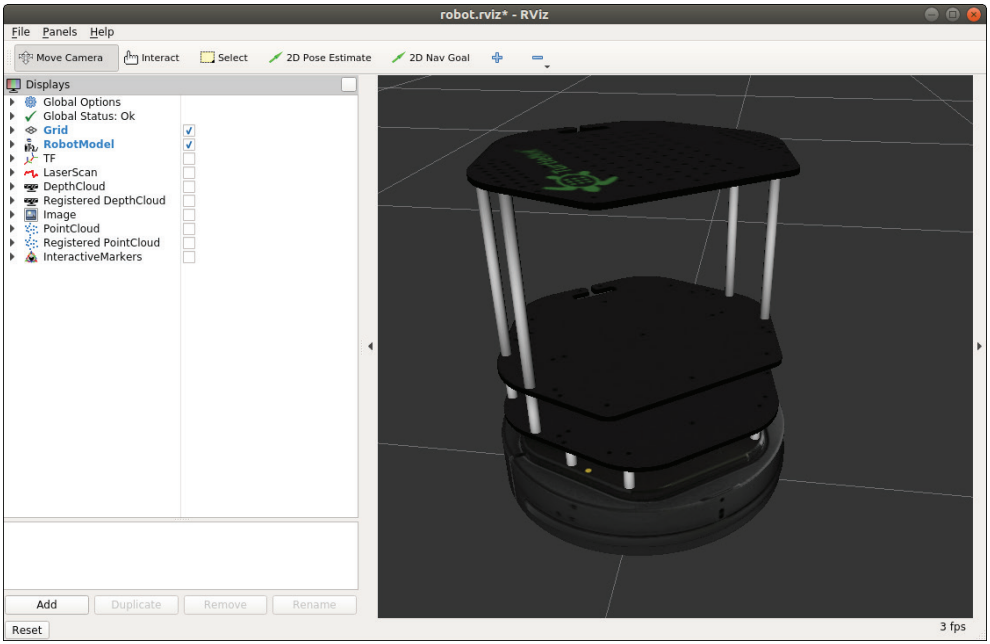
```
$ sudo apt-get install ros-kinetic-turtlebot-rviz-launchers
```

После установки этого пакета запустите визуализацию робота и данных датчиков. Для этого выполните следующие действия.

Не завершая работу Gazebo, запустите еще один терминал, введите команду Sudo su и пройдите авторизацию. Далее введите команду:

```
$ roslaunch turtlebot_rviz_launchers view_robot.launch
```

После отработки команды на экране появится окно программы визуализации Rviz, в котором вы увидите модель робота.



Визуализация TurtleBot 2 в RViz

Теперь мы можем включить необходимые датчики.

Закройте окно RViz, Gazebo и один из терминалов.

В следующем разделе мы узнаем, как дистанционно управлять перемещением этого робота.

Перемещение робота

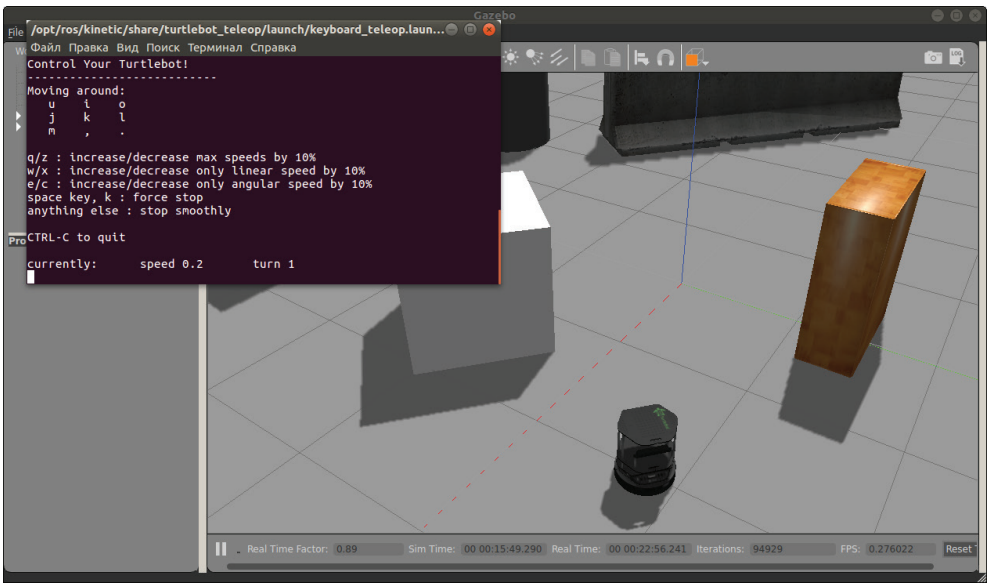
Плагин дифференциального привода робота позволяет принимать сообщения Ros Twist (`geometry_msgs/Twist`), состоящие из текущих линейных и угловых скоростей робота. Дистанционное управление роботом осуществляется вручную, с помощью джойстика или клавиатуры и сообщений Ros Twist. Далее мы рассмотрим, как осуществляется дистанционное управление роботом Turtlebot 2 с помощью клавиатуры. Для этого нам необходимо установить пакет для управ-

ления роботом TurtleBot 2. Для установки пакета дистанционного управления TurtleBot запустите окно терминала (если ранее оно было закрыто), введите команду `sudo su`, свой пароль и воспользуйтесь следующей командой:

```
$ sudo apt-get install ros-kinetic-turtlebot-teleop
```

Теперь запустите моделирование робота (`$ roslaunch turtlebot_gazebo turtlebot_world.launch`) и разверните сцену так, чтобы робот стоял на горизонтальной плоскости и был хорошо виден. Далее запустите еще одно окно терминала и узел дистанционного управления:

```
$ roslaunch turtlebot_teleop keyboard_teleop.launch
```

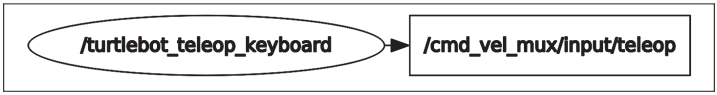


Дистанционное управление TurtleBot 2

После запуска узла дистанционного управления в терминале отобразятся комбинации клавиш для перемещения робота. С их помощью робот перемещается в Gazebo и Rviz.

Для движения робота вперед используется клавиша **,** (запятая); назад – клавиша **I**; для разворота на месте по часовой стрелке – клавиша **L**; для разворота на месте против часовой стрелки – клавиша **J**; для поворота вправо – клавиша **M**; влево – клавиша **.** (точка); назад вправо – клавиша **U**.

При нажатии клавиш на клавиатуре Gazebo посылает сообщение регулятору управления дифференциальным приводом, а регулятор, в свою очередь, дает команду на перемещение робота. Узел teleop отправляет тему с именем `/cmd_vel_mux/input/teleop` (`geometry_msgs/Twist`). Описанное взаимодействие показано на схеме:



Узел дистанционного управления TurtleBot

Создание симуляции в Chefbot

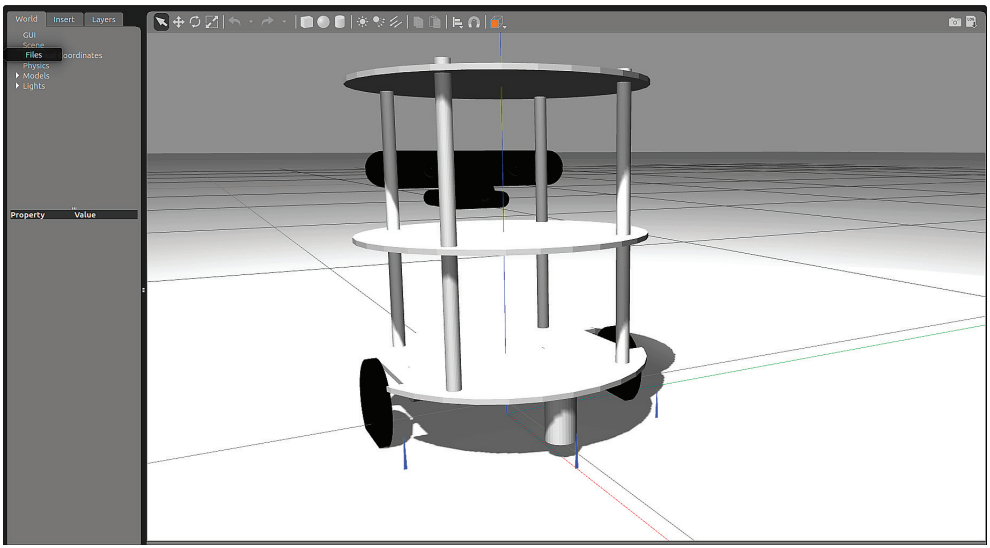
В предыдущем разделе мы смоделировали дистанционное управление роботом TurtleBot 2. В этом разделе рассмотрим, как с помощью Gazebo смоделировать созданный ранее робот ChefBot.

Сначала нужно скопировать пакет `chefbot_gazebo` в рабочее пространство `catkin` и выполнить команду `catkin_make` для сборки пакета. Убедитесь, что в рабочей области находятся 2 пакета: `chefbot_description` и `chefbot_gazebo`. Пакет `chefbot_gazebo` содержит файл запуска, связанный с моделированием и его параметрами, а `chefbot_description` содержит модель urdf-робота вместе с его параметрами моделирования и файлом запуска, который используется для просмотра робота в Rviz и Gazebo.

Приступим к созданию нашей модели и ознакомимся с порядком создания робота Chefbot в Gazebo, после чего более подробно изучим файл xacro и рассмотрим параметры моделирования.

```
$ roslaunch chefbot_description view_robot_gazebo.launch
```

На следующем рисунке показан снимок экрана Chefbot в Gazebo:



Chefbot в Gazebo

Посмотрим, как добавить модель робота URDF в Gazebo. Вы можете найти URDF-описание модели робота в `chefbot_description/launch/view_robot_gazebo.launch`.

Первый раздел кода вызывает файл `upload_model.launch`, назначение которого – создание параметра `robot_description`. Если все пройдет успешно, то в Gazebo будет создан незаполненный мир:

```
<launch>
  <include file="$(find chefbot_description)/launch/upload_model.launch" />

  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="recording" value="false"/>
    <arg name="debug" value="false"/>
</include>
```

Так как же модель робота в параметре `robot_description` отображается в Gazebo? Для этого предназначается следующий фрагмент кода:

```
  <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-param
robot_description -urdf -z 0.1 -model chefbot"/>
```

Узел под названием `spawn_model`, находящийся внутри пакета `gazebo_ros`, прочитает `robot_description` и создаст модель в Gazebo. Аргумент `-z 0.1` указывает на высоту модели, размещенной в Gazebo. Значение высоты отображает, на каком расстоянии от земли будет создана модель. Например, если значение высоты будет 0.1, то модель зависнет на высоте 0.1. При включенном режиме гравитации она упадет на землю. Однако этот параметр можно изменить так, как нужно нам. Аргумент `-model` обозначает имя модели робота в Gazebo. Этот узел проведет анализ файла описания робота `robot_description` и начнет визуализацию робота в Gazebo.

После того как модель появится на экране, можно начать преобразование робота (tf), используя следующие строки программного кода:

```
  <node pkg="robot_state_publisher" type="robot_state_publisher"
name="robot_state_publisher">
    <param name="publish_frequency" type="double" value="30.0" />
  </node>
```

Ros tf публикуется на частоте 30 Гц.

Преобразование глубины изображения с помощью лазерного сканера

Датчик глубины робота обеспечивает 3D-координаты окружающей его среды. Чтобы робот мог автономно передвигаться, необходимо создать трехмерную карту. Для этого и используются трехмерные координаты окружающей среды, получаемые от датчика глубины. Есть разные методы создания такой карты.

Один из алгоритмов, используемых для этого робота, называется **gmapping** (http://wiki.ros.org/gmapping). Алгоритм gmapping для создания карты в основном использует лазерное сканирование, но в нашем случае мы получаем от датчика целое 3D-облако точек. Мы можем преобразовать трехмерные данные глубины, полученные с помощью лазерного сканирования, взяв срез данных глубины. Исполняемый файл **nodelet** (http://wiki.ros.org/nodelet) предназначен для получения данных глубины и их конвертации в данные лазерного сканирования:

```
  <node pkg="nodelet" type="nodelet" name="laserscan_nodelet_manager"
args="manager"/>
  <node pkg="nodelet" type="nodelet" name="depthimage_to_laserscan"
        args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet
laserscan_nodelet_manager">
    <param name="scan_height" value="10"/>
    <param name="output_frame_id" value="/camera_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <remap from="image" to="/camera/depth/image_raw"/>
    <remap from="scan" to="/scan"/>
  </node>
</launch>
```

Nodelet – это особый тип узла ROS, которому присущи свойства транспортного средства нулевого копирования, т. е. для подписки на тему данному узлу не требуется пропускная способность сети. Благодаря этому свойству преобразование изображения глубины (`sensor_msgs/Image`) в лазерное сканирование (`sensor_msgs/LaserScan`) будет проходить быстрее и эффективнее. Одно из свойств nodelet заключается в том, что этот узел может быть динамически загружен как плагин. Этому узлу можно задавать различные свойства, такие как `range_min`, название темы изображения и темы вывода лазера.

Теги и плагины URDF для моделирования Gazebo

Ранее мы узнали, как создать визуализацию модели робота в Gazebo. Теперь более подробно рассмотрим все связанные с моделированием теги и плагины, включенные в URDF-модель.

Большая часть из тегов Gazebo находится в файле `chefbot_description/gazebo/chefbot.gazebo.xacro`. Кроме того, в моделировании используются некоторые теги `chefbot_description/urdf/chefbot.xacro`. В моделировании chefbot.xacro также очень большую роль играют теги `<collision>` и `<inertial>`. Тег `<collision>` определяет в URDF границу вокруг робота, которая в основном используется для обнаружения столкновения. Тег `<inertial>` описывает массу всех компонентов конструкции и момент инерции. Ниже приведен пример определения тега `<inertial>`:

```
<inertial>
  <mass value="0.564" />
  <origin xyz="0 0 0" />
```

```
  <inertia ixx="0.003881243" ixy="0.0" ixz="0.0"
           iyy="0.000498940" iyz="0.0"
           izz="0.003879257" />
</inertial>
```

Эти параметры являются частью динамики робота, поэтому при моделировании данные величины будут оказывать влияние на модель. Кроме того, при моделировании все эти все сообщения, соединения и свойства робота будут обновляться.

Далее ознакомимся с тегами, находящимися внутри файла `gazebo/chefbot.gazebo.xacro`. Важный тег Gazebo, используемый нами, называется `<gazebo>`. Он нужен для определения свойств моделирования элемента в роботе. Мы можем определить свойство, применимое ко всем сообщениям или одному конкретному сообщению. Ниже представлен фрагмент кода, находящегося внутри xacro-файла и определяющего коэффициент трения:

```
<gazebo reference="chefbot_wheel_left_link">
  <mu1>1.0</mu1>
  <mu2>1.0</mu2>
  <kp>1000000.0</kp>
  <kd>100.0</kd>
  <minDepth>0.001</minDepth>
  <maxVel>1.0</maxVel>
</gazebo>
```

Свойство `reference` используется для указания ссылки в роботе. Таким образом, предыдущие свойства будет применяться только к `chefbot_wheel_left_link`.

В следующем фрагменте кода показано, как задается цвет робота. Можно задать собственный цвет, определить пользовательские цвета или использовать цвета, предлагаемые в Gazebo по умолчанию. Для `base_link` используется белый цвет Gazebo из свойств, предлагаемых в Gazebo по умолчанию:

```
<material name="blue">
  <color rgba="0 0 0.8 1"/>
</material>
<gazebo reference="base_link">
  <material>Gazebo/White</material>
</gazebo>
```

Чтобы увидеть все теги, используемые в моделировании, перейдите по ссылке http://gazebosim.org/tutorials/?tut=ros_urdf.

Мы рассмотрели основные теги, применяемые в моделировании. Теперь рассмотрим плагины Gazebo-ROS, которые были использованы в моделировании.

Плагин датчика препятствия

Датчик препятствия представляет собой комплект ИК-датчиков, которые предназначены для обнаружения препятствий на пути робота. Данный комплект позволяет предотвратить столкновение робота с препятствием или его падение.

Это один из датчиков ходовой части робота Turtlebot 2, называемый kobuki (http://kobuki.yujinrobot.com/). Этот плагин был использован при моделировании робота Turtlebot 2.

Мы можем самостоятельно задать такие параметры датчиков, как минимальный и максимальный углы развертки ИК-лучей, разрешение и частота, с которой будут проводиться замеры. Этот параметр представляет собой количество замеров в секунду. Также можно ограничить зону обнаружения датчика. Из нижеприведенного кода видно, что в модели предусмотрены три датчика препятствия:

```
<gazebo reference="cliff_sensor_front_link">
  <sensor type="ray" name="cliff_sensor_front">
    <always_on>true</always_on>
    <update_rate>50</update_rate>
    <visualize>true</visualize>
    <ray>
      <scan>
        <horizontal>
          <samples>50</samples>
          <resolution>1.0</resolution>
          <min_angle>-0.0436</min_angle> <!-- -2.5 degree -->
          <max_angle>0.0436</max_angle> <!-- 2.5 degree -->
        </horizontal>
      </scan>
      <range>
        <min>0.01</min>
        <max>0.15</max>
        <resolution>1.0</resolution>
      </range>
    </ray>
  </sensor>
</gazebo>
```

Плагин контактного датчика

Вот фрагмент кода контактного датчика нашего робота. Если робот столкнется с любым объектом, сработает этот плагин. Обычно он подключается к `base_link` робота. Датчик срабатывает сразу после того, как бампер робота коснется препятствия:

```
<gazebo reference="base_link">
  <mu1>0.3</mu1>
  <mu2>0.3</mu2>
  <sensor type="contact" name="bumpers">
    <always_on>1</always_on>
    <update_rate>50.0</update_rate>
    <visualize>true</visualize>
    <contact>
      <collision>base_footprint_collision_base_link</collision>
    </contact>
  </sensor>
</gazebo>
```

Плагин гироскопа

Назначение данного плагина – измерение угловой скорости робота. Зная угловую скорость, можно вычислить положение. Положение робота используется в контроллере привода ходовой части для вычисления позы устройства (смотрите приведенный ниже код):

```
      <gazebo reference="gyro_link">
        <sensor type="imu" name="imu">
          <always_on>true</always_on>
          <update_rate>50</update_rate>
          <visualize>false</visualize>
          <imu>
            <noise>
            <type>gaussian</type>
            <rate>
              <mean>0.0</mean>
              <stddev>${0.0014*0.0014}</stddev> <!-- 0.25 x 0.25 (deg/s) -->
              <bias_mean>0.0</bias_mean>
              <bias_stddev>0.0</bias_stddev>
            </rate>
              <accel> <!-- not used in the plugin and real robot, hence
using tutorial values -->
                <mean>0.0</mean>
                <stddev>1.7e-2</stddev>
                <bias_mean>0.1</bias_mean>
                <bias_stddev>0.001</bias_stddev>
              </accel>
            </noise>
          </imu>
        </sensor>
      </gazebo>
```

Модуль дифференциального привода

Плагин дифференциального привода является наиболее важным в моделировании. Он имитирует поведение дифференциального привода нашего робота и после получения команды `velocity` (линейная и угловая скорости) в виде Ros Twist-сообщения (`geometry_msgs/Twist`) перемещает модель робота. Данный плагин также вычисляет одометрию, в результате чего мы получаем локальное положение робота (смотрите приведенный ниже код):

```
<gazebo>
    <plugin name="kobuki_controller" filename="libgazebo_ros_kobuki.so">
      <publish_tf>1</publish_tf>
      <left_wheel_joint_name>wheel_left_joint</left_wheel_joint_name>
      <right_wheel_joint_name>wheel_right_joint</right_wheel_joint_name>
      <wheel_separation>.30</wheel_separation>
      <wheel_diameter>0.09</wheel_diameter>
      <torque>18.0</torque>
      <velocity_command_timeout>0.6</velocity_command_timeout>
      <cliff_detection_threshold>0.04</cliff_detection_threshold>
```

```
      <cliff_sensor_left_name>cliff_sensor_left</cliff_sensor_left_name>
<cliff_sensor_center_name>cliff_sensor_front</cliff_sensor_center_name>
<cliff_sensor_right_name>cliff_sensor_right</cliff_sensor_right_name>
      <cliff_detection_threshold>0.04</cliff_detection_threshold>
      <bumper_name>bumpers</bumper_name>
      <imu_name>imu</imu_name>
    </plugin>
</gazebo>
```

Для вычисления одометрии нам необходимо знать такие параметры, как расстояние между колесами, диаметр колеса и вращающий момент моторов. В нашей конструкции расстояние между колесами составляет 30 см, диаметр колеса – 9 см, а вращающий момент равен 18 Н. Чтобы опубликовать преобразование робота, следует установить значение `publish_tf`, равное 1. Каждый тег внутри плагина является его параметром. Вы можете видеть, что данный плагин принимает все входные сигналы от датчика контакта (imu) и датчика препятствия.

Плагин `libgazebo_ros_kobuki.so` устанавливается вместе с пакетами симуляции Turtlebot 2. Этот плагин мы используем в нашем роботе. Перед запуском моделирования убедитесь, что моделирование Turtlebot 2 установлено в вашей системе.

Плагин камеры глубины

Данный плагин имитирует характеристики камеры глубины таких камер, как Kinect или Astra, и разных датчиков глубины с различными характеристиками. Имя этого плагина – `libgazebo_ros_openni_kinect.so`. Ниже приведен его код:

```
    <plugin name="kinect_camera_controller"
filename="libgazebo_ros_openni_kinect.so">
      <cameraName>camera</cameraName>
      <alwaysOn>true</alwaysOn>
      <updateRate>10</updateRate>
      <imageTopicName>rgb/image_raw</imageTopicName>
      <depthImageTopicName>depth/image_raw</depthImageTopicName>
      <pointCloudTopicName>depth/points</pointCloudTopicName>
      <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
<depthImageCameraInfoTopicName>depth/camera_info</depthImageCameraInfoTopicName>
      <frameName>camera_depth_optical_frame</frameName>
      <baseline>0.1</baseline>
      <distortion_k1>0.0</distortion_k1>
      <distortion_k2>0.0</distortion_k2>
      <distortion_k3>0.0</distortion_k3>
      <distortion_t1>0.0</distortion_t1>
      <distortion_t2>0.0</distortion_t2>
      <pointCloudCutoff>0.4</pointCloudCutoff>
    </plugin>
```

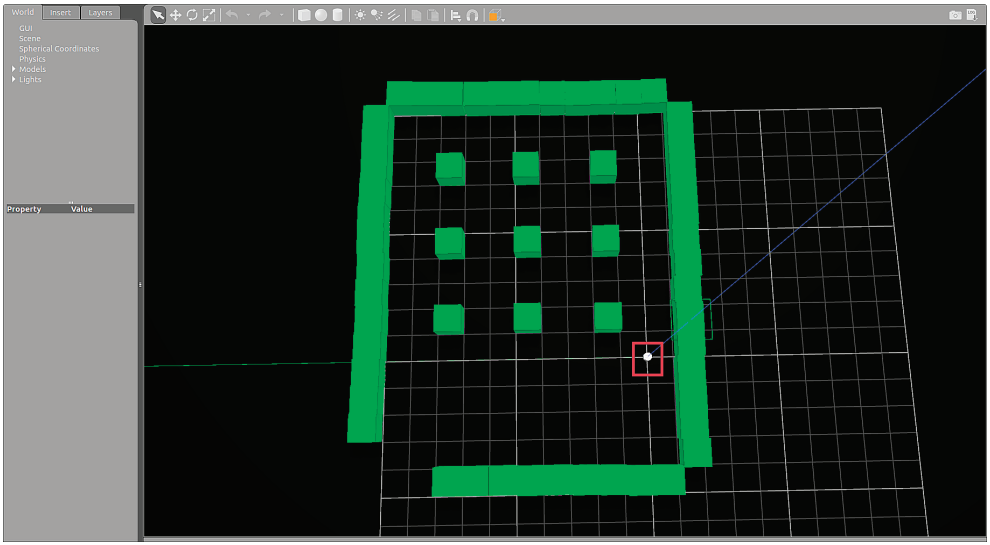
Плагин публикует изображение в трех основных цветах (красном, зеленом и синем – RGB), показывает глубину изображения и данные облака точек. Можно установить матрицу камеры в плагине, а также настроить другие параметры.

Чтобы узнать больше о плагине камеры глубины в Gazebo, перейдите по ссылке http://gazebosim.org/tutorials?tut=ros_depth_cameraamp;cat=connect_ros.

Визуализация данных датчика робота

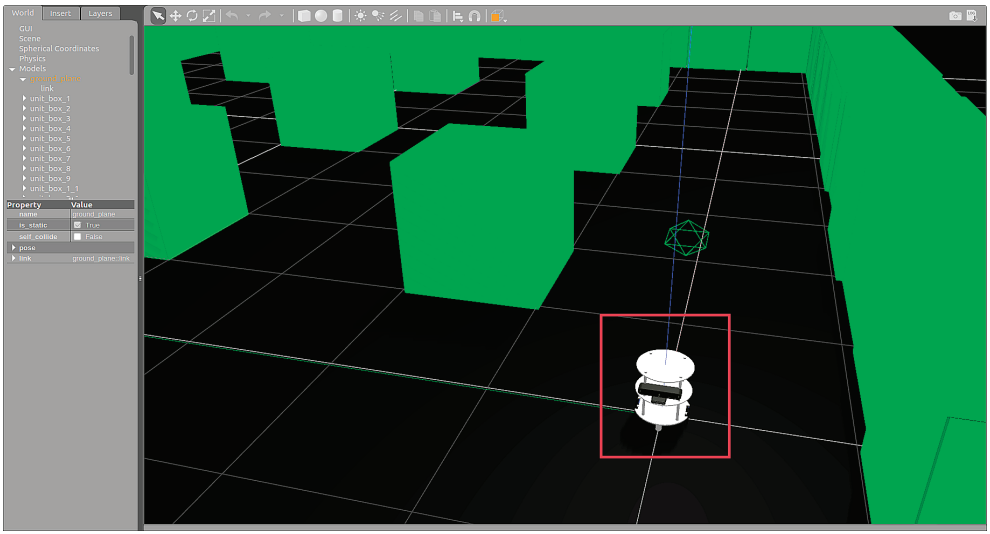
В этом разделе мы узнаем, как сделать видимыми данные датчика от моделируемого робота. В пакете `chefbot_gazebo` есть два файла запуска: для запуска робота в незаполненном мире и в окружающей среде (в нашем случае – в отеле). Пользовательскую среду можно построить с использованием инструментов и самого Gazebo. Создайте с помощью примитивных меш-объектов простую среду и сохраните ее как `*.world`-файл, который может быть входом узла `gazebo_ros` в файле запуска. Для запуска гостиничной среды в Gazebo используйте следующую команду:

```
$ roslaunch chefbot_gazebo chefbot_hotel_world.launch
```



Chefbot в Gazebo в среде отеля

Девять кубиков внутри пространства представляют девять столов. Робот может перемещаться к любому из этих столов для доставки еды. Мы расскажем, как это делать, но перед этим узнаем, как сделать видимыми различные типы данных, получаемых от датчиков модели робота.

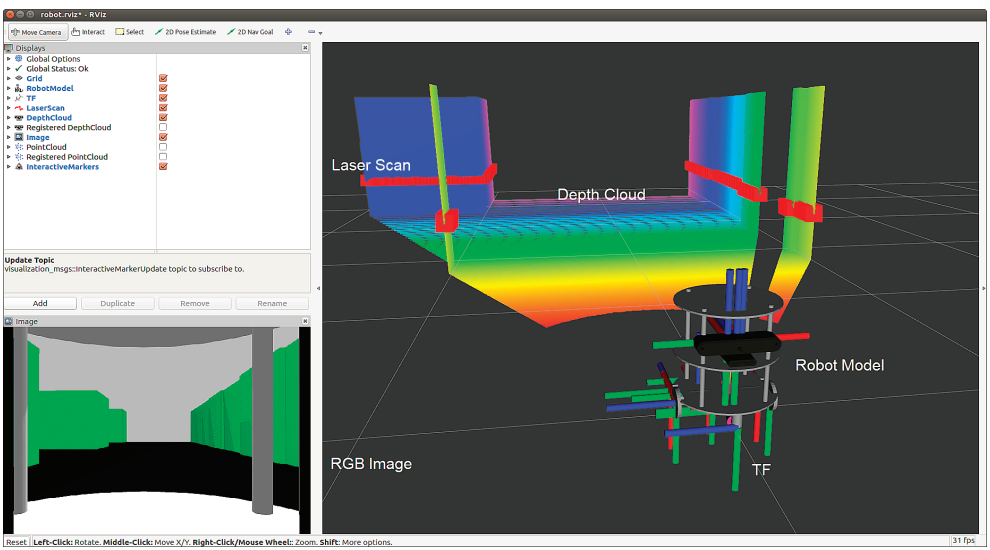


Среда отеля в Gazebo с роботом Chefbot

Чтобы сделать видимыми данные, получаемые от датчика робота, запустим Rviz. Для этого выполните следующую команду:

```
$ roslaunch chefbot_description view_robot.launch
```

Как данные будут отображаться на экране, показано на приведенном ниже рисунке:



Визуализация датчика Chefbot в Rviz

Для просмотра разных типов данных от датчиков Chefbot можем в Rviz выбирать требуемые способы отображения. На предыдущем рисунке показаны облако глубины, данные, полученные от датчика лазерного сканирования, TF-ракеты, сама модель робота и RGB-камера изображения.

Начало работы с одновременной локализацией и картографированием

Одно из требований к Chefbot – то, что он должен уметь автономно перемещаться в окружающей среде и доставлять пищу. Для достижения этого требования следует использовать несколько алгоритмов, таких как SLAM (одновременная локализация и отображение) и AMCL (Адаптивная локализация Монте-Карло). Существует несколько подходов к решению проблемы автономной навигации. В этой книге мы в основном придерживаемся данных алгоритмов. Алгоритмы SLAM используются для решения двух задач: отображения окружающей среды и одновременного определения положения робота на карте. Это похоже на проблему с курицей и яйцом. Но теперь для одновременного решения этих двух задач существуют различные алгоритмы. Алгоритм AMCL используется для локализации робота на существующей карте. Алгоритм, который мы используем в этой книге, называется **Gmapping** (http://www.openslam.org/gmapping.html), который реализует **Fast SLAM 2.0** (http://robots.stanford.edu/papers/Montemerlo03a.html). Стандартная библиотека gmapping упакована в пакет ROS и называется **Ros Gmapping** (http://wiki.ros.org/gmapping). Этот пакет может быть использован в нашем приложении.

Идея узла SLAM заключается в том, что при перемещении робот, используя данные лазерного сканирования и одометрии, создает карту окружающей среды.

Более подробную информацию вы можете получить, обратившись к странице Ros Gmapping, расположенной по адресу: http://wiki.ros.org/gmapping.

Реализация SLAM в среде Gazebo

В этом разделе будет рассказано, как реализовать SLAM и применить его к созданной нами визуализации. Вы можете проверить код в `chefbot_gazebo/launch/gmapping_demo.launch` и `launch/includes/gmapping.launch.xml`.

Мы в основном используем узел из пакета gmapping и настраиваем его с соответствующими параметрами. Фрагмент кода `gmapping.launch.xml` полностью определяет этот узел. Ниже приведен фрагмент кода данного узла:

```
<launch>
    <arg name="scan_topic" default="scan" />

        <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
            <param name="base_frame" value="base_footprint"/>
            <param name="odom_frame" value="odom"/>
            <param name="map_update_interval" value="5.0"/>
            <param name="maxUrange" value="6.0"/>
            <param name="maxRange" value="8.0"/>
```

Имя используемого нами узла – `slam_gmapping`, а имя пакета – `gmapping`. Для правильной работы необходимо этому узлу предоставить несколько необходимых параметров, которые можно найти на странице Gmapping wiki.

Создание карты с помощью SLAM

В этом разделе мы узнаем, как с помощью SLAM создать карту нашей среды. Однако, перед тем как начать отображение, следует выполнить несколько команд. Все эти команды поочередно выполняются в терминале Linux.

Во-первых, используя следующую команду, необходимо начать симуляцию:

```
$ roslaunch chefbot_gazebo chefbot_hotel_world.launch
```

Далее запустите еще один терминал и запустите в нем узел клавиатуры teleoperation. Это поможет нам настроить перемещение робота вручную с помощью клавиатуры:

```
$ roslaunch chefbot_gazebo keyboard_teleop.launch
```

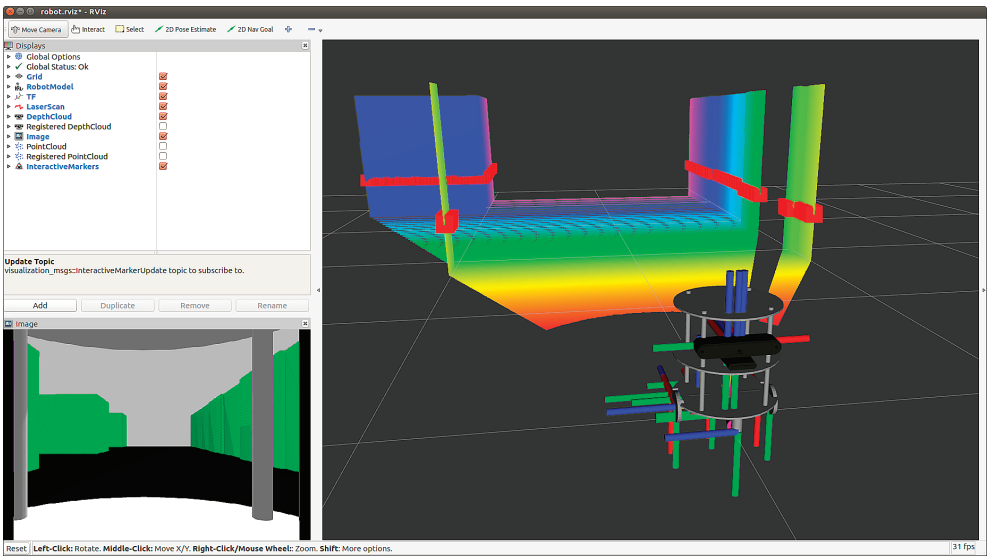
Запустите еще один терминал и выполните в нем нижеприведенную команду, которая запустит SLAM:

```
$ roslaunch chefbot_gazebo gmapping_demo.launch
```

Теперь начнется создание карты. Чтобы показать на экране, как создается карта, запустите Rviz с помощью настроек навигации:

```
$ roslaunch chefbot_description view_navigation.launch
```

Теперь мы можем видеть карту, созданную в Rviz (см. рисунок ниже).



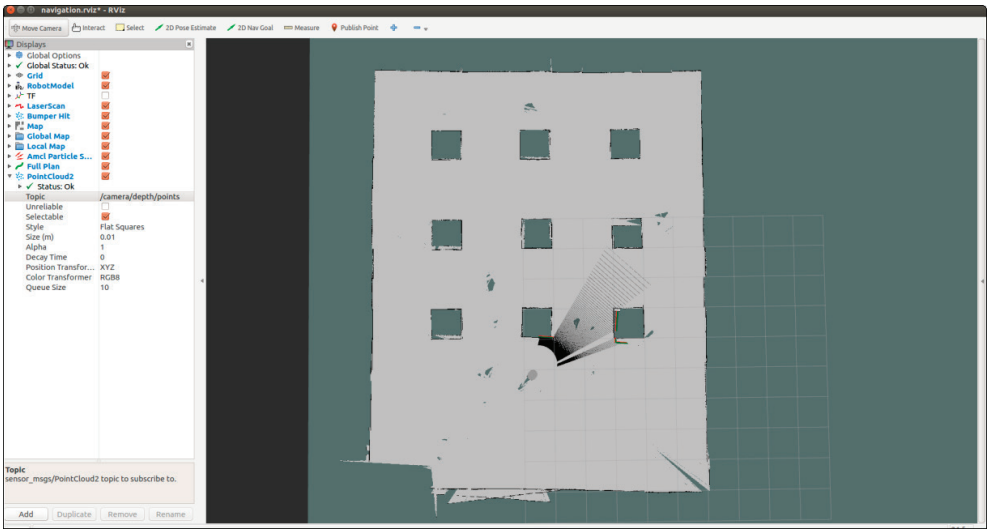
Создание карты в Rviz с помощью Gmapping

Теперь для перемещения робота мы можем использовать узел teleop. В результате мы увидим, как создается карта в Rviz. Для создания хорошей, качественной карты робот должен перемещаться с как можно меньшей скоростью и как можно чаще поворачиваться на 360°. После того как робот пройдет весь путь и карта будет создана, сохраните ее с помощью следующей команды:

```
$ rosrun map_server map_saver -f ~/Desktop/hotel
```

Карта будет сохранена с расширениями `*.pgm` и `*.yaml`, где расширение `.pgm` обозначает карту, а файл `.yaml` – конфигурацию этой карты. Согласно данной команде, карта будет сохранена на Рабочем столе.

На показанном ниже рисунке представлена карта, созданная роботом, который перемещался по окружающей среде:



Финальная карта, построенная с помощью Gmapping

Карту можно сохранить в любое время. Но прежде убедитесь, что робот охватил всю область окружения и сопоставил все пространство, как показано на предыдущем снимке экрана. Когда мы убедимся, что карта полностью построена, снова введите команду `map_saver`, после чего закройте все терминалы. Если окружающую среду отобразить не удастся, вы можете воспользоваться картой из `chefbot_gazebo/maps/hotel`.

Начало работы с адаптивным методом локализации Монте-Карло

Итак, карта окружающей среды создана. Теперь мы должны научить робота перемещаться со своего текущего положения к целевой позиции. Перед тем

как запустить автономную навигацию, следует определить положение робота на текущей карте. Алгоритм, который применяется для определения местоположения робота на карте, называется AMCL. AMCL использует многочастичный фильтр для отслеживания положения робота на карте. В нашем роботе для реализации AMCL мы используем пакет ROS (http://wiki.ros.org/amcl). Алгоритм AMCL похож на Mapping, в котором для настройки узла `amcl`, находящегося внутри пакета `amcl`, предусмотрено множество настроек. Все параметры `amcl` вы можете найти на странице Ros wiki.

Итак, настало время запустить AMCL для нашего робота. Для запуска этого алгоритма предусмотрен файл запуска, который помещен в `chefbot_gazebo/amcl_demo.launch` и `chefbot_gazebo/includes/amcl.launch.xml`.

Рассмотрим код `amcl_demo.launch`, описание которого показано в коде исполняемого файла:

```
<launch>
    <!-- Map server -->
    <arg name="map_file" default="$(find chefbot_gazebo)/maps/hotel.yaml"/>

    <node name="map_server" pkg="map_server" type="map_server" args="$(arg
map_file)" />
```

Первый узел запускает исполняемый файл `map_server` из пакета `map_server`. Узел `map_server` загружает статическую карту, которую мы ранее сохранили, и публикует ее в тему под названием карта (map) (`nav_msgs/OccupancyGrid`). Можно файл карты упомянуть как аргумент файла `amcl_demo.launch`, и при наличии такого файла узел `map_server` будет загружен; в противном случае будет загружена карта по умолчанию, расположенная в файле **c**`hefbot_gazeob/maps/hotel.yaml`.

После загрузки карты мы запускаем узел `amcl` и перемещаем базовый узел. Узел `AMCL` помогает локализовать робота на текущей карте, а узел `move_base` в стеке навигации ROS помогает в навигации робота от текущей до целевой позиции. Об узле `move_base` подробнее будет рассказано в следующих главах. В узле `move_base` также необходимо настроить параметры. Файлы параметров хранятся внутри папки `chefbot_gazebo/param`, что и показано в следующем коде:

```
    <!-- Localization -->
    <arg name="initial_pose_x" default="0.0"/>
    <arg name="initial_pose_y" default="0.0"/>
    <arg name="initial_pose_a" default="0.0"/>
    <include file="$(find chefbot_gazebo)/launch/includes/amcl.launch.xml">
        <arg name="initial_pose_x" value="$(arg initial_pose_x)"/>
        <arg name="initial_pose_y" value="$(arg initial_pose_y)"/>
        <arg name="initial_pose_a" value="$(arg initial_pose_a)"/>
    </include>
    <!-- Move base -->
    <include file="$(find
chefbot_gazebo)/launch/includes/move_base.launch.xml"/>
</launch>
```

Чтобы узнать больше о навигационном стеке ROS, перейдите по следующей ссылке: http://wiki.ros.org/navigation/Tutorials/RobotSetup.

Реализация AMCL в среде Gazebo

В этом разделе будет рассказано о том, как реализовать AMCL в создаваемом нами роботе Chefbot. Для включения AMCL в моделирование воспользуйтесь командами, приведенными ниже. Каждая команда выполняется в отдельном терминале.

Первая команда запускает имитатор Gazebo:

```
$ roslaunch chefbot_gazebo chefbot_hotel_world.launch
```

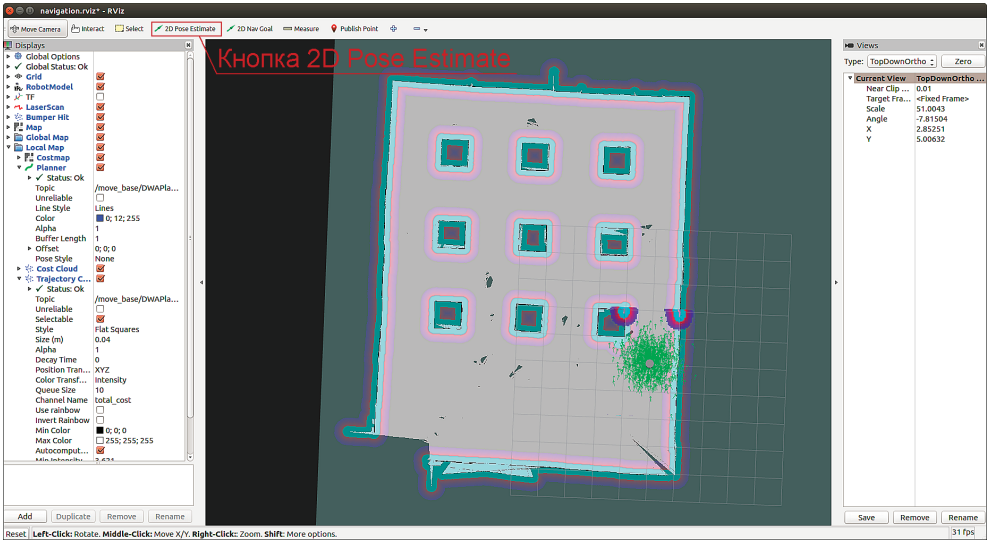
Теперь мы можем запустить файл запуска AMCL. Обратите внимание, как уже упоминалось ранее, в этом файле запуска в качестве аргумента может присутствовать файл карты. Но файл карты в качестве аргумента может и отсутствовать. Чтобы загрузить созданную ранее пользовательскую карту, выполните следующую команду:

```
$ roslaunch chefbot_gazebo amcl_demo.launch map_file:=/home/<your_user_name>/Desktop/hotel
```

Если вы хотите использовать карту, предлагаемую по умолчанию, выполните команду

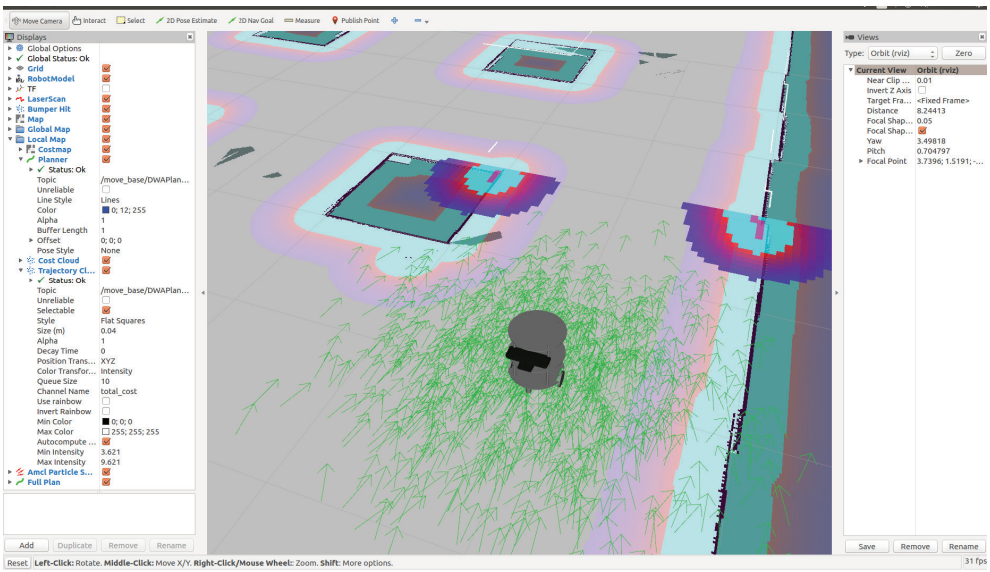
```
$ roslaunch chefbot_gazebo amcl_demo.launch
```

После того как AMCL запустится, можно начинать визуализацию карты и робота в Rviz. На экране появится окно **Rviz**, в котором будет показана карта (см. рисунок, показанный ниже). Можно увидеть карту и робота в окружении зеленых точек. Зеленые точки называются частицами `amcl`. Они указывают на неопределенность положения робота. Если концентрация частиц amcl вокруг робота высокая, значит, неопределенность положения робота очень высокая. Когда робот начнет движение, количество частиц начнет уменьшаться, и местоположение станет более точным. Если робот не может определить свое местонахождение на карте, воспользуйтесь кнопкой **2D Pose Estimate**, которая расположена на панели инструментов RViz. Далее укажите положение робота на карте вручную. Эта кнопка видна на показанном ниже рисунке.



Кнопка **2D Pose Estimate** в RViz

При приближении камеры к роботу в Rviz вы увидите частицы, как на предыдущем рисунке. Мы также сможем увидеть окрашенные в разные цвета препятствия, находящиеся вокруг робота.



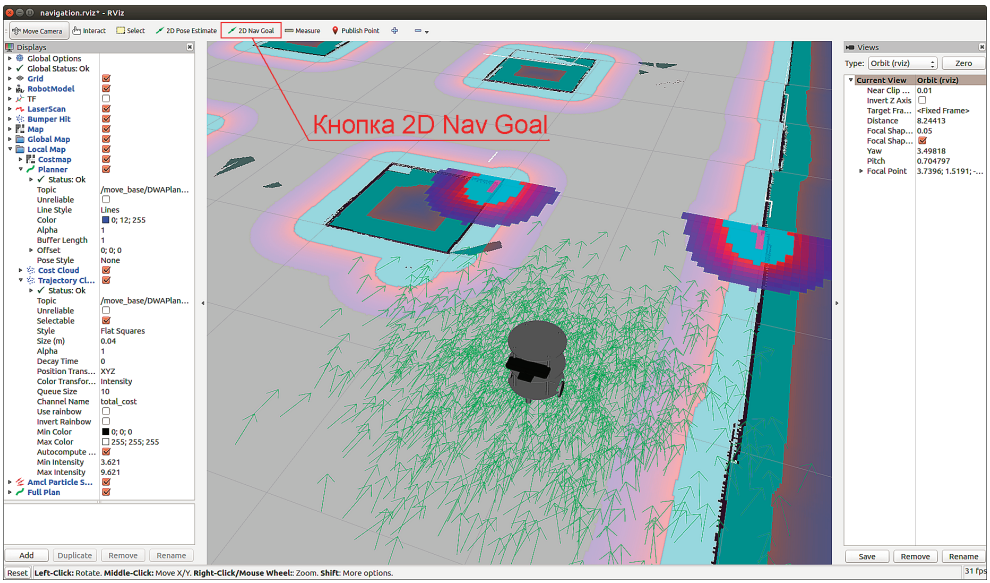
Облако AMCL вокруг робота

В следующем разделе мы узнаем, как запрограммировать Chefbot для автономного перемещения по этой карте. Вам не нужно закрывать ранее запущенные терминалы; мы можем перемещаться по роботу автономно в самой Rviz.

Автономная навигация Chefbot в отеле с использованием Gazebo

Чтобы запустить автономную навигацию робота, нам просто роботу нужно подать команду, в которой устанавливается расположенная на карте конечная точка маршрута. Ранее мы показали, где в панели инструментов RViz находится кнопка **2D Pose Estimate**. Рядом с ней вы увидите кнопку **2D Nav Goal**, назначение которой – выбор конечной точки маршрута. Нажмите на эту кнопку и щелкните мышью на конечной точке маршрута. На экране появится стрелка, указывающая на положение робота. После указания на карте конечной точки маршрута вы увидите, как робот планирует путь со своей текущей позиции к конечной точке. Робот медленно двинется со своего текущего положения в область, где расположена цель, избегая всех препятствий. На рисунке, показанном ниже, представлено планирование пути и перемещение робота в заданную позицию.

Цветная сетка вокруг робота показывает карту затраченных ресурсов и путь, запланированный роботом на обход препятствий.



Кнопка **2D Nav Goal** в панели инструментов RViz

Таким образом, если мы внутри карты зададим позицию ближе к столу, робот сможет подойти к этому столу и доставить еду, а затем вернуться в исходное положение. Чтобы не командовать роботом из RViz, можно написать узел ROS, и робот пройдет этот путь автоматически. Как это сделать, будет рассказано в следующих главах данной книги.

Итоги

В этой главе мы узнали, как имитировать нашего робота под названием Chefbot. О конструкции робота Chefbot было рассказано в предыдущей главе. Эту главу мы начали с обзора имитатора Gazebo, его различных возможностей и особенностей. После этого мы рассмотрели, как с помощью взаимодействия ROS и имитатора Gazebo выполнить визуализацию робота. Для этого нами был установлен пакет TurtleBot 2 и смоделирован Turtlebot 2 в Gazebo. Далее было создано моделирование Chefbot с использованием Gmapping, AMCL сымитирована автономная навигация робота в гостиничной среде. Теперь мы знаем, что точность визуализации робота зависит от того, насколько хорошо была сгенерирована карта. В следующей главе мы подробно рассмотрим конструкцию робота и его электронную схему.

Вопросы

1. Как мы можем визуализировать датчик в Gazebo?
2. Как ROS взаимодействует с Gazebo?
3. Какие важные теги URDF для моделирования вы знаете?
4. Что такое Gmapping, и как мы можем реализовать его в ROS?
5. В чем заключается функция узла move_base в ROS?
6. Что такое AMCL, и как мы можем реализовать его в ROS?

Дополнительная информация

Чтобы узнать больше о URDF, Xacro и Gazebo, обратитесь к книге «Освоение ROS для программирования роботов», второе издание: (https://www.packtpub.com/hardware-andcreative/mastering-ros-robotics-programming-second-edition).

Глава 5

Проектирование оборудования и схем ChefBot

В этой главе мы обсудим конструкцию, принципы работы оборудования ChefBot и выбор аппаратных компонентов. В предыдущей главе была разработана базовая структура робота и сымитирована его работа в гостинице. Имитация производилась с использованием Gazebo и ROS. Мы протестировали несколько переменных, таких как масса тела робота, крутящий момент двигателя, диаметр колеса и другие параметры. Также создали модель автономной навигации ChefBot в гостиничной среде.

Теперь, чтобы создаваемый нами робот смог ориентироваться в окружающей среде, нам необходимо выбрать все аппаратные компоненты и определиться, как все эти компоненты соединить. Мы знаем, что основная функция данного робота – навигация. Робот должен перемещаться из исходного положения в конечную точку маршрута без каких-либо столкновений с окружающей средой. В этой главе мы обсудим датчики и аппаратные компоненты, необходимые для достижения этой цели. Нами будет рассмотрена и объяснена структура робота в блок-схемах. Также мы обсудим основной принцип работы робота. Кроме того, нам потребуется выбрать компоненты, необходимые для сборки робота. Также мы проведем обзор интернет-магазинов, где можно приобрести необходимые компоненты.

Если у вас TurtleBot уже есть, эту главу вы можете пропустить. Данная глава предназначена для тех, кто оборудование для своего робота будет создавать самостоятельно. Давайте рассмотрим спецификации, соответствующие проекту нашего устройства. Робот состоит из шасси, датчиков, приводов, платы контроллера и ПК.

В этой главе мы рассмотрим следующие темы:

- структурная схема и описание робота Chefbot;
- выбор и описание компонентов робота;
- работа оборудования Chefbot.

Технические условия

В этой главе описаны компоненты, необходимые для сборки робота. Вам необходимо приобрести эти компоненты или аналогичные им.

Спецификации оборудования Chefbot

В этом разделе мы обсудим некоторые из важных спецификаций, упомянутых в главе 3 «Моделирование дифференциального привода робота». Окончательный прототип робота будет соответствовать следующим требованиям:

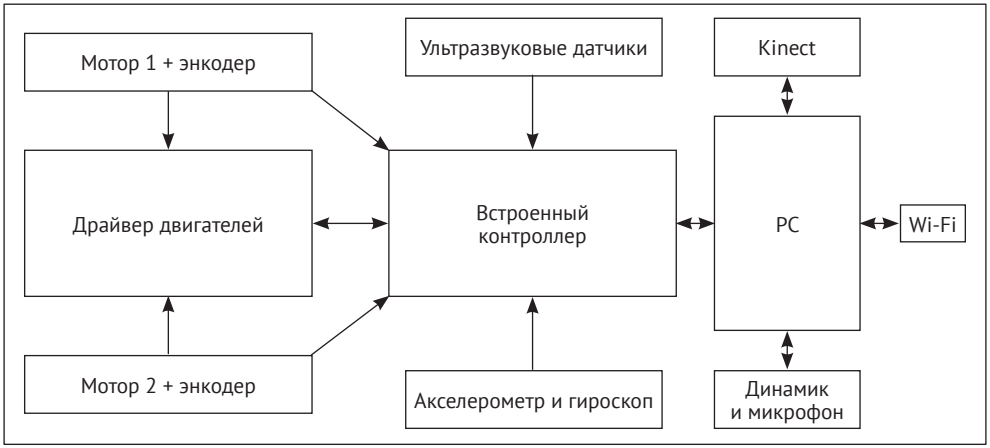
- **простая и рентабельная конструкция шасси робота**: конструкция шасси, по сравнению с существующими роботами, должна быть простой и рентабельной;
- **автономная навигация**: робот должен автономно перемещаться и определять себя в пространстве. Для этого он должен быть оснащен всеми необходимыми датчиками;
- **продолжительное время работы от батарей**: робот должен работать непрерывно при питании от батарей не менее 1 часа;
- **обход препятствий**: робот должен уметь обходить подвижные и неподвижные препятствия.

Конструкция и оборудование робота должны соответствовать всем этим требованиям. Давайте рассмотрим один из возможных вариантов соединения компонентов. В следующем разделе мы рассмотрим блок-схему робота и с ее помощью разберем принцип работы нашего робота.

Структурная схема робота

Робот приводится в движение двумя моторами постоянного тока со встроенными редукторами и энкодерами. Моторы управляются с помощью драйвера двигателей. Драйвер двигателя со встроенным контроллером с помощью команд, посылаемых моторам, контролирует движение робота. Энкодер считает количество оборотов вала двигателя и передает эти данные встроенному контроллеру. Это и есть данные одометрии робота. Кроме энкодера, к плате контроллера подключены ультразвуковые датчики. Они взаимодействуют с контроллером и определяют расстояние до препятствий. Кроме того, к плате контроллера подключены сенсоры, назначение которых – улучшение расчета данных одометрии. Встроенный контроллер подключен к персональному компьютеру, который выполняет в роботе все вычисления высокого уровня. Видеозрение и звуковые датчики подключаются ПК, а для удаленных операций предусмотрен Wi-Fi.

Каждый блок робота показан на следующей схеме.



Блок-схема конструкции робота

Двигатель и энкодер

Как уже было сказано ранее, проектируемый нами робот приводится в движение с помощью дифференциального привода. Поэтому нам потребуется два двигателя. В каждый двигатель встроен квадратурный энкодер, контролирующий вращение и обеспечивающий обратную связь (http://letsmakerobots.com/node/24031).

Квадратурный энкодер преобразует вращение двигателя и передает данные в виде квадратных импульсов; после декодирования мы получаем количество импульсов энкодера, которые могут быть использованы для обратной связи. Зная диаметр колеса и количество тактов двигателя, мы можем вычислить угол и смещение робота. Эти вычисления очень полезны для ориентирования робота.

Выбор двигателей, энкодеров и колес для робота

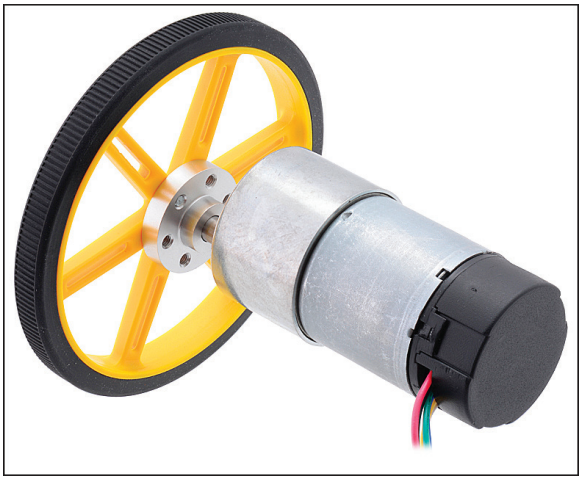
Моделируя робот, мы получили представление о его параметрах. При моделировании мы учитывали, что необходимый для стабильного управления роботом крутящий момент равен 18 кг-см. Но расчетный крутящий момент будет меньше. Чтобы иметь запас мощности, был выбран повышенный крутящий момент. Требованиям нашей конструкции удовлетворяют двигатели с сайта Pololu. Мы можем выбрать мотор постоянного тока с высоким крутящим моментом и встроенным редуктором и энкодером. Напряжение питания этого двигателя – 12 В, скорость вращения – 80 об/мин. Для нашей конструкции мы выбираем следующий двигатель: https://www.pololu.com/product/1447.

Если возникают трудности с приобретением электроники с иностранного сайта, можете воспользоваться альтернативой (сайт AliExpress):

- https://ru.aliexpress.com/wholesale?catId=0&initiative_id=SB_20180605015358&SearchText=JGA25-371;

- https://ru.aliexpress.com/item/Best-Price-DC-12V-80RPM-Self-Locking-function-Square-High-Torque-Turbo-Worm-Geared-Motor-With/32747006160.html.

Ниже показана фотография двигателя, предлагаемого на сайте Pololu. Двигатель снабжен встроенным квадратурным энкодером с разрешением 64 импульса на оборот вала двигателя, что соответствует 8400 импульсам оборота выходного вала редуктора.



Мотор постоянного тока со встроенным энкодером и колесом

Из мотора выходит 6 цветных проводов. Назначение и цвет этих проводов описаны в следующей таблице.

Цвет	**Функция**
Красный	Питание двигателя (подключается к одной клемме драйвера двигателя)
Черный	Питание двигателя (подключается к другой клемме драйвера двигателя)
Зеленый	GND (минус) энкодера
Синий	VCC (плюс) энкодера (3.5–5 В)
Желтый	Выход A энкодера
Белый	Выход B энкодера

Согласно нашей конструкции, было выбрано колесо диаметром 90 мм. Pololu предлагает колесо в 90 мм. Это колесо доступно по адресу: http://www.pololu.com/product/1439.

Альтернатива с сайта AliExpress: https://ru.aliexpress.com/wholesale?catId=0&initiative_id=SB_20180605015453&SearchText=RC+1%2F10+cartire+%D1%88%D0%B8%D1%80%D0%B8%D0%BD%D0%B0+40+%D0%BC%D0%BC+%D0%B4%D0%B8%D0%B0%D0%BC%D0%B5%D1%82%D1%80+%D1%88%D0%B8%D0%BD%D1%8B+90+%D0%BC%D0%BC.

На предыдущем рисунке это колесо показано в сборе вместе с выбранным мотором. Кроме того, для соединения колеса и двигателя нам потребуются следующие комплектующие:

- ступица для крепления колеса на вал двигателя. Деталь доступна по адресу: http://www.pololu.com/product/1083.
  Альтернатива через AliExpress: https://ru.aliexpress.com/wholesale?catId=0&initiative_id=SB_20180605015556&SearchText=UH18032+6+мм;
- кронштейн для крепления двигателя к шасси робота: http://www.pololu.com/product/1084.
  Альтернатива на AliExpress: https://ru.aliexpress.com/item/37-miniature-DC-gear-motor-bracket-Motor-mounting-seat-L-type-motor-horizontal-mounting-seat-J17236/32800410274.html.

Драйвер двигателя

Драйвер двигателя – это электрическая схема, управляющая направлением вращения двигателя. Под словосочетанием «управление двигателями» мы понимаем контроль напряжения, подаваемого на двигатели, и контроль скорости вращения и направления вращения вала двигателя. Изменение направления вращения вала двигателя производится изменением полярности на клеммах двигателя. Для этого используется схема стандартного Н-моста, применяемого в регуляторах оборотов. Н-мост представляет собой электрическую цепь, проводящую электрический ток в требуемом направлении. Сила тока, проходящая через контакты Н-моста, зависит от характеристик выбранной нами контактной группы. Следует отметить, что вместо реле можно применять электронные ключи.

Ниже показана стандартная схема Н-моста, собранная на переключателях.

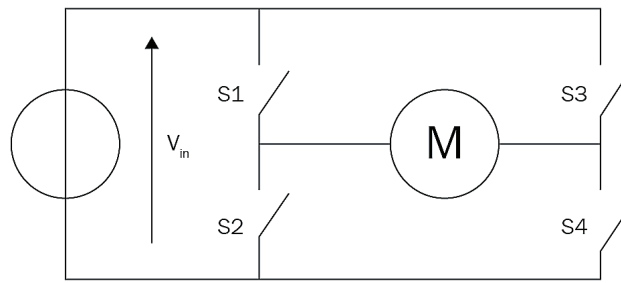


Схема Н-моста

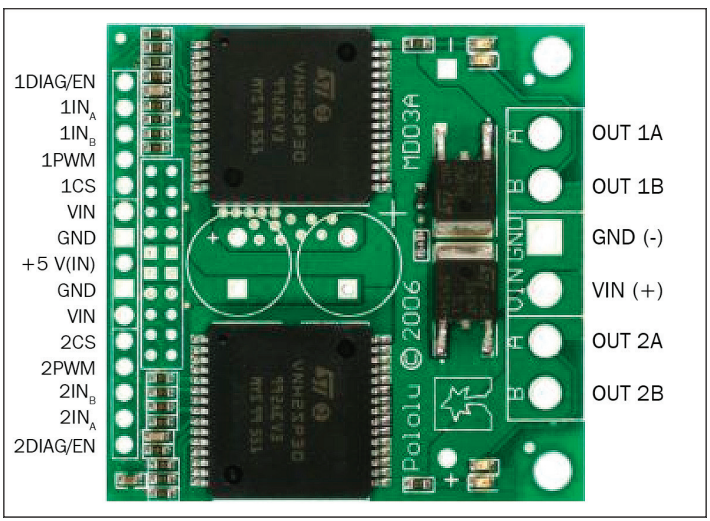
Направление вращения вала двигателя определяется включением и выключением четырех переключателей.

S1	S2	S3	S4	Результат
1	0	0	1	Вал двигателя вращается по часовой стрелке
0	1	1	0	Вал двигателя вращается против часовой стрелки
0	0	0	0	Напряжение на клеммы не подается
0	1	0	1	Клеммы двигателя замкнуты, вал не вращается
1	0	1	0	Клеммы двигателя замкнуты, вал не вращается
1	1	0	0	Короткое замыкание источника питания
0	0	1	1	Короткое замыкание источника питания
1	1	1	1	Короткое замыкание источника питания

Нами были рассмотрены варианты коммутации переключателей H-моста, с помощью которых мы управляем направлением вращения вала двигателя. Далее следует выбрать сам контроллер двигателя и обсудить принцип его работы.

Выбор контроллера двигателя

Контроллер двигателя – это плата, на которой смонтирована электронная схема управления. Данный контроллер совместим с выбранным нами двигателем и доступен на сайте Pololu. На следующем рисунке показан один из драйверов, который будет использован в нашем роботе.



Двухканальный контроллер
двигателя Dual VNH2SP30 Motor Driver Carrier MD03A

Данное устройство доступно на http://www.pololu.com/product/708.

Альтернативный вариант на AliExpress: https://ru.aliexpress.com/wholesale?catId=0&initiative_id=SB_20180605015705&SearchText=vnh2sp30.

Драйвер может управлять двумя двигателями с суммарным максимальным током потребления 30 А. Драйвер состоит из двух микросхем. Одна микросхема управляет правым двигателем, другая – левым двигателем. Описание контактов для подключения драйвера приведено ниже.

Входные контакты

Через контакты, описание которых приведено в следующей таблице, происходит управление направлением вращения и скоростью вращения вала двигателя.

Наименование контакта	Функция
1DIAG/EN, 2DIAG/EN	Контроль технического состояния драйвера двигателей 1 и 2. При нормальной работе сигнал на этих контактах отсутствует
1INa, 1INb, 2INa, 2INb	Управление направлением вращения двигателей 1 и 2. • Если INA = INB = 0, двигатель остановлен. • Если INA = 1, ИНБ = 0, вал двигателя будет вращаться по часовой стрелке. • Если INA = 0, INB = 1, вал двигателя вращается против часовой стрелки. • Если INA = INB = 1, двигатель остановлен
1PWM, 2PWM	Управление скоростью вращения двигателей 1 и 2. Управление производится кратковременной подачей напряжения на контакты двигателя
1CS, 2CS	Контакты для подключения энкодера

Выходной контакт

Выходные контакты драйвера двигателя будут управлять двумя двигателями. Ниже приведено назначение выходных контактов.

Наименование контакта	Функция
OUT 1A, OUT 1B	Контакты для подключения двигателя 1
OUT 2A, OUT 2B	Контакты для подключения двигателя 2

Контакты для подачи питания

Ниже перечислены выводы для подключения питания.

Наименование контакта	Функция
VIN (+), GND (–)	Контакты для входного напряжения питания двух двигателей. Диапазон напряжения от 5,5 В до 16 В
+5 VIN, GND (–)	Напряжение для питания платы контроллера двигателей. Напряжение должно быть равным 5 В

Встроенный контроллер

Плата контроллера – это устройство ввода/вывода, которое получает данные от ультразвукового датчика, инфракрасного датчика, энкодера, акселерометра и гироскопа, сигналы управления от бортового компьютера, обрабатывает полученные данные и отправляет управляющие импульсы на Н-мост драйвера управления двигателями.

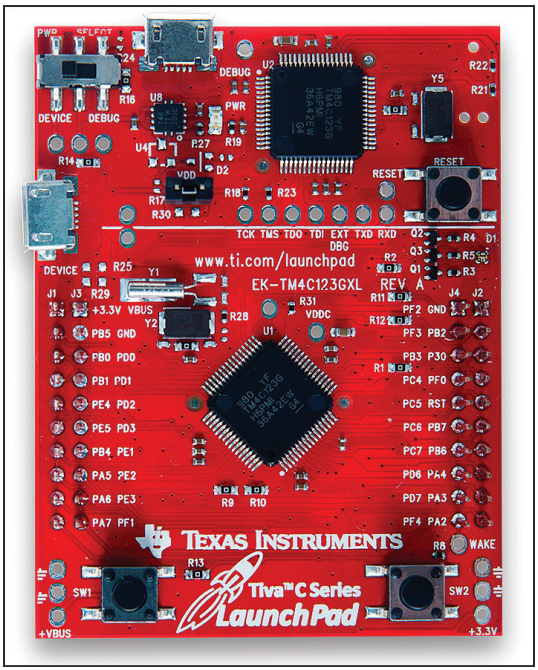
Основные функциональные возможности платы контроллера следующие:

- сопряжение драйвера двигателя и энкодера;
- сопряжение ультразвукового датчика;
- обмен данными между бортовым ПК и платой контроллера.

В следующих главах мы рассмотрим платы ввода-вывода и их взаимодействие с различными компонентами. Это такие популярные устройства, как Ардуино (https://www.arduino.cc/) или Tiva C LaunchPad (http://www.ti.com/tool/EK-TM4C123GXL) Texas Instruments. Мы для нашего робота остановим свой выбор на плате ввода-вывода Tiva C LaunchPad. Это устройство отличается от Ардуино следующими параметрами:

- Тива C launchpad содержит микроконтроллер на основе 32-разрядного ядра Cortex-M4 с 256 КБ флеш-памяти, 32 КБ ОЗУ и 80 МГц; технические характеристики Arduino гораздо ниже;
- высокая производительность обработки данных в сочетании с быстрой обработкой прерываний;
- 12 таймеров;
- 16 выходов PWM (ШИМ – широтно-импульсный модулятор);
- есть два входа для подключения квадратурных энкодеров;
- восемь универсальных асинхронных приемников/передатчиков (UART);
- контакты 5 В общего назначения ввода/вывода (GPIO);
- низкая стоимость и меньший, по сравнению с Arduino, размер платы;
- легко программируемый интерфейс IDE с названием Energia (http://energia.nu/). Код, написанный в Energia, совместим с платой Arduino.

На следующем рисунке показана плата Texas Instrument's Tiva C LaunchPad.



Плата контроллера Tiva C LaunchPad

Распиновку данной серии Texas Instrument Launchpad вы найдете, перейдя по ссылке http://energia.nu/pin-maps/guide_stellarislaunchpad/.

Данная распиновка совместима со всеми сериями контроллера на плате ввода-вывода Tiva C LaunchPad. И все они программируются в IDE Energia.

Ультразвуковые датчики

Ультразвуковые датчики, также называемые пинг-датчиками, в нашей конструкции используются для измерения расстояния между роботом и объектом. Основное применение для пинг-датчиков – обнаружить препятствие и предотвратить с ним столкновение. Ультразвуковой датчик излучает высокочастотные звуковые волны и принимает отраженное от препятствия эхо. Зная скорость движения звуковой волны и время между моментом излучения звуковой волны и моментом получения отраженного от препятствия эха, можно вычислить расстояние от робота до данного препятствия.

Одним из главных моментов в проектировании робота является условие, чтобы робот не получил повреждения от столкновений с препятствиями. В следующем разделе будут показаны числовые значения, получаемые от ультразвукового датчика. Можно установить несколько датчиков и разместить их не только в передней части робота, но и по бокам и в задней части устройства. Эти датчики позволят обнаруживать препятствия со всех сторон робота.

Датчик Kinect в основном используется для обнаружения препятствий и избежания столкновений с ними. Kinect может обнаруживать препятствия на расстоянии от 0,8 м. Поэтому пространство на расстоянии от 0 до 0,8 м может быть проконтролировано только с помощью ультразвукового датчика. Ультразвуковой датчик позволит понизить опасность столкновений робота с препятствиями.

Выбор ультразвукового датчика

Одним из самых популярных и дешевых ультразвуковых датчиков является датчик HC-SR04. Мы выбираем этот датчик для нашего робота из-за следующих факторов:

- дальность обнаружения составляет от 2 см до 4 м;
- рабочее напряжение 5 В;
- рабочий ток не более 15 мА.

Данный датчик предназначен для точного обнаружения препятствий. Как уже упоминалось ранее, напряжение питания датчика равно 5 В. Ниже показана фотография датчика HC-SR04, еще ниже – его распиновка.

Ультразвуковой датчик HC-SR04

Описание контактов ультразвукового датчика HC-SR04:

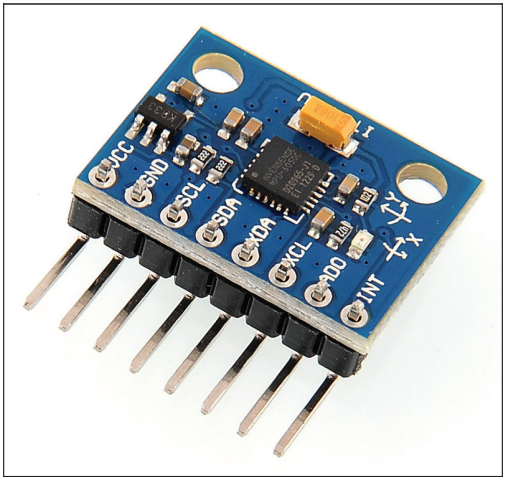
Наименование контакта	Функциональность
Vcc, GND	Контакты питания ультразвукового датчика. Для нормальной работы напряжение питания равно 5 В
Trig	Входной контакт датчика, на который подается импульс определенной продолжительности, с помощью которого формируется излучаемая ультразвуковая волна
Echo	Выходной вывод датчика. С этого контакта снимается импульс определенной продолжительности и с определенной временной задержкой по отношению к входному импульсу

Инерциальный блок измерения (акселерометр и гироскоп)

В этом роботе используется **инерциальный измерительный модуль** (**IMU**), назначение которого – получение точных значений одометрии и уточнения положения робота. **Одометрия** – это использование данных о движении приводов для оценки перемещения. Значения одометрии, получаемые только от одного датчика, могут содержать ошибки, и вследствие этих ошибок точность измерения может понизиться. Чтобы компенсировать ошибки измерения и как следствие – движения робота, мы в этом роботе будем использовать IMU. Для нашей конструкции мы выбираем блок измерения MPU 6050 для IMU по следующим причинам:

- встроенный в одну микросхему акселерометр и гироскоп;
- высокая точность и чувствительность;
- интерфейс для подключения дополнительного датчика, в нашем случае – магнитометра;
- простая установка и подключение;
- модуль MPU 6050 подключается напрямую к микропроцессору;
- напряжение питания – 3,3 В;
- для упрощенного взаимодействия MPU 6050 и LaunchPad доступны библиотеки с программным обеспечением.

На следующем рисунке показан модуль MPU 6050.



Устройство MPU 6050

Ниже приведено описание контактов данного устройства.

Контакты	Функция
VDD	Положительный контакт питания 2,3–3,4 В
GND	«Земля»
INT	Настраиваемое прерывание (в момент получения данных буфером устройства)
SCL,	Serial Clock Line (SCL) – линия синхронизации
SDA	Serial Data Line (SDA) – линия передачи данных I2C
ASCL, ASDA	Дополнительный I2C-интерфейс для подключения внешнего магнитометра
AD0	I2C-адрес; по умолчанию AD0 подключен к земле. В этом случае адрес устройства — 0x68; если подключить AD0 к контакту питания, то адрес изменится на 0x69

Модуль можно купить здесь: https://amperkot.ru/products/modul_giroskopa_akselerometr_trehosnyiy_mpu6050_6dof_gy521/23867419.html.

Kinect/Orbbec Astra

Kinect – это бесконтактный сенсорный игровой контроллер. Состоит из видеокамеры и двух сенсоров, расположенных в верхней части устройства. Мы используем Kinect для организации трехмерного зрения робота. Робот с помощью Kinect будет получать трехмерный образ окружающей его среды. Трехмерные изображения преобразуются в мелкие точки, называемые облаками точек (point cloud). Данные облака точек будут иметь все 3D-параметры окружающей среды.

В основном в роботе Kinect используется вместо лазерного сканера. Данные лазерного сканирования алгоритм SLAM использует для строительства карты окружающей среды. Лазерный сканер – это очень дорогое устройство. Поэтому вместо лазерного сканера мы применим Kinect как виртуальный лазерный сканер.

Kinect

Основные датчики, используемые в Kinect, – это ИК-камера, ИК-прожектор и RGB-камера. С помощью ИК-камеры, ИК-прожектора и RGB-камеры генерируется трехмерное облако точек окружающей среды. Также Kinect оснащен микрофонами и моторизированной станиной, обеспечивающей наклон датчика.

Kinect можно приобрести здесь: https://www.ebay.com/sch/-/117044/i.html?_nkw=kinect+xbox+360&Brand=Microsoft&_dcat=117044.

Orbbec Astra

Альтернативой Kinect является Orbbec Astra (https://orbbec3d.com/product-astra/). Astra работает с тем же программным обеспечением, которое использует Kinect. Преобразование облака точек в данные лазерного сканирования выполняется этим же программным обеспечением, поэтому при использовании Astra нам необходимо только заменить драйвер устройства. После создания карты робот может перемещаться по окружающей среде.

Центральный процессор

Управление роботом осуществляется с помощью бортового ПК, на котором выполняется навигационный алгоритм. В качестве бортового компьютера можно использовать ноутбук, мини-ПК или ноутбук. Компания Intel выпускает мини-компьютер под названием Intel **Next Unit of Computing** (**NUC**). Этот компьютер имеет очень маленький размер, малый вес и благодаря процессору Intel Celeron Core i3 или Core i5 обладает хорошими вычислительными возможностями. Компьютер работает с объемами оперативной памяти до 16 ГБ и имеет встроенный модуль Wi-Fi/Bluetooth. Мы выбираем Intel NUC потому, что он обладает хорошей производительностью, очень маленькими размерами. Кроме того, он очень легкий. Мы не будем рассматривать популярные микрокомпьютеры, такие как Rasberry pi (http://www.raspberrypi.org/) или BeagleBone (http://beagleboard.org/), потому что вычислительные мощности этих микрокомпьютеров ниже, чем у Intel Next Unit of Computing.

NUC использует процессор Intel DN2820FYKH. Далее представлена детализация этого компьютера:

- процессор Intel Celeron Dual Core с тактовой частотой 2,39 ГГц;
- 4 ГБ оперативной памяти;
- жесткий диск 500 ГБ;
- интегрированная графика Intel;

- разъем наушников/микрофона.
- питание 12 В.

На следующем рисунке показан мини-компьютер Intel NUC.

Intel NUC DN2820FYKH

Вы можете купить NUC на Amazon: http://a.co/2F2flYl.

Учтите, модель NUC, ссылка на приобретение которой указана выше, устаревшая. Если этого микрокомпьютера в продаже уже нет, вы можете попробовать подобрать похожий по техническим требованиям NUC, используя следующие ссылки:

- **Intel NUC BOXNUC6CAYH** (https://www.intel.com/content/www/us/en/products/boards- kits/nuc/kits/nuc6cayh.html);
- **Intel NUC KIT NUC7CJYH** (https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc7cjyh.html);
- **Intel NUC KIT NUC5CPYH** (https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc5cpyh.html);
- **Intel NUC KIT NUC7PJYH** (https://www.intel.com/content/www/us/en/products/boards- kits/nuc/kits/nuc7pjyh.html).

Динамики/микрофон

Основная функция робота – автономная навигация. Мы добавим дополнительную функцию, при которой робот может взаимодействовать с пользователями с помощью речи. Робот может получать голосовые команды и отвечать пользователю с помощью синтезатора речи (TTS). Синтезатор речи преобразовывает текст в речевой формат (звук). Для получения голосовых команд и для ответов пользователю роботу необходимы микрофон и динамики. Это стандартные аппаратные средства. Единственное к ним требование – возможность подключения к компьютеру через USB. Динамики и микрофон можно заменить Bluetooth-гарнитурой.

Источник питания/аккумулятор

Одним из наиболее важных компонентов является питание. В техническом задании было обусловлено, что робот должен непрерывно работать не менее часа. Желательно, чтобы напряжение, отдаваемое аккумуляторной батареей, соответствовало напряжению питания всех компонентов компьютера. Немаловажную роль играют габаритные размеры и вес аккумуляторной батареи. Чем меньше вес и габариты при заданной емкости, тем лучше. И еще одно немаловажное требование, которое следует учесть при выборе источника питания. Это максимальный ток батареи. Максимальный ток, необходимый для всей цепи, не должен превышать максимальный ток источника питания робота. В нашем случае – аккумуляторной батареи. Максимальный ток и напряжение питания компонентов робота приведены в следующей таблице.

Компоненты	**Максимальный ток (амперы)**
Intel NUC PC	12 В, 3 А
Kinect	12 В, 1 А
Моторы	12 В, 0,7 А
Драйверы моторов, IMU, ультразвуковой датчик, микрофон и колонки	5 В, < 0,5 А

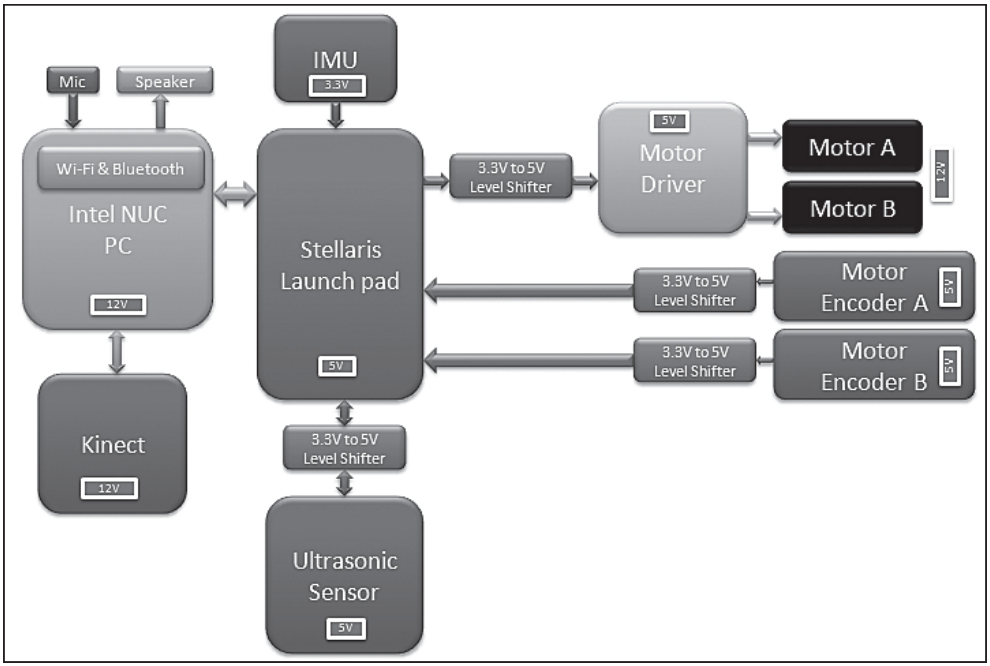
Чтобы соответствовать этим требованиям, мы для нашего робота выбираем аккумуляторную батарею 12 В, 10 А·ч.

Загерметизированная свинцовокислотная батарея

Мы эту батарею можем купить по адресу http://a.co/iOaMuZe. Вы можете выбрать другую батарею. Основное требование – напряжение питания 12 В, емкость не менее 10 А·ч, максимальный ток батареи – не менее 5,5 А.

Как работает оборудование ChefBot?

Мы можем объяснить работу оборудования ChefBot, используя следующую блок-схему. Это более детализированная версия нашей первой блок-схемы. В ней указываются напряжение питания для каждого компонента и взаимосвязь между этими компонентами.



Детализированная блок-схема оборудования Chefbot

Основная цель этой главы состоит в том, чтобы разработать аппаратное обеспечение для ChefBot со всеми аппаратными компонентами и изучить взаимосвязи между ними. Как уже говорилось ранее, основная задача этого робота – автономная навигация. Поэтому все аппаратное обеспечение оптимизировано на выполнение этой задачи.

Ходовая часть робота основана на системе дифференциального привода, состоящего из двух двигателей и двух колес. Для поддержки робота предусмотрены опорные колеса. Моторы перемещают робот в любом направлении в двухмерной плоскости, регулируя скорость перемещения и направление движения.

Для управления скоростью и направлением вращения колес используется драйвер двигателя. Выбранный нами драйвер двигателя одновременно управляет двумя двигателями, изменяя скорость вращения и направление вращения каждого двигателя.

Драйвер двигателя подключен к микроконтроллеру Tiva C Launch Pad, который вырабатывает команды для управления двигателями дифференциального привода. Драйвер двигателя взаимодействует с Launch Pad с помощью преобразователя напряжения, который преобразует напряжение от 3,3 до 5 В и наоборот. Мы используем этот преобразователь, потому что рабочее напряжение драйвера двигателя равно 5 В, а напряжение на контактах микроконтроллера LaunchPad равно 3,3 В.

Каждый двигатель оборудован встроенным датчиком обратной связи, вырабатывающим импульсы в зависимости от скорости и направления вращения вала двигателя. Этот датчик называется энкодером. Энкодер – датчик, преобразующий подконтрольную величину в электрический сигнал. Подсчитывая количество полученных от энкодера импульсов, можно высчитать количество оборотов, произведенных двигателем, и рассчитать текущее положение робота. Датчики подключаются к LaunchPad с помощью преобразователей напряжения.

Остальные датчики, такие как ультразвуковой датчик и IMU, также взаимодействуют с LaunchPad. Ультразвуковой датчик предназначен для обнаружения объектов, которые расположены рядом с роботом и которые не способен обнаружить сенсор IMU, для уточнения данных используется совместно с энкодерами. Все значения датчиков поступают на LaunchPad и отправляются через USB на бортовой ПК. LaunchPad получает все значения датчиков и отправляет их на обработку бортовому ПК.

Кроме LaunchPad, бортовой компьютер взаимодействует с Kinect, микрофоном и динамиком. На бортовом персональном компьютере установлен ROS, который получает данные с Kinect и преобразует их в данные, эквивалентные данным лазерного сканирования. С помощью этих данных SLAM создает карту окружающей среды. Динамик и микрофон используются для связи пользователя и робота.

Команды для управления двигателями, созданные в узлах ROS, отправляются на LaunchPad. После обработки в LaunchPad эти команды отправляются в широтно-импульсный модулятор и далее поступают на схему драйвера двигателей.

Далее мы обсудим в деталях взаимодействие каждого компонента робота и программное обеспечение, необходимое для взаимодействия этих компонентов.

Итоги

В этой главе мы рассмотрели особенности конструкции разрабатываемого нами робота. Главная задача данного робота – это автономная навигация. Робот должен перемещаться в окружающей его среде, анализируя все препятствия с помощью данных, получаемых от датчиков. Нами были рассмотрены блок-схема и назначение каждого блока. Далее нами были выбраны необходимые компоненты, отвечающие всем техническим требованиям. Также были предложены альтернативные компоненты. В следующей главе будет представ-

лена более подробная информация о приводах, используемых в конструкции, и их взаимодействии.

Вопросы

1. Что такое блок-схема робота?
2. Каковы принцип работы и основные функции схемы H-моста?
3. Назовите основные компоненты алгоритма навигации робота.
4. Какие критерии необходимо учитывать при выборе компонентов для робота?
5. Каковы основные функции Kinect в этом роботе?

Дополнительная информация

Дополнительную информацию о контроллере Tiva-C LaunchPad вы можете найти по следующей ссылке: http://processors.wiki.ti.com/index.php/Getting_Started_with_the_TIVA%E2%84%A2C_Series_TM4C123G_LaunchPad.

Глава 6

Согласование приводов и датчиков с контроллером робота

В предыдущей главе мы обсудили выбор аппаратных компонентов, необходимых для сборки нашего робота. Основные компоненты – это датчики и привод робота. Приводы обеспечивают мобильность робота, а датчики предоставляют информацию об окружающей среде.

В этой главе мы уделим особое внимание различным типам приводов и датчиков, которые мы собираемся использовать в этом роботе, и их сопряжению с Tiva C LaunchPad. Tiva C LaunchPad – это 32-битный микроконтроллер ARM от Texas Instruments, работающий на частоте 80 МГц.

Начнем главу с обсуждения приводов. Рассматриваемый нами привод – это двигатель постоянного тока, совмещенный с редуктором и энкодером. Редуктор предназначен для уменьшения частоты вращения ведущего вала и увеличения крутящего момента. Этот вид двигателей очень экономичен и удовлетворяет нашим требованиям к конструкции робота.

В первом разделе данной главы мы рассмотрим конструкцию системы привода робота. Система привода робота состоит из двух двигателей – редукторов постоянного тока с энкодерами и драйвером двигателя. Драйвер двигателя контролирует Tiva C LaunchPad. Мы рассмотрим сопряжение драйвера двигателя и квадратурного энкодера с LaunchPad Tiva C. Далее мы рассмотрим некоторые из новейших приводов, которые могут заменить существующий двигатель постоянного тока с редуктором и энкодером. Если роботу будет необходимо перевозить более тяжелую полезную нагрузку или потребуется повышенная точность, следует использовать более новый привод.

И в конце главы нами будут рассмотрены различные датчики, обычно используемые в мобильных роботах.

В этой главе мы рассмотрим:

- подключение двигателя постоянного тока с редуктором к Tiva C к LaunchPad;
- взаимодействие квадратурного энкодера с Tiva C LaunchPad;
- объяснение кода взаимодействия;
- взаимодействие приводов Dynamixel;
- сопряжение ультразвуковых датчиков и ИК-датчиков сближения;
- взаимодействие инерциальных измерительных блоков (IMUs).

Технические условия

Вам понадобятся компоненты оборудования робота и установленный в Ubunru 16.04 LTS компонент под названием **Energia IDE**. Energia – это инструмент редактирования кода с открытым исходным кодом, разрабатываемый сообществом для LaunchPad.

Согласование редукторного двигателя постоянного тока с Tiva C LaunchPad

В предыдущей главе нами был выбран двигатель постоянного тока со встроенным редуктором и встроенным энкодером от Pololu (возможно использование альтернативного варианта) и мы подключали к микроконтроллеру Tiva C LaunchPad. Нам для взаимодействия двигателей с LaunchPad понадобятся следующие компоненты:

- два двигателя с металлическим редуктором pololu 37Dx57L mm с 64 импульсами, вырабатываемыми энкодером за 1 оборот вала;
- колеса Pololu размером 90×10 мм и ступица для соединения колеса и вала редуктора;
- сдвоенный драйвер двигателя VNH2SP30, MD03A с сайта Pololu (или альтернатива);
- герметичный свинцово-кислотный или литий-ионный аккумулятор напряжением 12 В;
- преобразователь логических уровней от 3,3 до 5 В (https://www.sparkfun.com/products/12009);
- Tiva C LaunchPad и совместимый интерфейс.

На следующем рисунке показана схема взаимодействия двух двигателей с помощью H-моста.

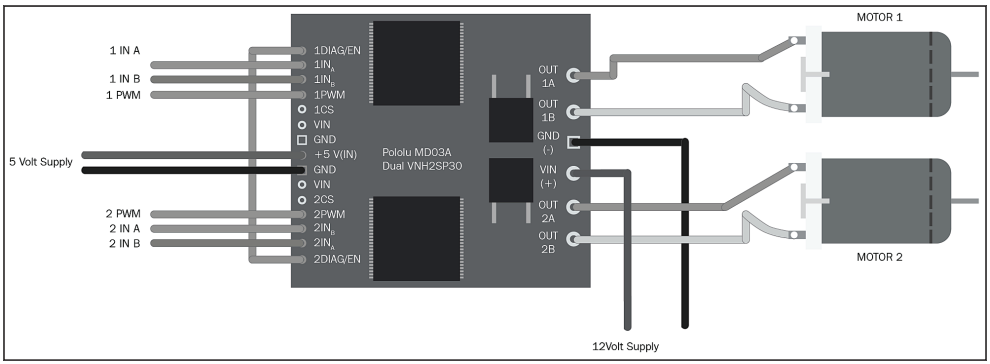


Схема взаимодействия двух электродвигателей

Для взаимодействия двигателей с LaunchPad нам необходимо соединить микроконтроллер с двумя двигателями ходовой части робота. Рабочее напряжение драйвера двигателя равно 5 В, а контроллер LaunchPad вырабатывает напряжение 3,3 В. Потому для согласования микроконтроллера и двигателей мы должны подключить двигатели к микроконтроллеру с помощью преобразователей, поэтому мы должны подключаться к уровню переключения, как показано на следующей схеме.

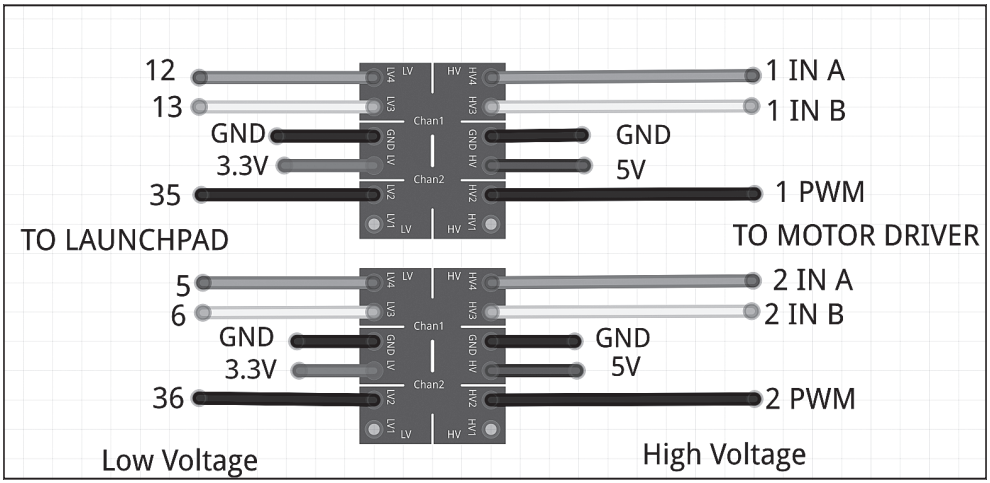


Схема переключения уровня

Оба редукторных двигателя постоянного тока подключены к драйверу двигателя через контакты OUT1A, OUT1B и OUT2A, OUT2B. Питающее напряжение двигателей подается через контакты VIN (+) и GND (–). Эти двигатели постоянного тока работают от напряжения 12 В, поэтому мы и подаем входное на-

пряжение, равное 12 В. Драйвер двигателя обеспечивает входное напряжение в диапазоне от 5,5 до 16 В.

Первый контакт 1DIAG/EN в большинстве случаев остается неподключенным (см. верхнюю схему подключения двух двигателей к Polou MD03A Dual VNH2SP30). Эти контакты предназначены для управления (включения или выключения) H-моста и контроля его состояния. Сигналы управления направлением вращения двигателя поступают на контакты 1INA и 1INB. Сигнал на включение или отключение двигателя поступает на контакт 1PWM. Скорость вращения двигателя контролируется его кратковременным включением и отключением. Контрольные данные с датчика тока поступают на контакт CS. При выходном токе, равном 1 А, на контакте CS появится сигнал с напряжением 0,13 В. Контакты VIN (напряжение питания 12 В и GND (земля)) не используются. Питание драйвера двигателя снимается с контактов +5 В (IN) и GND. Управление драйверами каждого двигателя поступает на отдельные контакты. Так, сигналы для управления направлением вращения первого двигателя поступают на контакты 1INA и 1INB, а сигналы для управления направлением вращения второго двигателя подаются на контакты 2INA и 2INB.

В следующей таблице показана таблица истинности комбинаций входных и выходных данных.

INA	INB	DIAGA/ENA	DIAGA/ENA	OUTA	OTUB	CS	Режим работы
1	1	1	1	H	H	Высокий Imp	Останов, Vcc
1	0	1	1	H	L	Значение тока I = I_{out}/K, I_{out} – выходное значение тока	Вращение по часовой стрелке
0	1	1	1	L	H	Значение тока I = I_{out}/K, I_{out} – выходное значение тока	Вращение против часовой стрелки
0	0	0	0	L	L	High Imp	Останов, GND

Управляя двигателями с помощью этих сигналов, мы можем перемещать робота в любую точку окружающего пространства. А управляя скважностью импульсов (широтно-импульсная модуляция), мы управляем скоростью вращения двигателя. Это основная логика управления двигателем постоянного тока с помощью H-моста.

Для взаимодействия драйвера двигателей с LaunchPad необходим преобразователь логических уровней 3,3–5 В. Это объясняется тем, что напряжение на контактах LaunchPad равно 3,3 В, а напряжение на контактах драйвера двигателя равно 5 В. Поэтому нам необходимо с помощью логического преобразователя преобразовать сигнал с 3,3 до 5 В и наоборот.

В дифференциальном приводе используется два двигателя. Далее мы обсудим работу дифференциального привода.

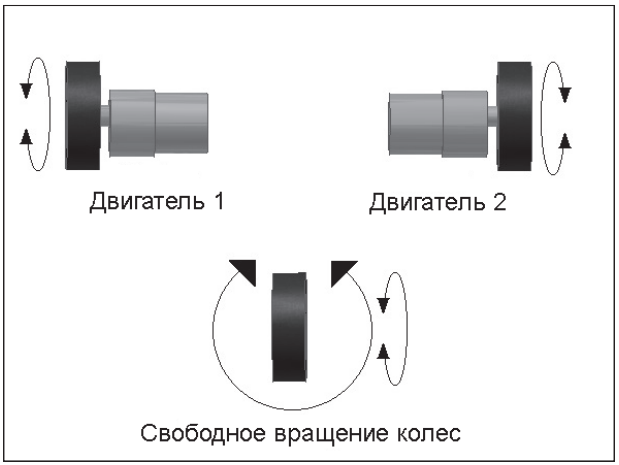
Дифференциальный привод колесного робота

Как мы уже многократно упоминали, разрабатываемый нами робот – робот с дифференциальным колесным приводом. В дифференциальном колесном

роботе механизм управления основан на независимом вращении двух отдельно управляемых колес, расположенных по обе стороны корпуса робота. Он может менять направление движения, используя разность в скорости вращения колес. Поэтому ему не требуется рулевое управление.

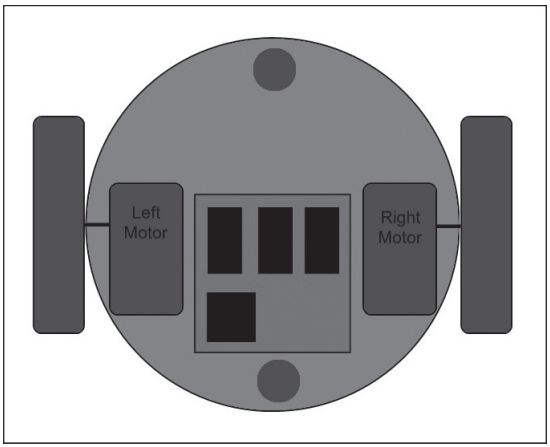
Для поддержки корпуса робота используются поддерживающие колеса.

На следующем рисунке показано типичное представление дифференциального привода.



Дифференциальный колесный робот

Если валы двигателей будут вращаться в одном направлении, робот будет двигаться вперед или назад. Если скорость вращения одного двигателя будет превышать скорость вращения второго двигателя, робот будет совершать поворот в сторону двигателя с меньшей скоростью вращения. Если правый двигатель остановлен, а левый работает, то робот будет поворачивать вправо, для левого поворота необходимо остановить левый двигатель, а вал редуктора правого двигателя должен вращаться по часовой стрелке. На следующем рисунке показана установка двигателей нашего робота. Два двигателя смонтированы на противоположных сторонах опорной пластины. К ведущим валам редукторов каждого двигателя крепятся ведущие колеса. Для поддержки робота используются опорные колеса.



Вид на основание робота сверху

Далее, согласно таблице истинности, используя LaunchPad, мы можем запрограммировать драйвер двигателя. Программирование осуществляется с помощью IDE под названием Energia (http://energia.nu/). LaunchPad программируется с помощью языка Wiring (http://wiring.org.co/).

Установка Energia IDE

Последнюю версию Energia можно скачать по следующей ссылке: http://energia.nu/download/. Далее мы рассмотрим порядок установки на 64-разрядную Ubuntu 16.04.LTS. Используемая нами версия Energia – 0101E0018.

1. Загрузите Energia для Linux 64-разрядной версии по ссылке выше.
2. Извлеките Energia из архива в домашнюю папку пользователя.
3. Инструкции по установке платы микроконтроллера Tiva C приведены на сайте http://energia.nu/guide/guide_linux/.
4. Вы должны загрузить файл `71-ti-permissions.rules` по следующей ссылке: http://energia.nu/files/71-ti-permissions.rules.
5. Файл правил даст разрешение пользователю на чтение и запись в LaunchPad. Вы должны сохранить файл под именем `71-ti-permissions.rules` и, чтобы скопировать файлы правил в системную папку для получения разрешения, выполнить следующую команду из текущего пути:

   ```
   $ sudo mv 71-ti-permissions.rules /etc/udev/rules.d/
   ```

6. После копирования файла выполните команду для активации правила:

   ```
   $ sudo service udev restart
   ```

7. Теперь вы можете подключить плату микроконтроллера Tiva C LaunchPad к компьютеру и в терминале Linux выполнить команду `dmesg`. Эта команда позволит просмотреть журнал ядра Linux. Если плата микро-

контроллера подключена, то устройство последовательного порта (COM) в конце сообщения отобразит строку (см. следующий рисунок).

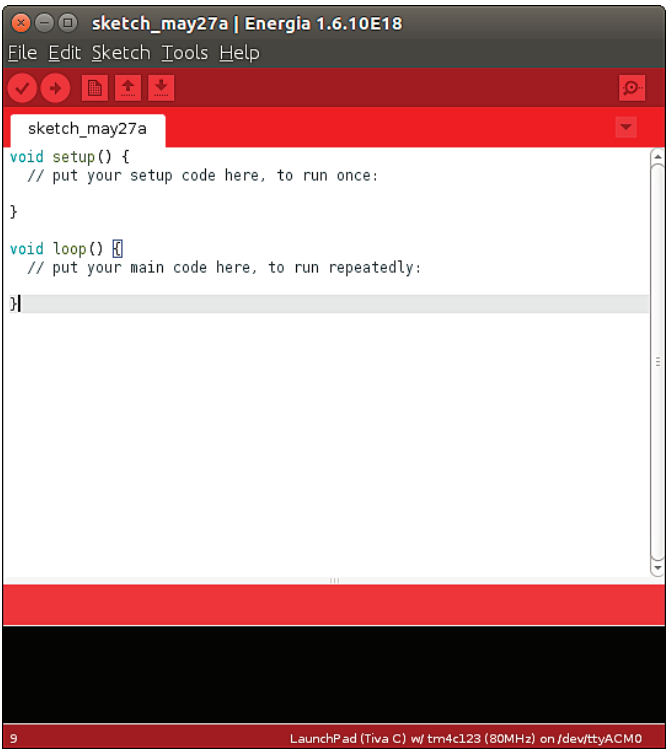
```
[  569.441209] usb 1-5: New USB device found, idVendor=1cbe, idProduct=00fd
[  569.441215] usb 1-5: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[  569.441218] usb 1-5: Product: In-Circuit Debug Interface
[  569.441222] usb 1-5: Manufacturer: Texas Instruments
[  569.441225] usb 1-5: SerialNumber: 0E2258F8
[  569.461748] cdc_acm 1-5:1.0: ttyACM0: USB ACM device
[  569.461943] usbcore: registered new interface driver cdc_acm
[  569.461944] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters
```

Сообщение в терминале о подключении платы микроконтроллера к COM-порту

8. Если в терминале появилось сообщение о том, что микроконтроллер к COM-порту подключен, используя следующую команду, запустите Energia:

   `$./energia`

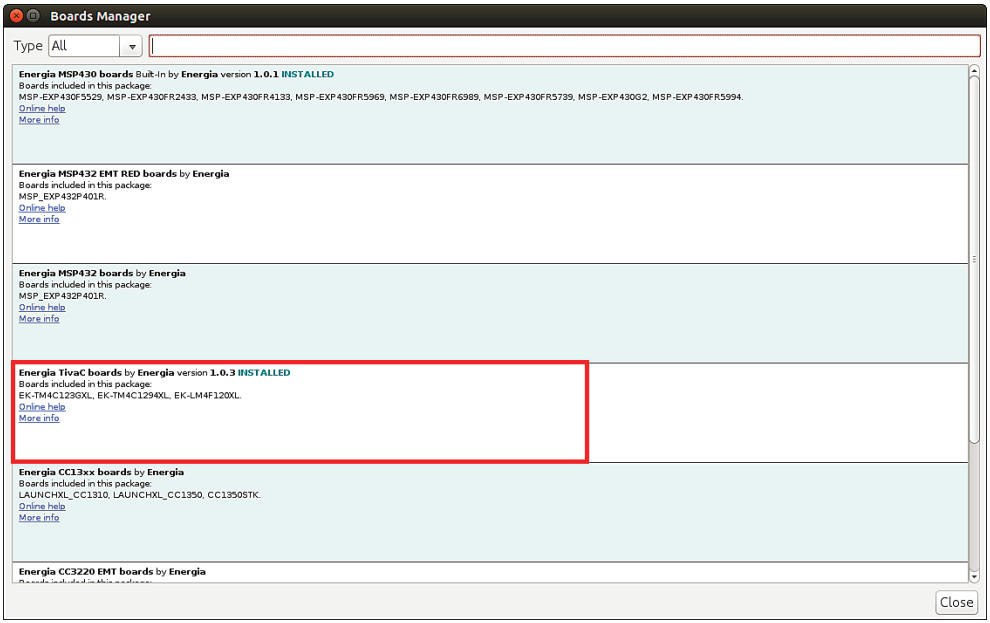
   Запуск Energia IDE показан на следующем рисунке.



Energia IDE

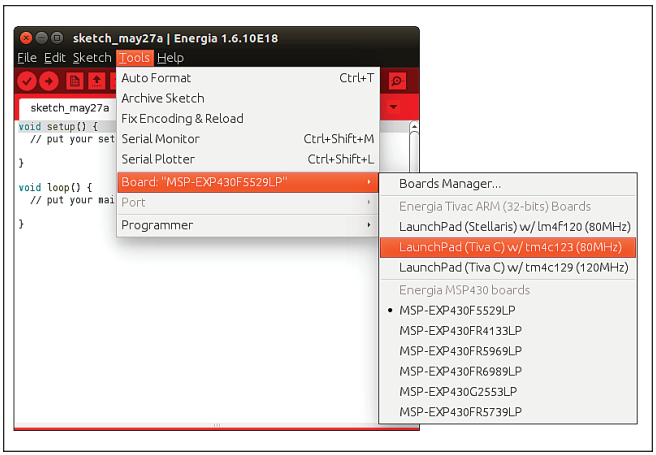
Теперь нам нужно выбрать плату tm4c123 в IDE для компиляции специального кода для этой платы. Чтобы выбрать плату tm4c123, следует установить ее пакеты.

Для установки пакетов выберите опцию **Tools → Boards → Boards Manager**.



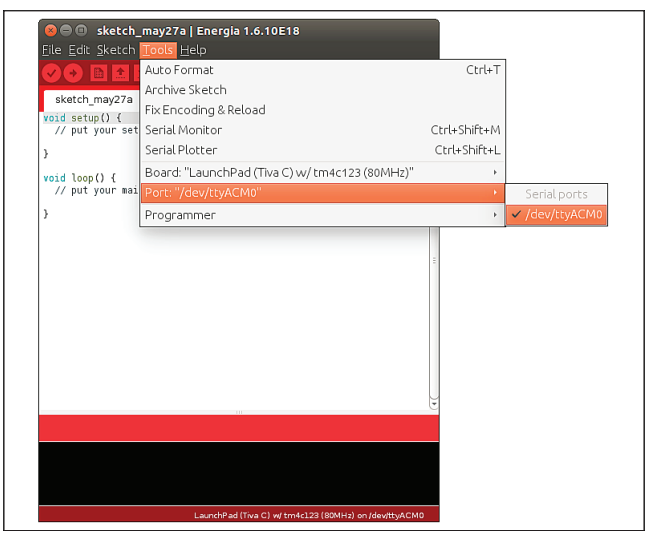
Панель управления Energia

9. Выберите вкладку **Tools → Boards → Launchpad (Tiva C) w/tm4c123 (80MHz)** и, как показано на следующем рисунке, микроконтроллер.



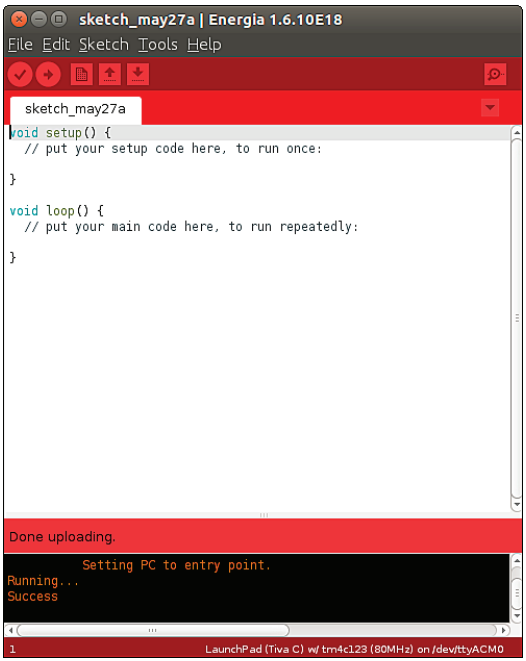
Выбор микроконтроллера

10. Перейдите в меню **Tools** → **Serial Port** → **/dev/ttyACM0** и выберите последовательный порт.



Выбор последовательного порта

11. Скомпилируйте и загрузите код с помощью кнопки **Upload**. Кнопка **Upload** выполнит оба процесса. Следующий снимок экрана иллюстрирует успешную загрузку.



Компиляция и загрузка прошли успешно

Больше сведений об установке Energia на Linux, Mac OS X и Windows вы найдете по следующим ссылкам:

- http://energia.nu/guide/guide_linux/ для Linux;
- http://energia.nu/Guide_MacOSX.html для Mac OS X;
- http://energia.nu/Guide_Windows.html для Windows.

Код взаимодействия с двигателями

Следующий код используется для тестирования двух двигателей в дифференциальном приводе. Этот код перемещает робот в течение 5 с вперед и в течение 5 с назад. Затем робот в течение 5 с перемещается влево, далее в течение 5 с вправо. После каждого движения робот останавливается на 1 с.

В начале кода мы определяем контакты для INA, INB и PWM двух двигателей следующим образом:

```
///Пины левого мотора
#define INA_1 12
#define INB_1 13
#define PWM_1 PC_6

///Пины правого мотора
#define INA_2 5
#define INB_2 6
```

```
#define PWM_2 PC_5
```

Следующий код содержит пять функций для перемещения робота вперед, назад, влево и вправо. Пятая функция – остановка робота. Мы будем использовать функцию записи цифрового значения PIN-кода `digitalWrite()`. Первый аргумент `digitalWrite()` является номером пина, а второй аргумент – это значение, которое должно быть записано для пина. Значение может быть ВЫСОКИМ или НИЗКИМ. Мы будем использовать функцию `analogWrite()` для записи значения пину PWM. Первым аргументом этой функции является число пинов, а вторым – значение ШИМ. Диапазон этого значения – от 0 до 255. При высокой частоте ШИМ драйвер двигателя включает и отключает двигатель с высокой частотой. Поэтому двигатель будет вращаться с большой скоростью. На низкой частоте ШИМ частота переключения драйвера двигателя будет низкой, и скорость вращения вала тоже будет низкой. Текущее значение скорости вращения будет высоким.

```
voidmove_forward()
{
    //Установим вращение левого мотора по часовой стрелке, а вращение правого мотора против
    // часовой стрелки
    //Левый мотор
    digitalWrite(INA_1,HIGH);
    digitalWrite(INB_1,LOW);
    analogWrite(PWM_1,255);
    //Правый мотор
    digitalWrite(INA_2,LOW);
    digitalWrite(INB_2,HIGH);
    analogWrite(PWM_2,255);
}

//////////////////////////////////////////////////////

voidmove_left()
{
    //Левый мотор
    digitalWrite(INA_1,HIGH);
    digitalWrite(INB_1,HIGH);
    analogWrite(PWM_1,0);
    //Правый мотор
    digitalWrite(INA_2,LOW);
    digitalWrite(INB_2,HIGH);
    analogWrite(PWM_2,255);
}

//////////////////////////////////////////////////////

voidmove_right()
{
    //Левый мотор
    digitalWrite(INA_1,HIGH);
    digitalWrite(INB_1,LOW);
    analogWrite(PWM_1,255);
    //Правый мотор
```

```
    digitalWrite(INA_2,HIGH);
    digitalWrite(INB_2,HIGH);
    analogWrite(PWM_2,0);
}
//////////////////////////////////////////////////////
void stop()
{
    //Левый мотор
    digitalWrite(INA_1,HIGH);
    digitalWrite(INB_1,HIGH);
    analogWrite(PWM_1,0);
    //Правый мотор
    digitalWrite(INA_2,HIGH);
    digitalWrite(INB_2,HIGH);
    analogWrite(PWM_2,0);
}
//////////////////////////////////////////////////
voidmove_backward()
{
    //Левый мотор
    digitalWrite(INA_1,LOW);
    digitalWrite(INB_1,HIGH);
    analogWrite(PWM_1,255);
    //Правый мотор
    digitalWrite(INA_2,HIGH);
    digitalWrite(INB_2,LOW);
    analogWrite(PWM_2,255);
}
```

Контакты INA и INB для обоих двигателей – это входные контакты. Поэтому на этих контактах может присутствовать как высокое, так и низкое значение (логическая единица или логический ноль). Функция `pinMode()` используется для установки режима ввода/вывода. Первый аргумент `pinMode()` – номер контакта, второй аргумент – это режим. Мы можем контакт назначить как входной, так и выходной. Чтобы назначить контакт как выходной, запишите `OUTPUT` (выходной аргумент) в качестве второго аргумента; чтобы назначить контакт как входной, запишите `INPUT` (входной аргумент) в качестве второго аргумента. Это показано в следующем коде. Нет необходимости назначать контакт PWM в качестве выходного, потому что `analogWrite()` назначает без установки режима вывода `pinMode()`:

```
void setup()
{
    //Установим пины левого мотора на приём сигнала - OUTPUT
    pinMode(INA_1,OUTPUT);
    pinMode(INB_1,OUTPUT);
    pinMode(PWM_1,OUTPUT);

    //Установим пины правого  мотора на приём сигнала – OUTPUT
```

```
    pinMode(INA_2,OUTPUT);
    pinMode(INB_2,OUTPUT);
    pinMode(PWM_2,OUTPUT);
}
```

Следующий фрагмент является основным циклом кода. Он в течение 5 с вызывает каждую функцию: `move_forward()`, `move_backward()`, `move_left()` и `move_right()`. После вызова каждой функции робот останавливается на 1 с.

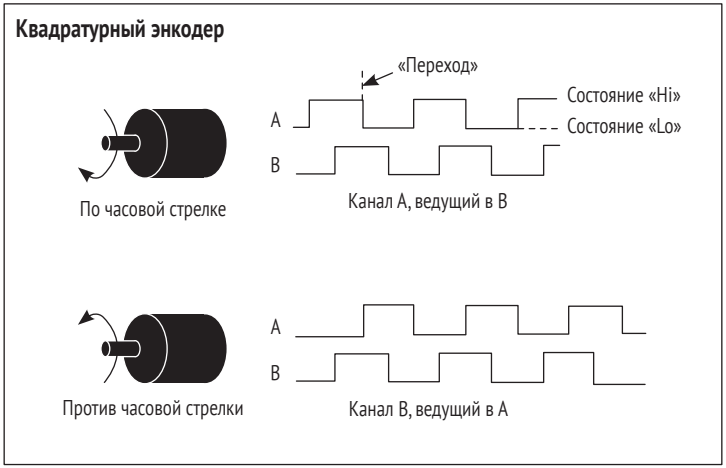
```
void loop()
{

  //Движение вперед 5 с
move_forward();
delay(5000);
  //Остановка 1 с
stop();
delay(1000);

  //Движение назад  5 с
move_backward();
delay(5000);
  //Остановка 1 с
stop();
delay(1000);

  //Поворт налево 5 с
move_left();
delay(5000);
  //Остановка 1 с
stop();
delay(1000);

  //Поворот направо 5 с
move_right();
delay(5000);
  //Остановка 1 с
stop();
delay(1000);
}
```

Интерфейс квадратурного энкодера с Tiva C Launchpad

Колесный энкодер – это датчик, прикрепленный к двигателю и определяющий количество оборотов колеса. Если мы знаем количество оборотов, то можем вычислить смещение, скорость, ускорение и угол поворота колеса.

Как уже упоминалось ранее, для данного робота мы выбрали мотор со встроенным энкодером. Этот энкодер квадратурного типа определяет направление и скорость вращения вала двигателя. В энкодерах могут быть использованы различные типы датчиков, такие как оптические датчики или датчики Холла. Данный энкодер для определения характеристик вращения использует эффект Холла. Квадратурный энкодер имеет два канала, а именно **канал A** и **канал B**.

Каждый канал генерирует цифровые сигналы со сдвигом на 90°. На следующем рисунке показана осциллограмма типичного квадратурного энкодера.



Осциллограммы канала A и канала B квадратурного энкодера

Если вал двигателя вращается по часовой стрелке, то фронт импульса канала A будет опережать фронт импульса канала B. Если вал двигателя вращается против часовой стрелки, то фронт импульса канала B будет опережать фронт импульса канала A. Эти значения используются для определения направления вращения вала двигателя. Далее мы обсудим, как перевести показания энкодера в значения скорости и смещения.

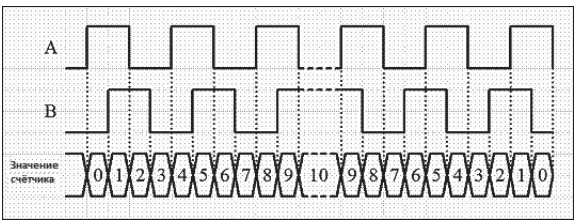
Обработка данных энкодера

Данные датчика представляют собой двухканальный импульсный сигнал со сдвигом фронтов импульсов на 90° по фазе. Используя эти данные, мы можем определить направление вращения и количество оборотов вала двигателя. С помощью этих данных мы определим скорость вращения вала двигателя и перемещение робота.

Работу энкодера характеризуют следующие значения: **количество импульсов на оборот** (**PPR**), или **количество линий на оборот** (**LPR**), и **количество счета на оборот** (**CPR**). PPR – это количество электрических импульсов на один оборот вала двигателя. Некоторые производители вместо PPR используют название CPR, так как каждый импульс состоит из двух вертикальных линий: переднего фронта (нарастание амплитуды импульса), когда сигнал скачкообразно растет от 0 до 1, и спада, когда амплитуда сигнала уменьшается от 1 до 0. Энкодер имеет два канала, каждый из которых генерирует импульсный сигнал. Фронты импульсов канала A и канала B сдвинуты по фазе по отношению друг к другу на 90°. Общее число линий (фронтов и спадов) будет в четыре

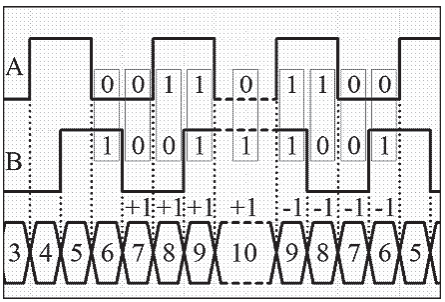
раза больше количества импульсов. Большинство квадратурных приемников использует так называемое 4X-декодирование для подсчета всех пиков от A- и B-каналов энкодера, уступая 4X-разрешению по сравнению с грубыми значениями PPR.

В нашем двигателе pololu указывается значение CPR, равное 64 на валу двигателя. Это соответствует 8400 CPR на выходном валу редуктора. По сути дела, мы получаем 8400 CPR на выходном валу редуктора, на один оборот вала двигателя. На следующем рисунке показано, как вычисляется количество импульсов энкодера.



Импульсы энкодера со значением счетчика

В этой спецификации энкодера задается количество импульсов на оборот; данное значение рассчитывается по количеству пиков, проходящих через канал энкодера. Один импульс канала энкодера соответствует четырем пикам. Таким образом, чтобы получить значение 8400 в нашем двигателе, необходимо, чтобы PPR был 8400/4 = 2100. Из предыдущего рисунка мы сможем рассчитать количество в одном обороте, но мы также должны определить и направление вращения. Количество импульсов не зависит от направления вращения вала двигателя и одинаково при вращении как по часовой, так и против часовой стрелки. Но для декодирования сигнала важно знать направление движения. На следующем рисунке показано, как декодировать импульсы энкодера.



Определение направления вращения вала двигателя
по импульсам энкодера

Если мы понаблюдаем за структурой кодированного сигнала, то можем понять, что он соответствует 2-битному Gray-кодированию. Gray-код кодирует числа так, что два соседних значения различаются только в одном разряде. Gray-коды широко используются в роторных энкодерах для эффективного кодирования.

Подробнее о Gray-коде читайте здесь: http://en.wikipedia.org/wiki/Gray_code.

Можно предсказать направление вращения двигателя по переходным состояниям. Таблица перехода приведена ниже.

Состояние	Переход по часовой стрелке	Переход против часовой стрелки
0,0	0,1 до 0,0	1,0 до 0,0
1,0	0,0 до 1,0	1,1 до 1,0
1,1	1,0 до 1,1	0,1 до 1,1
0,1	1,1 до 0,1	0,0 до 0,1

Будет удобнее представить переходы на схеме переходного состояния:

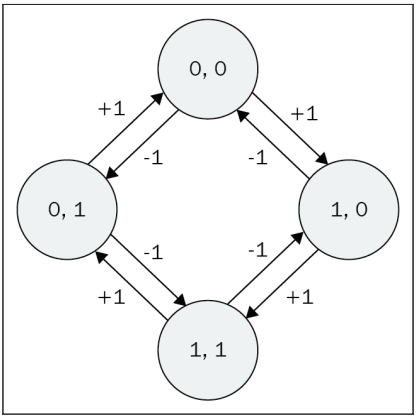


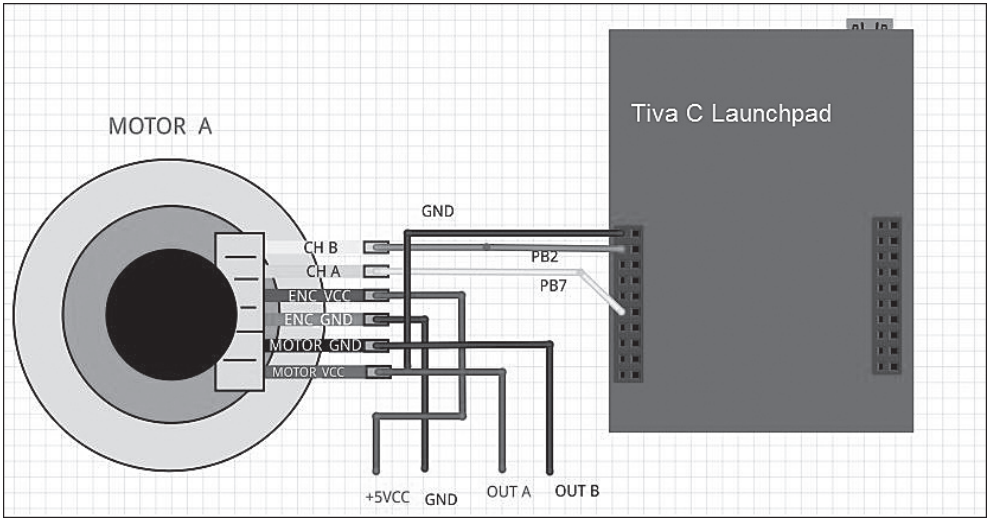
Диаграмма перехода состояний энкодеров

Получив Gray-код, мы можем обработать поступающие импульсы с помощью микроконтроллера. Контакты каналов двигателя необходимо подключить к контактам прерывания микроконтроллера. Таким образом, когда в канале появятся реберные переходы, будет генерироваться прерывание. При появлении на контакте сигнала прерывания микроконтроллер выполнит функцию прерываний. В соответствии с текущим состоянием контактов и их предыдущих значений мы определяем направление вращения вала двигателя и решаем, нужно ли увеличивать или уменьшать количество прерываний (импульсов). Это и есть основная логика обработки энкодера.

Получив значение счетчика, можно вычислить угол поворота (в градусах) с помощью выражения: Угол = (Значение счетчика / CPR) × 360. Если заменить CPR на 8400, уравнение принимает такой вид: Angle = 0,04285 × Значение счет-

чика, то есть для поворота на один градус должно быть получено 24 отсчета, или шесть закодированных импульсов канала.

На следующем рисунке показана схема взаимодействия энкодера одного из двигателей с Tiva C LaunchPad.



Взаимодействие энкодера двигателя с Tiva C LaunchPad

Максимальный уровень выходного импульса от энкодера – от 0 до 5 В. В этом случае мы можем напрямую воздействовать на энкодер с LaunchPad, так как микроконтроллер работает с сигналами до 5 В. Если энкодер не может развить сигнал до уровня 5 В, следует применить преобразователь уровня от 3,3 до 5 В, как это мы делали для драйвера двигателя раньше.

В следующем разделе мы напишем код для Energia, который позволит протестировать сигнал квадратурного энкодера. Нам необходимо убедиться, получим ли мы правильное значение счетчика от энкодера.

Код согласования квадратурного энкодера

Этот код выведет данные счетчика левого и правого энкодеров двигателя через последовательный порт. Два энкодера находятся в схеме декодирования 2X, поэтому мы получим 4200 CPR. В первом разделе кода мы определяем контакты для двухканальных выходов двух энкодеров и даем описание переменной `count` для двух энкодеров. Переменная энкодера использует ключевое слово `volatile` перед типом переменных данных.

Основное использование `volatile` состоит в том, что переменная с `volatile keyword` будет сохранена в оперативной памяти, а обычные переменные – в регистре процессора. Значения кодировщика меняются очень быстро, поэтому

использование обычной переменной будет неточным. Чтобы получить точность, будем использовать переменные volatile для энкодера, как показано ниже:

```
//Определение пинов энкодера

// Левый энкодер

#define Left_Encoder_PinA 31
#define Left_Encoder_PinB 32

volatile long Left_Encoder_Ticks = 0;

// Переменная для чтения текущего состояния левого датчика энкодера
volatile bool LeftEncoderBSet;

//Правый энкодер

#define Right_Encoder_PinA 33
#define Right_Encoder_PinB 34
volatile long Right_Encoder_Ticks = 0;
// Переменная для чтения текущего состояния правого датчика энкодера
volatile bool RightEncoderBSet;
```

Следующий фрагмент кода является определением функции setup(). В языке wiring setup() – это встроенная функция, используемая для инициализации и выполнения действий на один такт для переменных и функций. Внутри setup() мы инициализируем последовательную передачу данных со скоростью передачи данных 115 200 и вызов определяемой пользователем функции SetupEncoders() для инициализации контактов энкодера. В основном передача последовательных данных используется для проверки значения счетчика энкодера через serial terminal.

```
void setup()
{
    // Последовательный порт  со скоростью передачи 115 200 бод
  Serial.begin(115200);
  SetupEncoders();
}
```

Определение SetupEncoders() приводится в следующем коде. Для получения импульса энкодера нам понадобятся два контакта LaunchPad в качестве входных. Настроим входные контакты энкодера на LaunchPad с помощью нагрузочного резистора. Функция attachInterrupt() настроит один из выводов энкодера в качестве прерывания. Функция attachInterrupt() имеет три аргумента. Первый аргумент – это значение pin, второй – **процедура обслуживания прерывания** (ISR), и третьим аргументом является условие прерывания, то есть условие, при котором запустится прерывание ISR. В этом коде мы настроем контакт A левого и правого энкодеров как контакт прерывания; он вызывает ISR при появлении положительного импульса.

```
void SetupEncoders()
{
```

```
  // Квадратурные энкодеры
  // Левый энкодер
  pinMode(Left_Encoder_PinA, INPUT_PULLUP);
  // устанавливает pin A в качестве входа
  pinMode(Left_Encoder_PinB, INPUT_PULLUP);
  // устанавливает pin B в качестве входа
  attachInterrupt(Left_Encoder_PinA, do_Left_Encoder, RISING);

  // Правый энкодер
  pinMode(Right_Encoder_PinA, INPUT_PULLUP);
  // устанавливает pin A в качестве входа
  pinMode(Right_Encoder_PinB, INPUT_PULLUP);
  // устанавливает pin B в качестве входа
  attachInterrupt(Right_Encoder_PinA, do_Right_Encoder, RISING);
}
```

Следующая часть кода является встроенной функцией `loop()` на языке wiring. Функция `loop()` – это бесконечный цикл, в который мы помещаем основной код. В этом коде мы вызываем функцию `Update_Encoders()` для непрерывной публикации значения энкодера через последовательный терминал (serial terminal).

```
void loop()
{
  Update_Encoders();
}
```

Следующий код является определением функции `Update_Encoders()`. Он публикует два значения энкодера в строке со стартовым символом `e`, и значения разделяются пробелами табуляции. Функция `Serial.print()` – это встроенная функция, которая выведет заданную в качестве аргумента символ/строку.

```
void Update_Encoders()
{
  Serial.print("e");
  Serial.print("\t");
  Serial.print(Left_Encoder_Ticks);
  Serial.print("\t");
  Serial.print(Right_Encoder_Ticks);
  Serial.print("\n");
 }
```

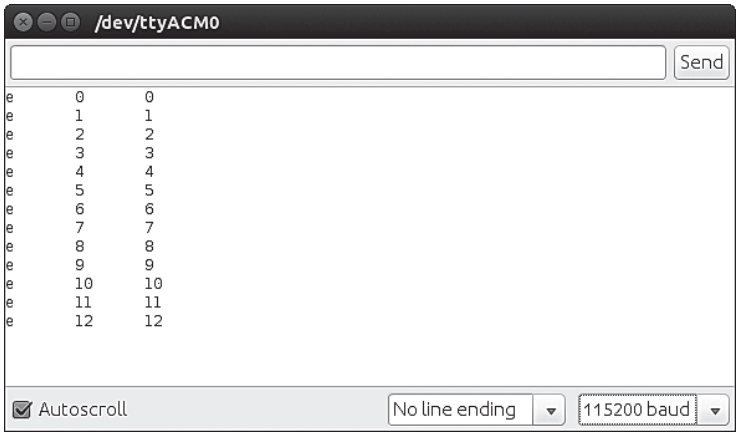
Следующий код является определением ISR левого и правого кодировщиков. Когда на каждом из выводов будет обнаружен положительный фронт импульса, будет вызвана одна из ISR. Текущие контакты прерывания – Pin A каждого из энкодеров. После получения прерывания мы можем предположить, что возрастающее значение на контакте A имеет более высокое значение, поэтому считывать его нет необходимости. Читаем контакт B двух энкодеров и сохраняем состояние контактов как `LeftEncoderBSet` или `RightEncoderBSet`. Далее, после сравнения текущего состояния с предыдущим состоянием контакта B, мы можем

определить направление вращения и решить, следует ли в соответствии с таблицей состояния перехода увеличить или уменьшить количество импульсов.

```
void do_Left_Encoder()
{
  LeftEncoderBSet = digitalRead(Left_Encoder_PinB);
  // Читаем данные с вводного пина
  Left_Encoder_Ticks -= LeftEncoderBSet ? -1 : +1;
}

void do_Right_Encoder()
{
  RightEncoderBSet = digitalRead(Right_Encoder_PinB);
  // Читаем данные с вводного пина
  Right_Encoder_Ticks += RightEncoderBSet ? -1 : +1;
}
```

Загрузите черновик кода и просмотрите выходные данные с помощью последовательного монитора в Energia. Откройте меню **Tools → Serial monitor**. Проверните валы обоих двигателей вручную, чтобы отобразить данные, генерируемые энкодерами. Установите скорость передачи данных в последовательном мониторе, совпадающую с инициализированной в коде, – 115 200.

Выходные данные будут выглядеть так:



Подключение энкодера к LaunchPad

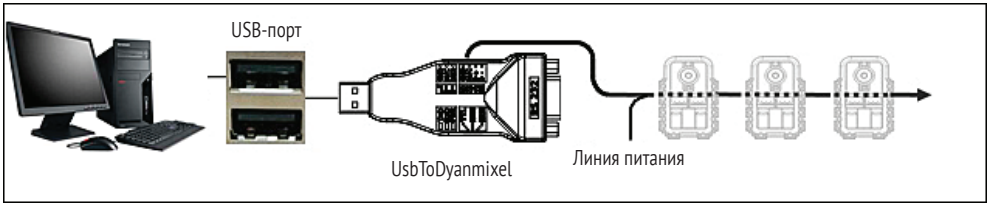
Чтобы модернизировать робот для работы с высокой точностью и повышенной полезной нагрузкой, следует подумать о приводах высокого качества, таких как Dynamixel. Сервоприводы Dynamixel – это интеллектуальные приводы, которые имеют встроенное ПИД-регулирование и мониторинг параметров сервопривода и энкодера, таких как крутящий момент, положение и т. д.

Работа с приводом Dynamixel

Dynamixel – это исполнительные механизмы с возможностью подключения в общую шину управления, разработанные корейской фирмой ROBOTIS специально для нужд робототехники. Эти приводы многофункциональны, имеют возможность передавать управляющему компьютеру для контроля такие параметры, как положение, скорость, внутренняя температура, контроль входного напряжения и прочее (функция обратной связи).

Сервоприводы Dynamixel соединяются последовательно, друг за другом. Данный способ последовательного подключения позволяет контролировать все сервоприводы одним контроллером. Сервоприводы Dynamixel подключаются через RS485 или TTL. Список доступных сервоприводов Dynamixel находится по адресу: http://www.robotis.com/shop/.

Интерфейс Dynamixel очень простой. Dynamixel поставляется с контроллером под названием USB2Dyanmixel. Данный контроллер преобразует USB в уровни Dynamixel, совместимые с TTL/RS485. На следующем рисунке показано подключение схемы Dynamixel.



Сопряжение приводов Dynamixel с ПК

ROBOTIS предоставляет Dynamixel SDK доступ к регистрам двигателей; мы можем считывать и записывать значения в регистры Dynamixel и получать для контроля такие данные, как положение, температура, напряжение и прочее.

Инструкции по установке USB2Dynamixel и пакета Dynamixel SDK: support.robotis.com/en/.

Dynamixel программируется с помощью библиотек Python. Одна из библиотек Python для обработки данных сервоприводов Dynamixel – это **pydynamixel**. Этот пакет доступен для таких операционных систем, как Windows и Linux. Данная библиотека работает с сервоприводами серий RX, MX и EX.

Пакет Python pydynamixel можно скачать по ссылке https://pypi.python.org/pypi/dynamixel/.

Скачайте пакет и распакуйте его в домашнюю папку. Затем откройте **Terminal** и выполните следующую команду:

```
sudo python setup.py install
```

После установки пакета можно попробовать обнаружить сервопривод, USB-2Dynamixel и описать случайную позицию серводвигателя. Для этого воспользуйтесь следующим примером, который написан для сервоприводов серии RX и MX:

```
#!/usr/bin/env python
```

Следующий код импортирует необходимые для этого примера модули Python и включает в себя модули Dynamixel Python:

```
import os
import dynamixel
import time
import random
```

Код, приведенный ниже, определяет основные параметры для связи с Dynamixel. Переменная `nServos` обозначает количество сервоприводов Dynamixel, подключенных к шине. Переменная `portName` указывает на последовательный порт USB2Dynamixel, к которому подключен сервопривод Dynamixel. Переменная `baudRate` – скорость передачи USB2Dynamixel и Dynamixel.

```
# Количество Dynamixels на нашей шине
nServos = 1

# Установите последовательный порт соответствующим образом
if os.name == "posix":
    portName = "/dev/ttyUSB0"
else:
    portName = "COM6"
# Скорость передачи данных по умолчанию устройства USB2Dynamixel
baudRate = 1000000
```

Следующий код – это функция Dynamixel Python для подключения к сервоприводу Dynamixel. Когда сервопривод будет подключен, программа для обнаружения подключенных устройств будет сканировать сообщения шины и определит номер сервопривода, начиная с ID от `1` до `255`. Идентификатор каждого сервопривода является его же идентификацией. Когда сервопровод будет идентифицирован `nServos` как `1`, сканирование будет остановлено, и на шине данных будет зафиксировано одно устройство:

```
# Подключение к последовательному порту
print "Connecting to serial port", portName, '...',
serial = dynamixel.serial_stream.SerialStream( port=portName,
baudrate=baudRate, timeout=1)
print "Connected!"
net = dynamixel.dynamixel_network.DynamixelNetwork( serial )
net.scan( 1, nServos )
```

Следующий код добавит идентификатор Dynamixel и сервопривод в список `myActutors`. Мы можем написать значения сервопривода, используя его ИД и сервопривод как объект. Мы можем использовать список `myActuators` для дальнейшей обработки:

```
# удерживать список dynamixels
myActuators = list()
print myActuators

Это действие создаст список для хранения данных, получаемых с приводов dynamixel.

print "Scanning for Dynamixels...",

for dyn in net.get_dynamixels():
    print dyn.id,
    myActuators.append(net[dyn.id])
print "...Done"
```

Следующий код напишет случайные положения каждому доступному на шине данных сервоприводу Dynamixel в диапазоне от 450 до 600. Диапазон позиций в Dynamixel – от 0 до 1023, который позволит установить такие параметры сервопривода, как скорость (`speed`), крутящий момент (`torque`), диапазон крутящего момента (`torque_limt`), максимальное значение крутящего момента (`max_torque`) и т. д.:

```
# Установка скорости и крутящего момента по умолчанию
for actuator in myActuators:
    actuator.moving_speed = 50
    actuator.synchronized = True
    actuator.torque_enable = True
    actuator.torque_limit = 800
    actuator.max_torque = 800
```

Следующий код будет публиковать текущую позицию в текущем приводе:

```
# Перемещайте сервоприводы случайным образом и публикуйте их текущие позиции
while True:
    for actuator in myActuators:
        actuator.goal_position = random.randrange(450, 600)
    net.synchronize()
```

Следующий код будет считывать все данные из приводов:

```
    for actuator in myActuators:
        actuator.read_all()
        time.sleep(0.01)

    for actuator in myActuators:
        print actuator.cache[dynamixel.defs.REGISTER['Id']], actuator.
cache[dynamixel.defs.REGISTER['CurrentPosition']]

    time.sleep(2)
```

Работа с ультразвуковыми датчиками расстояния

Как уже упоминалось ранее, основная характеристика разрабатываемого нами мобильного робота – это автономная навигация. Идеальная навигация означает, что робот может планировать свой путь от текущей позиции до места назначения и двигаться без каких-либо столкновений. Мы в этом роботе для обнаружения объектов в непосредственной близости используем ультразвуковые датчики расстояния. Эти объекты не могут быть обнаружены с помощью сенсора Kinect. Сочетание Kinect и ультразвуковых датчиков с высокой степенью вероятности позволяет избежать столкновения робота с препятствием.

Ультразвуковые датчики расстояния работают следующим образом. Передатчик излучает не слышный для человеческого уха ультразвуковой сигнал. После отправки ультразвуковой волны приемная часть ультразвукового датчика ждет отраженный от обнаруживаемого препятствия сигнал (эхо). Если отраженного сигнала нет, значит, перед роботом нет препятствий. Если ультразвуковой датчик принимает отраженный сигнал, значит, по курсу у робота появилось препятствие. Чем короче временной промежуток между излученным ультразвуковым сигналом и принятым от него эхом, тем меньше расстояние между роботом и препятствием. Зная этот временной промежуток, мы, используя следующую формулу, можем вычислить расстояние до препятствия:

Скорость звука × (Пройденное время / 2) = Расстояние до объекта.

Значение скорости звука мы принимаем равным 340 м/с.

Большинство ультразвуковых датчиков способно обнаружить препятствие на расстоянии от 2 до 400 см. В этом роботе мы используем датчик HC-SR04. Рассмотрим, как согласовать датчик HC-SR04 с микроконтроллером Tiva C Launchpad для измерения расстояния от робота до препятствий.

Согласование HC-SR04 с Tiva C LaunchPad

На следующем рисунке показана схема подключения ультразвукового датчика HC-SR04 к микроконтроллеру Tiva C LaunchPad.

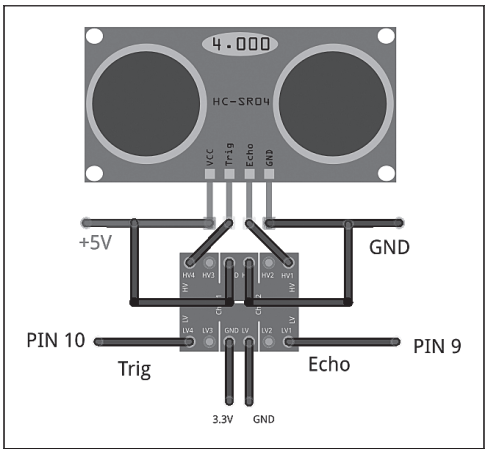


Схема подключения ультразвукового датчика HC-SR04
к микроконтроллеру LaunchPad

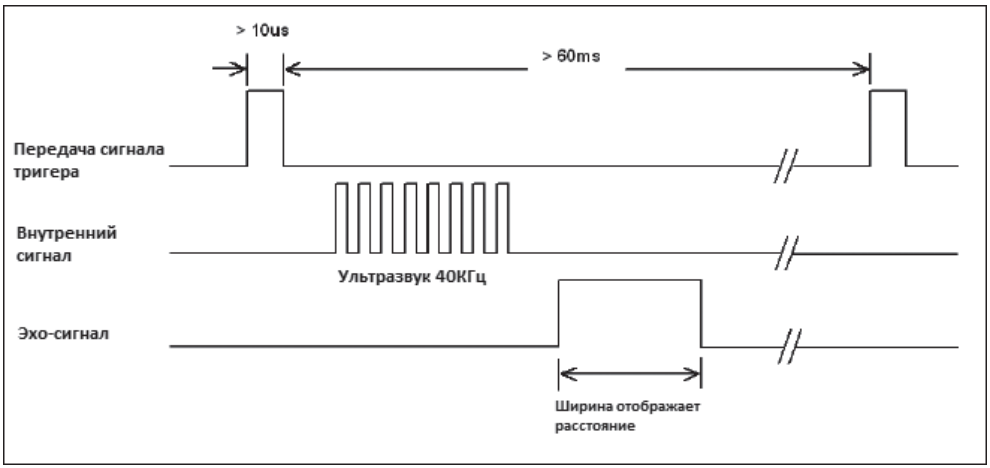
Питание ультразвукового датчика расстояния осуществляется напряжением +5 В. Так как амплитуда входного и выходного сигналов датчика равна 5 В, нам для взаимодействия с LaunchPad потребуется преобразователь с 5 на 3,3 В. Контакты преобразователя 5 В подключаются к контактам датчика Trig и Echo. Контакты преобразователя 3,3 В подключаются к 9-му и 10-му контактам микроконтроллера (как подключить преобразователь к датчику, показано на предыдущем рисунке). Теперь, когда датчик подключен, рассмотрим программу, управляющую контактами ввода/вывода (I/O).

Работа HC-SR04

Временные диаграммы сигнала (осциллограммы) для каждого контакта показаны на следующем рисунке. Для запуска триггера нам необходимо подать на его вход (Trig input) короткий импульс в 10 мкс. Триггер будет запущен, и на его выходе (Integral signal) пакет из 8 импульсов, модулированных частотой 40 кГц. Этот пакет импульсов будет воспроизведен динамиком датчика и, превратившись в звуковую волну, начнет свой путь к препятствию. Отразившись от препятствия, эхо-сигнал вернется в приемную часть датчика (микрофон). Эхо-сигнал (отраженный сигнал) – это расстояние до объекта, пропорциональное ширине импульса и дистанции. Дистанция рассчитывается через временной интервал между отправкой триггерных сигналов и приемом эхо-сигналов по следующей формуле:

Дистанция = Время получения отраженного сигнала × Скорость звука (340 м/с) / 2.

Чтобы избежать перекрытия между триггером и эхом, желательно перед новым запуском триггера использовать задержку по времени в 60 мс:



Входной и выходной сигналы ультразвукового датчика

Получение данных через Tiva C LaunchPad

Приведенный ниже код Energia для микропроцессора LaunchPad считывает с последовательного порта значения с ультразвукового датчика. Далее назначает контакты в LaunchPad для обработки ультразвукового эхо-сигнала и запуск триггеров. Одновременно устанавливаются переменные длительности импульса и расстояния в сантиметрах:

```
const int echo = 9, Trig = 10;
long duration, cm;
```

Следующий фрагмент кода является функцией `setup()`. Функция `setup()` вызывается при запуске программы. Используйте эту функцию для инициализации переменных, режимов контактов, запуска библиотек и т. д. Функция настройки запускается только один раз после каждого включения или перезагрузки LaunchPad.

Функция `setup()` инициализирует связь через последовательный порт со скоростью передачи данных 115 200 бод и устанавливает ультразвуковой режим, настраивая контакты с помощью функции `SetupUltrasonic()`:

```
void setup()
{
  // Последовательный порт со скоростью передачи 115200 бод
  Serial.begin(115200);
  SetupUltrasonic();
}
```

Следующая функция назначает контакт `Trigger` как `OUTPUT`, а контакт `Echo` – в качестве `INPUT`. Для определения контактов в качестве `INPUT` или `OUTPUT` используется функция `pinMode()`.

```
void SetupUltrasonic()
{
  pinMode(Trig, OUTPUT);
  pinMode(echo, INPUT);
}
```

После создания функции `setup()`, которая инициализирует и задает начальные значения, функция `loop()` оправдывая свое имя и создает последовательные циклы, позволяющие программе менять значения и передавать их. Используйте ее для полного контроля микропроцессора Launchpad.

Основной цикл находится в следующем коде. Эта функция представляет собой бесконечный цикл и вызывает функцию `Update_Ultra_Sonic()` для обновления и публикации значений ультразвука и считывания через последовательный порт:

```
void loop()
{
  Update_Ultra_Sonic();
  delay(200);
}
```

Следующий код является определением функции `Update_Ultra_Sonic()`. Эта функция выполняет следующие операции. Во-первых, она принимает значение сигнала пуска – `LOW` длительностью 2 мс и `HIGH` длительностью 10 мс. После 10 мс функция снова возвратит значение контакта к значению `LOW`. Все эти преобразования выполняются согласно временной диаграмме. Мы уже видели, что ширина импульса запуска составляет 10 мкс.

После импульса запуска длительностью 10 мкс нам следует отследить время получения сигнала от контакта `Echo`, т. е. сколько было затрачено времени на прохождение звука от излучателя датчика до объекта и от объекта к приемнику датчика. Мы можем считать длительность импульса с помощью функции `pulseIn()`. После того как будет определено это значение (время, затраченное на прохождение ультразвуковой волны к препятствию и обратно), мы, как показано в следующем коде, можем преобразовать время в сантиметры с помощью функции `microsecondsToCentimeters()`:

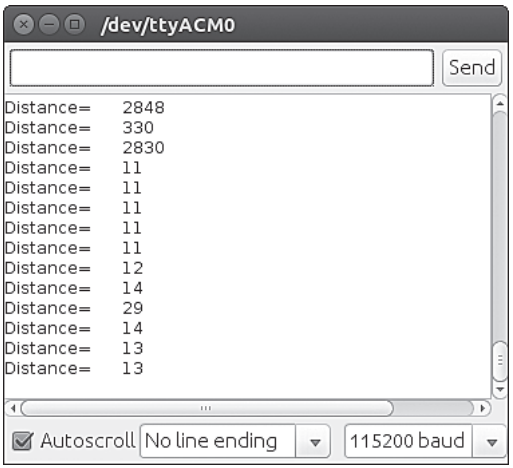
```
void Update_Ultra_Sonic()
{
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(Trig, LOW);

  duration = pulseIn(echo, HIGH);
  // перевести время в расстояние
  cm = microsecondsToCentimeters(duration);
  // Отправка через последовательный порт
  Serial.print("distance=");
  Serial.print("\t");
```

```
  Serial.print(cm);
  Serial.print("\n");
}
```

Следующий код является функцией преобразования из микросекунд в расстояние в сантиметрах. Скорость звука составляет 340 м/с, то есть 29 мс на сантиметр. Таким образом, мы получаем общее расстояние, разделив общее количество микросекунд на 29/2:

```
long microsecondsToCentimeters(long microseconds)
{
  return microseconds / 29 / 2;
}
```

После загрузки кода с помощью команды меню **Tools → Serial Monitor** откройте окно последовательного монитора Energia и измените в правом нижнем открывающемся списке значение скорости на 115 200. После этого вы можете увидеть значения ультразвукового датчика, как показано на рисунке.



Выход ультразвукового датчика расстояния в последовательном мониторе Energia

Согласование Tiva C LaunchPad с Python

В этом разделе мы рассмотрим, как подключить Tiva C LaunchPad с Python для передачи данных из Launchpad на бортовой ПК. Модуль PySerial используется для взаимодействия Launchpad с Python. Документацию PySerial и порядок установки на Windows, Linux и OS X вы найдете по следующей ссылке: http://pyserial.sourceforge.net/pyserial.html.

PySerial можно установить с помощью менеджера пакетов Ubuntu без затруднений. Для установки используется следующая команда, введенная в терминале:

```
$ sudo apt-get install python-serial
```

После установки пакета `python-serial` мы можем написать код для интерфейса LaunchPad. Код взаимодействия приведен в следующем разделе. Этот код импортирует модули `serial` и `sys`. Модуль `serial` обрабатывает последовательные порты LaunchPad и выполняет операции чтения и передачи данных и т. д. Модуль `sys` предоставляет доступ к переменным, которые используются или поддерживаются интерпретатором и взаимодействующими с ними функциями. Они всегда доступны для всей программы.

```
#!/usr/bin/env python
import serial
import sys
```

Когда мы подключаем LaunchPad к компьютеру, устройство регистрируется в ОС как виртуальный последовательный порт. В Ubuntu название устройства будет `/dev/ttyACMx`, где `x` – это количество подключенных устройств. Если подключено только одно устройство, значение `x` будет равно 0. Для взаимодействия с LaunchPad мы должны обрабатывать информацию с этого устройства.

Следующий код попытается открыть последовательный порт `/dev/ttyACM0` LaunchPad со скоростью передачи данных 115 200. Если это не удастся, на экране появится сообщение **Unable to open serial port** («Не удалось открыть последовательный порт»).

```
try:
    ser = serial.Serial('/dev/ttyACM0',115200)
except:
    print "Unable to open serial port"
```

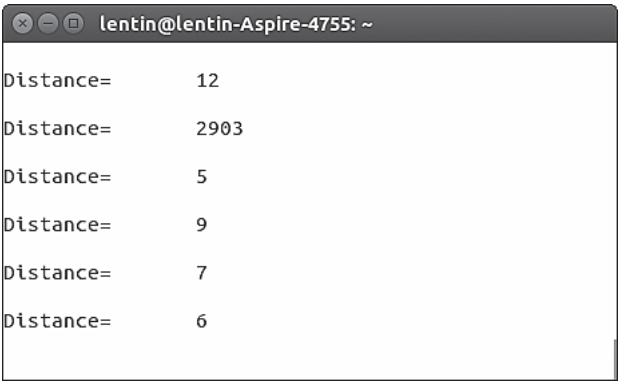
Следующий код будет считывать серийные данные до тех пор, пока серийный символ не перейдет на новую строку (`'\n'`) и не опубликует ее в терминале. Если для выхода из программы нажать на клавиатуре сочетание клавиш **Ctrl**+**C**, будет вызван `sys.exit(0)`.

```
while True:
    try:
        line = ser.readline()
        print line
    except:
        print "Unable to read from device"
        sys.exit(0)
```

После сохранения файла измените его разрешение на выполнение с помощью следующей команды:

```
$ sudo chmod +X script_name
$ ./script_name
```

Вывод скрипта будет выглядеть так:



Выход ультразвукового датчика расстояния в серийном мониторе Energia (изображение текстового поля терминала инвертировано)

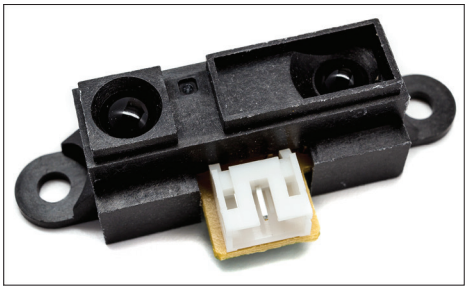
Работа с ИК-датчиком расстояния

Принцип определения расстояния от робота до препятствия с помощью инфракрасного датчика отличается от принципа определения расстояния с помощью ультразвукового датчика. Здесь от препятствия отражается не ультразвук, а инфракрасный свет, излучаемый прожектором ИК-датчика. Приемник ИК-излучения захватывает отраженный инфракрасный свет. Напряжение на выходе приемника ИК-излучения напрямую зависит от уровня его освещенности.

Один из популярных датчиков ИК-диапазона – это Sharp GP2D12. Ссылка для приобретения этого датчика приведена ниже: http://www.robotshop.com/en/sharp-gp2y0a21yk0f-ir-range-sensor.html.

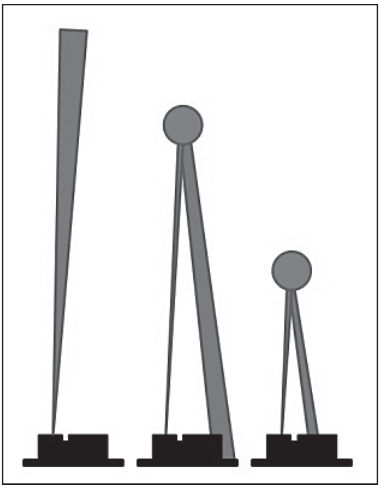
Альтернатива на AliExpress: https://ru.aliexpress.com/wholesale?catId=0&initiative_id=SB_20171211211645&SearchText=Sharp+GP2Y0A21YK0F+IR.

На следующем рисунке показан датчик Sharp GP2D12.



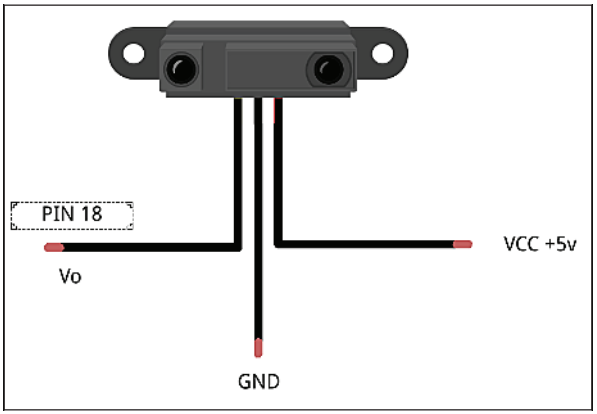
Датчик Sharp GP2D12

Датчик посылает луч инфракрасного света и использует триангулирование (метод разбиения поверхности на треугольники) для измерения расстояния. Диапазон обнаружения GP2D12 составляет от 10 до 80 см. Ширина светового пятна на расстоянии 80 см составляет 6 см. Передача и отражение светового луча ИК-диапазона:



Зондирование препятствия с помощью ИК-датчика

С левой стороны датчика находится прожектор инфракрасного излучения, который непрерывно посылает свет в ИК-диапазоне. После попадания на поверхность препятствия свет отражается и фиксируется объективом приемника ИК-излучения. Схема подключения инфракрасного датчика показана ниже.



Контакты для подключения датчика Sharp GP2D12

Аналоговый контакт Vo подключается к контакту АЦП (ADC) LaunchPad. Код взаимодействия датчика расстояния Sharp с микроконтроллером Tiva C LauncPhad будет рассмотрен в этом разделе. В этом коде мы назначим контакту 18 LaunchPad режим АЦП и будем считывать уровни напряжения от датчика расстояния Sharp. Уравнение диапазона ИК-датчика GP2D12 определяется следующим образом:

Range = (6787 / (*Volt* – 3)) – 4.

Здесь *Volt* – значение аналогового напряжения от контакта АЦП Vout. В первом разделе кода в качестве входных данных задается 18-й контакт Tiva C LaunchPad и запускается последовательная связь со скоростью передачи данных 115 200:

```
int IR_SENSOR = 18; // Датчик подключён к аналогову входу
int intSensorResult = 0; // Результат от датчика
float fltSensorCalc = 0; // Расчётное значение

void setup()
{
Serial.begin(115200); // Установки связи с компьютером для представления результатов
                      // через последовательный порт
}
```

В следующем разделе кода контроллер постоянно считывает данные с аналогового вывода и преобразует его в значение расстояния в сантиметрах:

```
void loop()
{

// Прочитать значения от датчика ИК
intSensorResult = analogRead(IR_SENSOR); //Получить значение датчика

// Рассчитать расстояние в см  по уравнению диапазона
fltSensorCalc = (6787.0 / (intSensorResult - 3.0)) - 4.0;

Serial.print(fltSensorCalc); //Отправить расстояние на компьютер
Serial.println(" cm"); //Добавить см к результату
delay(200); //Ожидание
}
```

Это основной код интерфейса датчика расстояния Sharp. Но инфракрасным датчикам расстояния присущи некоторые недостатки:

- мы не можем использовать эти датчики при солнечном свете. Поэтому работа такого робота вне помещения затруднена;
- если препятствие не отражает инфракрасный свет, датчик не работает;
- датчик работает только в пределах ИК-диапазона.

В следующем разделе мы обсудим работу IMU (инерционного измерительного блока) и его взаимодействие с микропроцессором Tiva C Launchpad. IMU передает данные одометрии, и его можно использовать как передатчик входных сигналов в алгоритмах навигации.

Работа с инерционным измерительным модулем

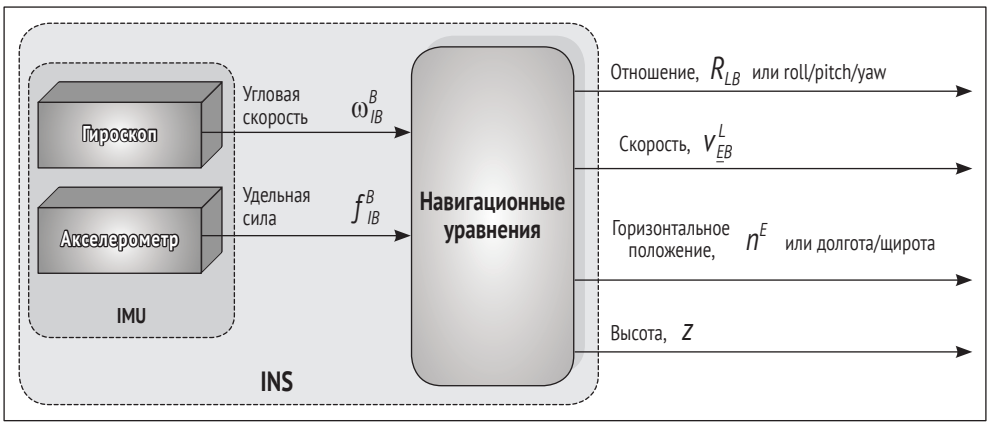
Инерциальный измерительный модуль (IMU) – это электронное устройство, которое с помощью комбинации акселерометра, гироскопа и магнитометра измеряет скорость, ориентацию и гравитационные силы. IMU в робототехнике можно применять для решения многих задач. Например, это балансировка беспилотных летательных аппаратов (БПЛА) и навигация роботов.

В этом разделе мы обсудим применение IMU в мобильной робототехнической навигации и познакомимся с несколькими моделями инерциальных блоков, предлагаемыми на рынке, рассмотрим их взаимодействие с LaunchPad.

Инерциальная навигация

IMU определяет ускорение и ориентацию робота по отношению к инерциальному пространству. Если мы знаем начальное положение, скорость и направление, можно вычислить скорость. По начальному положению, ускорению и скорости можно получить значение текущего положения. Для определения требуемого направления робота нам понадобится информация по ориентации робота. Эту информацию мы можем получить путем интеграции значений угловой скорости от гироскопа.

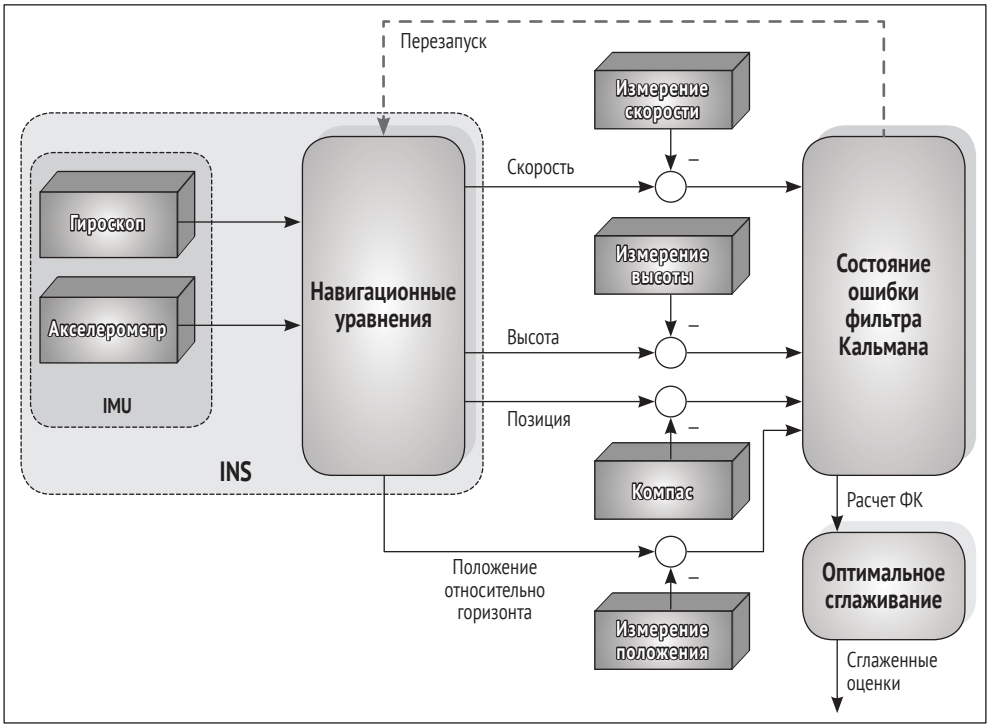
На следующем рисунке показана инерциальная навигационная система, которая преобразует значения IMU в данные одометрии.



Блок-схема IMU

Значения, полученные из IMU, преобразуются в навигационную информацию с помощью навигационных уравнений с применением оценочных фильтров, таких как фильтр Кальмана. **Фильтр Кальмана** – это алгоритм, который оценивает состояние системы от полученных данных измерений (http://en.wikipedia.org/wiki/Kalman_filter). Данные **инерциальной системы навигации** (ins) имеют погрешность, источник которой – ошибки измерений акселе-

рометра и гироскопа. Для уменьшения значения погрешности ins используют данные, получаемые от других датчиков, обеспечивающие прямые измерения измеряемых величин. Основываясь на данных измерения датчика, модели ошибки датчика и фильтра Калмана, оцениваются ошибки в навигационных уравнениях и все ошибки остальных датчиков. На следующей схеме показана инерциальная навигационная система с использованием фильтра Калмана.



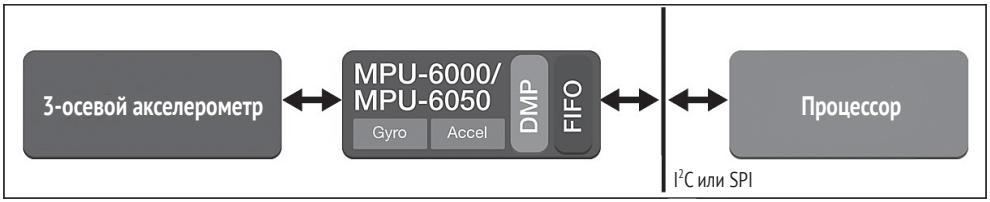
Инерциальная навигационная система с фильтром Кальмана

Вместе с данными от энкодера значение от IMU принимается как значение одометра, и его можно использовать для вычисления текущей позиции движущегося объекта с использованием ранее определенного положения. В следующем разделе познакомимся с наиболее популярным инерциальным измерителем от InvenSense, который называется **MPU 6050**.

Взаимодействие MPU 6050 с Tiva C LaunchPad

Семейство MPU-6000/MPU-6050 – это первые и единственные в мире 6-осевые устройства с низкой мощностью потребления, отслеживающие движение. Устройства имеют низкую цену, высокую эффективность и соответствуют требованиям IMU смартфонов, планшетов, носимых датчиков и роботов.

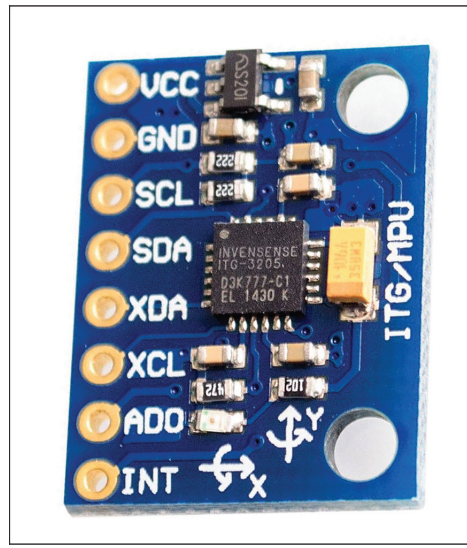
Устройства МПУ-6000/6050 сочетают в себе 3-осевой гироскоп и 3-осевой акселерометр на кремниевом кристалле, соединенный с бортовым цифровым процессором движения, который может вести комплексную обработку сложных 9-осевых алгоритмов объединения движения. На следующей блок-схеме показаны схема MPU 6050 и передача данных MPU 6050.



Блок-схема MPU 6050

Устройство MPU 6050 показано на следующем рисунке. Интегральный измерительный блок можно приобрести по адресу: https://www.sparkfun.com/products/11028.

Альтернатива на AliExpress: https://ru.aliexpress.com/wholesale?catId=202000062&initiative_id=AS_20171212051607&SearchText=mpu6050.

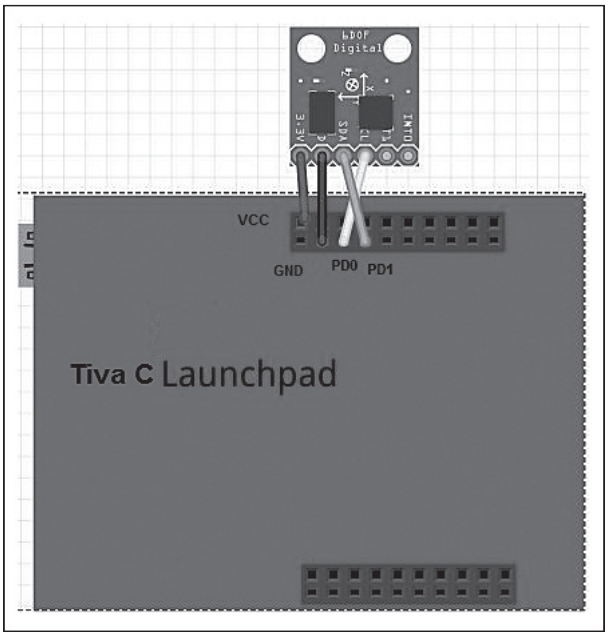


Инерциальный измерительный блок MPU 6050

Подключение MPU 6050 к LaunchPad приведено в следующей таблице. Остальные контакты не подключаются.

Контакты LaunchPad	Контакты MPU 6050
+3,3 В	VCC/VDD
GND	GND
PD0	SCL
PD1	SDA

Далее показана схема подключения MPU 6050 к Tiva C LaunchPad:



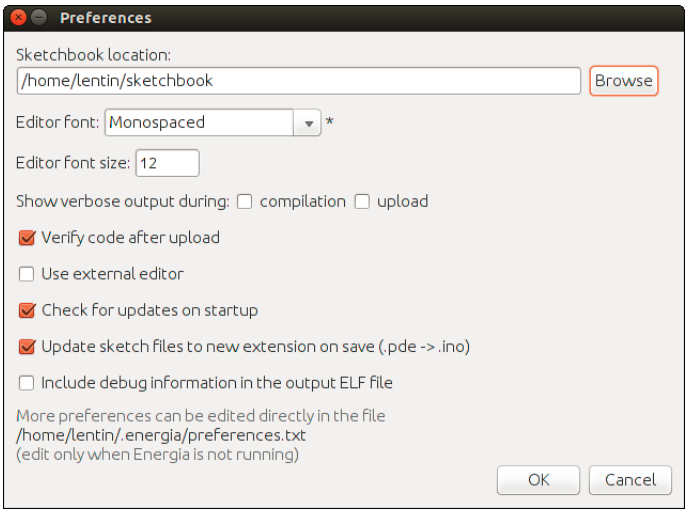
Подключение MPU 6050 к Tiva C LaunchPad

MPU 6050 и Launchpad взаимодействуют с помощью протокола I2C, питание – 3,3 В, получаемое от LaunchPad.

Установка библиотеки MPU 6050 в Energia

В этом разделе рассматривается код, позволяющий взаимодействовать с Energia. Код взаимодействия использует библиотеку для взаимодействия с MPU 6050 https://github.com/jrowberg/i2cdevlib/zipball/master.

Скачайте zip-файл по указанной ссылке. Откройте в Energia меню **File** → **Preference**, как показано на следующем рисунке.



Интерфейс подключения устройства MPU 6050 к LaunchPad

В поле ввода **Sketchbook location** введите путь `/home/имя_пользователя/Sketchbook`, как показано на предыдущем рисунке, и создайте папку с именем библиотеки. Извлеките файлы из папки `Arduino` (zip-файл) в местоположение `sketchbook/libraries`. Пакеты Arduino этого репозитория совместимы с LaunchPad. Извлеченные файлы содержат следующие пакеты: I2Cdev, Wire и MPU6050. Эти пакеты необходимы для взаимодействия с датчиками MPU 6050. В папке библиотеки присутствуют и другие пакеты датчиков, но сейчас мы их использовать не будем.

Предыдущая инструкция предназначена для Ubuntu, но может быть выполнена в Windows и Mac OS X.

Код согласования в Energia

Этот код используется для чтения и передачи из MPU 6050 в LaunchPad необработанных значений. Код использует стороннюю библиотеку МПУ 6050, совместимую с Energia IDE. Ниже приведены объяснения каждого блока кода.

В этом первом разделе кода мы включаем заголовки, необходимые для взаимодействия MPU 6050 с Launchpad. Например, I2C, Wire и библиотека MPU6050, которая создает из MPU6050 объект с именем `accelgyro`. Библиотека MPU6050.h содержит класс, называемый MPU6050, предназначенный для отправки и получения данных от датчика:

```
#include "Wire.h"

#include "I2Cdev.h"
#include "MPU6050.h"

MPU6050 accelgyro;
```

В следующем разделе мы запускаем I2C и последовательную связь с MPU 6050. В результате датчик начнет передавать значения через последовательный порт. Скорость последовательной передачи составляет 115 200 бод. `Setup_MPU6050()` – это пользовательская функция инициализации связи MPU 6050:

```
void setup()
{
  // Последовательный порт со скоростью передачи 115 200 бод
  Serial.begin(115200);
  Setup_MPU6050();
}
```

Следующий раздел является определением функции `Setup_MPU6050()`. Библиотека Wire позволяет взаимодействовать с устройствами через I2C. MPU 6050 использует I2C для передачи данных. Функция `Wire.begin()` запустит связь I2C между MPU6050 и launchPad. Кроме того, данная функция инициализирует устройство MPU 6050 с помощью метода `initialize()`, определенного в классе MPU6050. Если все пройдет успешно, то на экране появится сообщение **connection successful** («соединение успешно установлено»). В противном случае вы увидите сообщение **connection failed** («связь прервана»):

```
void Setup_MPU6050()
{
  Wire.begin();
    // инициализация устройства
  Serial.println("Initializing I2C devices...");
  accelgyro.initialize();

  // проверка соединения
    Serial.println("Testing device connections...");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful" : "MPU6050 connection failed");
}
```

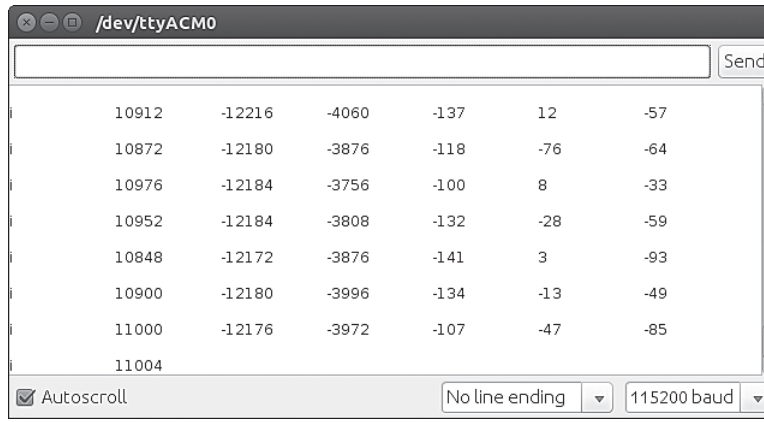
Следующий код-функция `loop()` непрерывно считывает значение датчика и публикует значения через последовательный порт. Функция `Update_MPU6050()`, ответственная за публикацию обновленного значения от MPU 6050:

```
void loop()
{

  // Обновление данных датчика MPU 6050
  Update_MPU6050();
}
```

`Update_MPU6050()` определяется следующим образом. Функция объявляет шесть переменных для обработки значений по трем осям акселерометра и гироскопа. Функция `getMotion6()` в классе MPU 6050 отвечает за считывание новых значений с датчика. После прочтения он будет публиковать данные через последовательный порт:

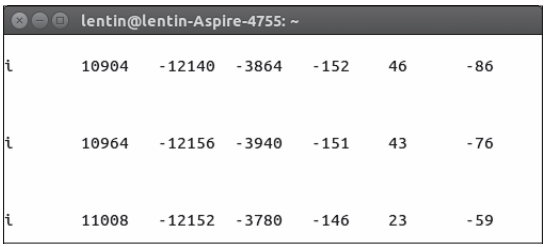
```
void Update_MPU6050()
{
   int16_t ax, ay, az;
  int16_t gx, gy, gz;

      // читать accel/gyro измерений от устройства
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    // вывод значений accel/gyro x/y/z по отдельности
    Serial.print("i");Serial.print("\t");
    Serial.print(ax); Serial.print("\t");
    Serial.print(ay); Serial.print("\t");
    Serial.print(az); Serial.print("\t");
    Serial.print(gx); Serial.print("\t");
    Serial.print(gy); Serial.print("\t");
    Serial.println(gz);
    Serial.print("\n");
}
```

Вывод данных из последовательного монитора показан здесь:



Вывод MPU 6050 на последовательный монитор

Мы можем прочитать эти значения с помощью кода, написанного на Python. Этот код мы использовали для ультразвукового датчика. Ниже приведен снимок экрана терминала при запуске скрипта Python.



Вывод MPU 6050 в терминале Linux
(изображение текстового поля терминала инвертировано)

Итоги

В этой главе мы обсудили взаимодействие с двигателями, которые были использованы в нашем роботе. Также было рассмотрено взаимодействие двигателя и энкодера с контроллером Tiva C LaunchPad. Обсудили код контроллера, позволяющий подключить датчики и двигатели. В дальнейшем, если для робота потребуются повышенная точность и усиленный крутящий момент, обратитесь к сервоприводам Dynamixel, которые могут заменить двигатели постоянного тока. Также мы обсудили применение роботизированных датчиков, которые были использованы в нашем роботе. Были рассмотрены ультразвуковые датчики расстояния, инфракрасные датчики расстояния и инерциальный блок (IMU). Эти три датчика помогают роботу ориентироваться в помещении. Мы также обсудили основной код, позволяющий подключить эти датчики к микроконтроллеру Tiva C LaunchPad. Далее, в следующей главе, мы уделим больше внимания взаимодействию с устройствами видеобзора и аудиодатчиками и использованию для этих целей Python.

Вопросы

1. Что такое цепь H-моста?
2. Что такое квадратурный энкодер?
3. Что такое схема кодирования 4X?
4. Как рассчитать смещение по данным энкодера?
5. Каковы особенности привода Dynamixel?
6. Каков принцип работы ультразвуковых датчиков?
7. Как вычислить расстояние от ультразвукового датчика до препятствия?
8. Каков принцип работы ИК-датчика расстояния?

Дополнительная информация

Подробнее о программировании Energia читайте по ссылке: http://energia.nu/guide/.

Глава 7

Согласование датчиков зрения с ROS

В предыдущей главе мы рассматривали некоторые роботизированные датчики, используемые в нашем роботе, и их взаимодействие с микроконтроллером LaunchPad. Основная тема этой главы – датчики машинного зрения и их интерфейс.

Робот, который мы проектируем, оборудован трехмерным датчиком технического зрения, который взаимодействует с помощью таких библиотек технического зрения, как **Open Source Computer Vision** (OpenCV), **Open Natural Interaction** (OpenNI) и **Point Cloud Library** (PCL). Основные задачи, возлагаемые на датчик технического трехмерного зрения, следующие: автономная навигация, избегание препятствий, обнаружение предмета, отслеживание (определение) людей и т. д.

Мы также рассмотрим взаимодействие датчиков зрения с ROS и обработку воспринимаемых изображений. Для этой цели воспользуемся библиотеками технического зрения, такими как OpenCV.

В последнем разделе этой главы мы рассмотрим алгоритм отображения и локализации, который будет использован в нашем роботе. Этот алгоритм называется **SLAM** (**одновременная локализация и отображение**), реализуем его с использованием датчика 3D vision, ROS и библиотек обработки изображений.

В этой главе мы рассмотрим следующие темы:

- список роботизированных датчиков технического зрения и библиотек изображений;
- введение в формат OpenCV с поддержкой OpenNI. и PCL;
- интерфейс ROS-OpenCV;
- обработка облака точек с помощью интерфейса PCL-ROS;
- преобразование данных облака точек в данные лазерного сканирования;
- введение в SLAM.

Технические требования

Для выполнения примера, приведенного в этой главе, вам понадобятся операционная система Ubuntu 16.04 с установленной Ros Kinetic, веб-камера и сте-

реокамера. В первом разделе мы познакомимся с двухмерными и трехмерными датчиками технического зрения. Эти датчики можно приобрести через интернет и применять в разных роботах.

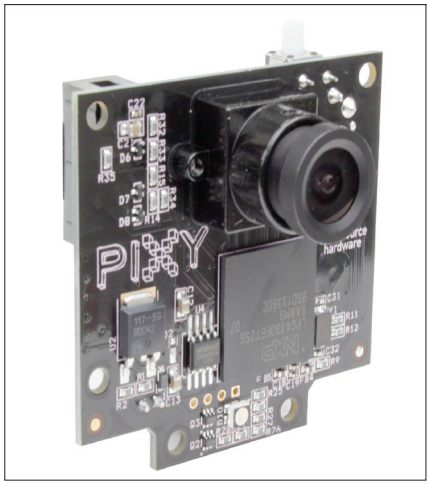
Список робототехнических датчиков зрения и библиотек для работы с изображением

Двухмерный датчик изображения, который может заменить обычная камера, обеспечивает двухмерное изображение окружающей среды, тогда как трехмерный датчик изображения обеспечивает, кроме двухмерного изображения, дополнительный параметр, называемый глубиной каждой точки изображения. Мы можем получить расстояние от датчика трехмерного изображения к каждой точке по осям x, y и z.

На рынке предлагается много датчиков технического зрения. Мы рассмотрим те датчики, которые будут использованы в нашем роботе.

Pixy2/CMUcam5

На следующем рисунке показан двухмерный датчик под названием Pixy/CMU Cam 5 (http://www.cmucam.org/). Датчик способен обнаружить с высокой точностью цветные объекты, и ему даже не помешает высокая скорость движения. Датчик может подключаться к плате Arduino. Этот датчик может быть использован для быстрого обнаружения объекта, а пользователь может указать, какой объект он должен отслеживать. В модуле Pixy установлены датчики CMOS и NXP (http://www.nxp.com/) и процессор для обработки изображений:



Вид модуля Pixy/CMUcam 5

Вид модуля Pixy2/CMUcam 5 (http://a.co/fZtPqck)

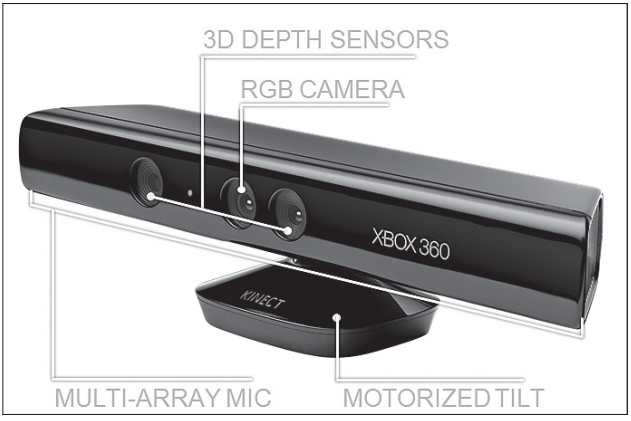
Веб-камера Logitech C920

На следующем рисунке вы увидите популярную веб-камеру Logitech, которая может захватывать изображения с разрешением до 5 мегапикселей и HD-видео:

Веб-камера Logitech HD C920 (http://a.co/02DUUYd)

Kinect 360

Теперь мы рассмотрим датчики трехмерного технического зрения, доступные на рынке. Наиболее популярными датчиками являются Kinect, Intel RealSense серии D400 и Orbbec Astra.



Датчик Kinect 360

Kinect – это 3D-датчик видеообзора, который используется вместе с игровой консолью Xbox 360 от Microsoft. Он содержит камеру RGB, инфракрасный прожектор, датчик глубины, блок с микрофоном и встроенный в основание камеры двигатель, позволяющий наклонять камеры под нужным углом. RGB-камера и камера глубины захватывает изображение с разрешением 640 × 480 с частотой 30 Гц. Камера RGB захватывает цветное двухмерное изображение, тогда как камера глубины захватывает монохромное изображение глубины. Kinect измеряет глубину в пределах от 0,8 до 4 м.

Некоторые применения Kinect – захват движения в трех плоскостях, структурное отслеживание, распознавание лиц и речи.

Kinect может подключаться к ПК с через интерфейс USB 2.0 и программируется с помощью Kinect SDK, OpenNI и OpenCV. Библиотека Kinect SDK работает только с платформами Windows, разрабатывается и поставляется корпорацией «Майкрософт». Остальные две библиотеки (OpenNI и OpenCV) с открытым исходным кодом и доступны для всех платформ. Мы используем одну из последних версий Kinect, поддерживается она только на платформе Windows (подробнее об этом читайте по адресу: https://www. microsoft.com/en-us/download/details.aspx?id=40278).

Производство датчиков серии Kinect прекращено, но вы все еще можете найти датчик на Amazon и eBay.

Intel RealSense серии D400

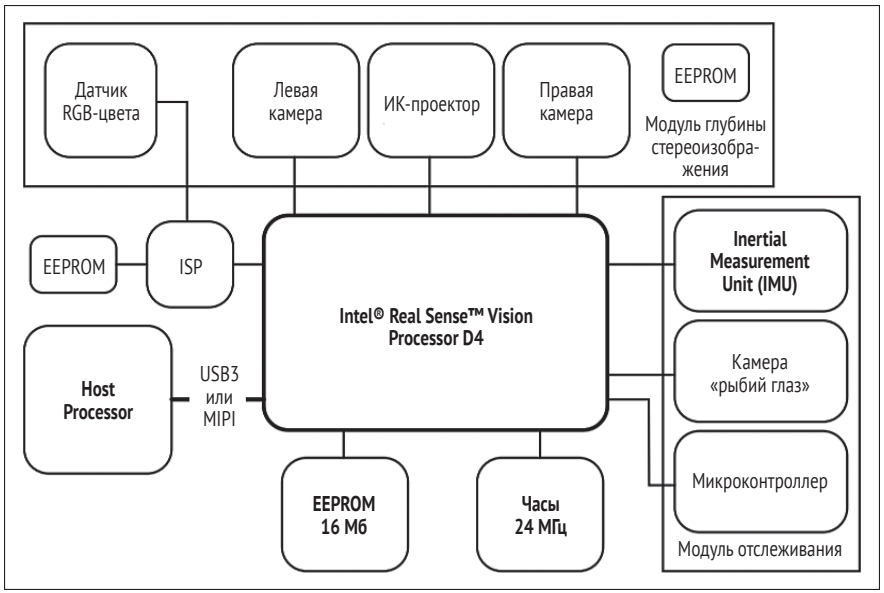
Датчики глубины Intel RealSense D400 имеют стереокамеры, которые идут с ИК-проектором для улучшения данных глубины (см. программное обеспечение https://software.intel.com/en- us/realsense/d400 более детально), как показано на рисунке.

Intel RealSense D400 series (https://realsense.intel.com/)

Наиболее популярные модели датчиков серии D400 – это D415 и D435. На рисунке сверху показан датчик D415. Ниже – датчик D435. Каждый из датчиков состоит из стереопары, RGB-камеры и ИК-прожектора. Стереопара камер вычисляет глубину окружающей среды с помощью ИК-прожектора.

Главная особенность этой камеры глубины в том, что она может работать как в помещении, так и на открытом воздухе. Камера предоставляет поток глубины изображений с разрешением 1280 × 720 на 90 кадров/с, а разрешение RGB-камеры составляет до 1920 × 1080. Intel RealSense взаимодействует с ПК через USB-C, который обеспечивает высокоскоростную передачу данных между датчиком и компьютером. Датчик имеет небольшие размеры и малый вес, что позволяет использовать его в мобильных роботах как датчик технического зрения.

Устройства Kinect и Intel RealSense функционально одинаковы, за исключением функции распознавания речи. Эти датчики работают с платформами Windows, Linux и Mac. Для работы устройства потребуются ROS, OpenNI и OpenCV. На следующем рисунке показана блок-схема камеры серии D400.



Блок-схема камеры серии D400

Справочную информацию по Intel RealSense вы найдете на странице по адресу: https://software.intel.com/sites/default/files/Intel_RealSense_Depth_Cam_D400_Series_Datasheet.pdf.

Чтобы прочитать исследовательскую работу о датчике глубины Intel Realsense, перейдите по следующей ссылке: https://arxiv.org/abs/1705.05548.

Пакет Intel RealSense SDK находится по ссылке: https://github.com/IntelRealSense/librealsense.

Датчик глубины изображения Orbbec Astra

Альтернативой датчику Kinect является новый датчик **Orbbec**. Этот датчик имеет такие же характеристики, как Kinect, и использует аналогичную технологию для получения глубины изображения. Так же, как и Kinect, он оснащен ИК-прожектором, RGB-камерой и ИК-датчиком. Устройство снабжено и микрофоном для распознавания голосовых команд.

На следующем рисунке показаны все элементы датчика глубины Orbbec Astra.

Элементы датчика глубины Orbbec Astra

В продаже можно найти датчики Astra двух моделей: Astra и Astra S. Основная разница между эти двумя моделями – глубина обзора. Astra имеет глубину обзора от 0,6 до 8 м, а глубина обзора Astra S – от 0,4 до 2 м. Astra S лучше всего подходит для 3D-сканирования, тогда как Astra может быть применена в робототехнике. Размер и вес Astra гораздо меньше, чем у Kinect. Эти две модели поставляют данные глубины изображения по RGB-каналам с разрешением 640 × 480 на 30 кадров/с. Можно использовать и более высокое разрешение, например 1280 × 960. Но в этом случае понижается частота кадров. Эти датчики, так же как и датчики Kinect, могут отслеживать структуру пространства.

Датчик совместим с фреймворком OpenNI, поэтому приложение, работающее с OpenNI, может работать и с этим датчиком. Мы данный датчик применим в нашем роботе.

SDK совместим с Windows, Linux и Mac OS X. Для получения дополнительной информации зайдите на сайт разработчика: https://orbbec3d.com/develop/.

Один из датчиков, который также можно использовать в нашем роботе, – это камера **ZED** (https://www.stereolabs.com/zed/). Эта стереокамера обеспечивает видеосигнал с высоким разрешением и хорошую частоту. Она дороже, чем перечисленные выше датчики технического зрения. Ее цена – около 450 долл. США. Эту камеру можно использовать для высококлассных роботов, где от датчиков требуется высокая точность.

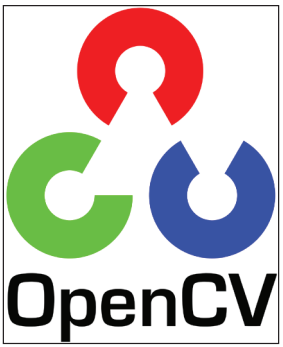
Сопряжение данного датчика с ROS мы рассмотрим в следующем разделе.

Введение в OpenCV, OpenNI и PCL

Давайте поговорим о программных фреймворках и библиотеках, которые мы используем в нашем роботе. Сначала рассмотрим OpenCV. Это одна из библиотек, которые мы будем использовать в роботе для обнаружения препятствия и остальных функций обработки изображений.

Что такое OpenCV?

OpenCV – это открытый исходный код с BSD-лицензированной библиотекой компьютерного зрения, который включает в себя сотни алгоритмов компьютерного зрения. Библиотека в основном ориентирована на компьютерное зрение в реальном времени. Разработана в исследовательской корпорации Intel в России и сейчас активно поддерживается Itseez (http://itseez.com/).



Логотип OpenCV

OpenCV в основном написан на C и C++, а его основной интерфейс разработан в C++. OpenCV имеет хорошие интерфейсы в Python, Java, Matlab/Octave и также имеет оболочки и для других языков. Например: C# и Ruby.

В новой версии OpenCV есть поддержка CUDA и OpenCL, ускоряющая работу GPU (https://developer.nvidia.com/cuda-zone).

OpenCV хорошо работает с большинством платформ ОС, таких как Windows, Linux, Mac OS X, Android, ОС FreeBSD, OpenBSD и FreeBSD, iOS и BlackBerry.

В Ubuntu OpenCV и Python были установлены при установке пакетов `ros-kinetic-desktop-full`. Если этот пакет по каким-то причинам не был установлен, примените приведенные ниже команды для установки пакета OpenCV-ROS:

```
$ sudo apt-get install ros-kinetic-vision-opencv
```

Чтобы проверить, установлен ли модуль OpenCV-Python в вашей системе, запустите терминал Linux и введите команду `python`. Будет показан интерпретатор Python. Далее для проверки установки OpenCV выполните в терминале следующие команды:

```
>>> import cv2
>>> cv2.__version__
```

Если эта команда будет выполнена успешно, значит, данная OpenCV установлена в вашей системе. Версия может быть 3.3.x или 3.2.x.

Чтобы попробовать OpenCV в Windows, для установки перейдите по следующей ссылке: https://docs.opencv.org/2.4/doc/tutorials/introduction/windows_install/windows_install.html.
Для установки OpenCV на MAC OS X обратитесь к ссылкам:
https://www.pyimagesearch.com/2016/12/05/macos-install-opencv-3-and-python-3-5/;
https://www.learnopencv.com/install-opencv3-on-macos/.

Основные области применения OpenCV:

- обнаружение объектов;
- распознавание жестов;
- взаимодействие человека и компьютера;
- мобильная робототехника;
- отслеживание движения;
- распознавание лиц.

Теперь мы можем рассмотреть процедуру установки OpenCV в Ubuntu 14.04.2 из исходного кода.

Установка OpenCV в Ubuntu из исходного кода

Есть разные варианты установки OpenCV. Один из вариантов – установка из исходных кодов. Как выполнить такую установку, узнайте, перейдя по ссылке: https://docs.opencv.org/trunk/d7/d9f/tutorial_linux_install.html.

Для работы с примерами, приведенными в этой главе, лучше всего использовать OpenCV, установленную вместе с ROS.

Чтение и отображение изображения с помощью интерфейса Python-OpenCV

Первый пример загрузит изображение в оттенках серого и отобразит его на экране. В следующем разделе кода мы импортируем модуль `numpy` для обработки массива изображений. Модуль `cv2` – это оболочка OpenCV для Python, которую мы можем использовать для доступа к API OpenCV Python. NumPy – это расширение языка программирования Python, добавляющее поддержку больших многомерных массивов и матриц, а также большую библиотеку высокоуровневых математических функций для работы с этими массивами (более подробная информация – на сайте https://pypi.python.org/pypi/numpy):

```
#!/usr/bin/env python
import numpy as np
import cv2
```

Следующая функция считывает изображение `robot.jpg` и загружает его в полутоновом формате. Первый аргумент функции `cv2.imread()` – это имя изображения. Следующий аргумент является флагом, который определяет тип цвета

загруженного изображения. Если флаг > 0, то изображение возвращает три цветовых канала цветного изображения (RGB). Если флаг = 0, загружаемое изображение будет черно-белым. Если флаг < 0, то загруженное изображение будет возвращено:

```
img = cv2.imread('robot.jpg', 0)
```

В следующем разделе кода будет показано, как с помощью функции `imshow()` прочитать изображение. Функция `cv2.waitKey(0)` является функцией привязки клавиатуры. Ее аргументом является время в миллисекундах. Если аргумент равен 0, функция будет ждать нажатия клавиши.

```
cv2.imshow('image',img)
cv2.waitKey(0)
```

Функция `cv2.destroyAllWindows()` закрывает все ранее открытые окна:

```
cv2.destroyAllWindows()
```

Сохраните предыдущий код с именем `image_read.py` и скопируйте файл `robot.jpg` в папку, в которой расположен код. Выполните код с помощью следующей команды:

```
$python image_read.py
```

Выходные данные загрузят изображение в оттенках серого, потому что в качестве значения функции `imread()` мы использовали 0:

Вывод прочитанного кода изображения

В следующем примере будет предпринята попытка использовать открытую веб-камеру. Программа завершит работу, когда пользователь нажмет любую кнопку.

Захват видео с веб-камеры

Следующий код захватит изображение с помощью веб-камеры с именем устройства: `/dev/video0` или `/dev/video1`.

Для захвата изображения с камеры нам нужно импортировать модули NumPy и cv2:

```
#!/usr/bin/env python
import numpy as np
import cv2
```

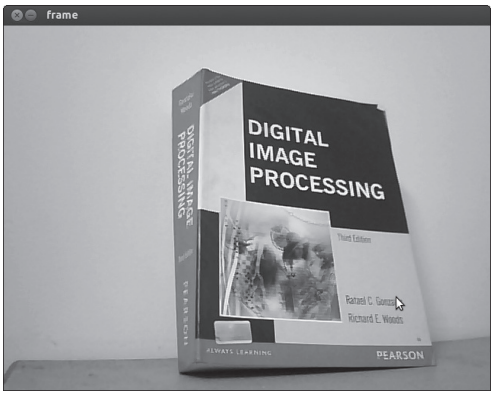
Следующая функция создает объект `VideoCapture`. Класс `VideoCapture` используется для захвата видео с видеофайлов или камер. Аргумент инициализации класса `VideoCapture` – индекс камеры или название видеофайла. Устройство `index` – это просто число для указания камеры. Первый индекс камеры равен 0 и имеет имя устройства `/dev/video0`. Поэтому в следующем коде мы поставим 0:

```
cap = cv2.VideoCapture(0)
```

Следующий раздел кода зациклен для чтения кадров изображения из объекта VideoCapture и показывает каждый кадр. При нажатии любой клавиши цикл прервется:

```
while(True):
    # Захват кадра
    ret, frame = cap.read()
    # Отображение результирующего кадра
    cv2.imshow('frame',frame)
    if cv2.waitKey(10):
        break
```

Ниже показан скриншот вывода программы:



Выход видеозахвата

Вы можете узнать больше о работе OpenCV-Python, перейдя по следующей ссылке: http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html.

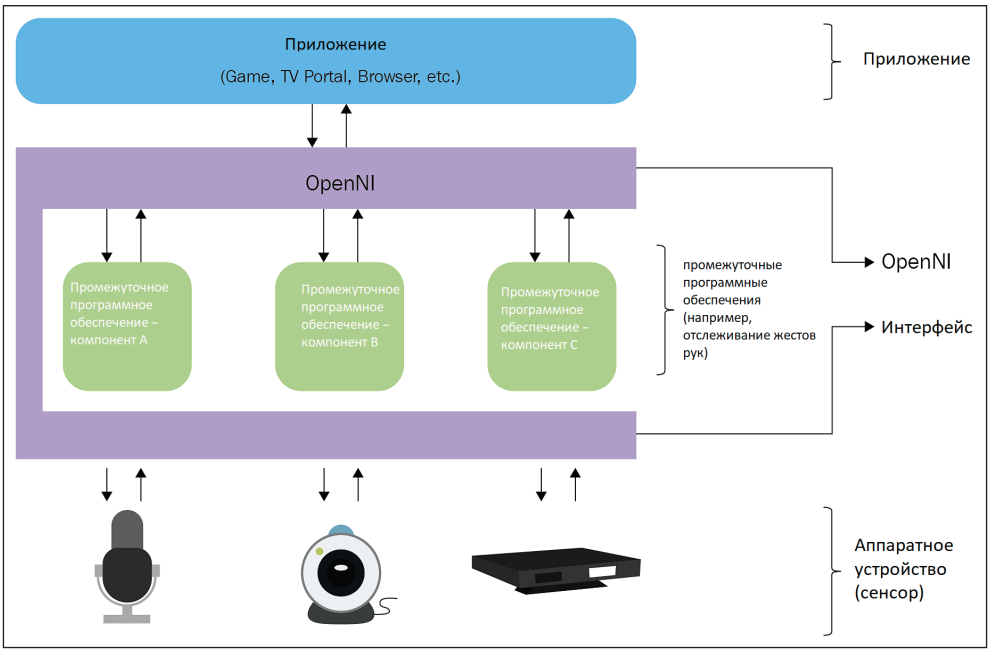
Далее мы рассмотрим библиотеку OpenNI и ее применение.

Что такое OpenNI?

OpenNI – это мультиязычный, кроссплатформенный фреймворк, содержащий API для создания приложений и использующий **естественное взаимодействие** (NI) (чтобы узнать больше, перейдите по ссылке https://structure.io/openni). Естественное взаимодействие – это способ, благодаря которому люди общаются между собой естественным образом: с помощью жестов, выражений и движений, – и открывают для себя мир, оглядываясь вокруг и манипулируя физическими объектами и материалами.

API OpenNI состоит из набора интерфейсов, которые используются для создания приложений NI.

На следующем рисунке показано трехслойное представление библиотеки OpenNI.



Архитектура программной поддержки OpenNI

Верхний слой представляет прикладной уровень, реализующий естественное взаимодействие на основе взаимодействия. Средний слой – это слой под-

держки OpenNI. Он предоставляет интерфейсы связи, взаимодействующие с датчиками и промежуточными компонентами, анализирующими данные, получаемые от датчиков. Middleware можно использовать для полного анализа тела, положения руки, обнаружения жестов и т. д. Одним из примеров среднего слоя является NITE, который может обнаружить жест и структуру (http://www.openni.ru/files/nite/index.html).

На нижнем уровне находятся аппаратные средства, захватывающие визуальные и звуковые элементы сцены. Нижний уровень включает в себя 3D-датчики, RGB-камеры, ИК-камеру и микрофон.

Последняя версия OpenNI2 поддерживает такие датчики, как ASUS Xtion Pro. Первая версия OpenNI в основном поддерживала датчик Kinect 360.

OpenNI кроссплатформенный, успешно скомпилирован и развернут для ОС Linux, Mac ОС X и Windows.

В следующем разделе мы рассмотрим, как установить OpenNI на Ubuntu 14.04.2.

Установка OpenNI в Ubuntu

Мы можем установить библиотеку OpenNI вместе с пакетами ROS. ROS уже взаимодействует с OpenNI, но при полной установке Ros desktop пакеты OpenNI могут и не установиться. В этом случае нам OpenNI следует установить с помощью менеджера пакетов.

Следующая команда установит библиотеку ROS-OpenNI (которая в основном поддерживается сенсором Kinect Xbox 360) в Kinetic и Melodic:

```
$ sudo apt-get install ros-<version>-openni-launch
```

Команда, показанная ниже, установит библиотеку ROS-OpenNI 2 (которая в основном поддерживается компаниями ASUS Xtion Pro и Carmine):

```
$ sudo apt-get install ros-<version>-openni2-launch
```

Исходный код и последняя сборка с поддержкой OpenNI для Windows, Linux и MacOS X доступны на http://structure.io/openni.

В следующем разделе мы рассмотрим, как установить PCI.

Что такое PCL?

Облако точек – это набор точек данных в пространстве, представляющих 3D-объект или среду. Обычно облако точек создается с помощью датчиков глубины, таких как Kinect и LIDAR. PCL (Point Cloud Library) – это крупномасштабный открытый проект для обработки 2D/3D-изображений и облаков точек. Фреймворк PCL содержит множество алгоритмов, которые выполняют фильтрацию, оценку характеристик, реконструкцию поверхности, регистрацию, подгонку модели и сегментацию. Используя эти методы, мы можем обрабатывать облако точек, извлекать ключ идентификаций для распознавания объектов в мире на основе их геометрических форм, создавать поверхности из облаков точек и визуализировать их.

Логотип PCL

PCL выпущен под лицензией BSD, т. е. с открытым исходным кодом и бесплатным для коммерческого использования или исследования. PCL является кроссплатформенным приложением и успешно скомпилирован и развернут для Linux, Mac OS X, Windows и Android/iOS.

Скачать PCL можно по следующей ссылке: http://pointclouds.org/downloads/.

PCL уже интегрирован в ROS. Библиотека PCL и его ROS устанавливаются вместе с полной установкой ROS. В предыдущей главе мы обсудили, как установить полный комплект ROS. PCL – это основное приложение ROS для обработки трехмерных изображений.

Чтобы подробнее узнать о пакете PCL, перейдите по следующей ссылке: http://wiki.ros.org/pcl.

Программирование Kinect с использованием Python ROS, OpenCV и OpenNI

Давайте посмотрим, как мы можем взаимодействовать с датчиком Kinect в ROS. ROS комплектуется драйвером OpenNI, который с помощью Kinect делает выборку RGB и глубины изображения.

Этот пакет может быть использован для Microsoft Kinect, PrimeSense Carmine, Asus Xtion Pro и Pro Live.

В основном этот драйвер публикует исходную глубину изображения, RGB и потоки ИК-изображений. Пакет `openni_launch` установит такие пакеты, как `openni_camera` и `openni_launch`.

Пакет `openni_camera` является драйвером Kinect, который публикует необработанные данные с информационных датчиков, в то время как пакет `openni_launch` содержит файлы запуска ROS. Эти файлы запускают несколько узлов одновременно и публикуют такие данные, как необработанная глубина, изображения RGB и IR, а также облако точек.

Как запустить драйвер OpenNI

Датчик Kinect подключается к компьютеру через USB. Чтобы убедиться, что компьютер устройство обнаружил, введите в терминал команду `dmesg`. После подключения Kinect для получения данных с устройства следует запустить драйвер OpenN ROS. Следующая команда откроет устройство OpenNI и загрузит все узлы для преобразования необработанных потоков глубина/RGB/IR в глубину изображений, разницу изображений и облака точек. Пакет узлов ROS предназначен для запуска нескольких алгоритмов в одном процессе с нулевым переносом копий между алгоритмами:

```
$ roslaunch openni_launch openni.launch
```

После запуска драйвера с помощью следующей команды мы можем отобразить список тем, опубликованных драйвером:

```
$ rostopic list
```

Изображение RGB можно просмотреть с помощью инструмента ROS с названием `image_view`:

```
$ rosrun image_view image_view image:=/camera/rgb/image_color
```

В следующем разделе мы увидим, как взаимодействуют эти изображения в процессе обработки изображений в OpenCV.

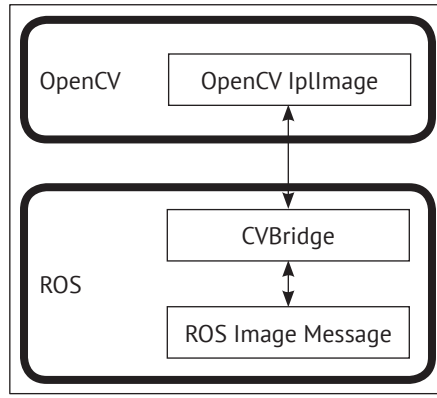
Интерфейс ROS в формате OpenCV

OpenCV также интегрирован в ROS, в основном для обработки изображений. Стек `vision_opencv` ROS включает в себя полную библиотеку OpenCV и ее интерфейс в ROS.

Метапакет `vision_opencv` состоит из нескольких пакетов:

- `cv_bridge`: содержит класс CvBridge; этот класс преобразует сообщения из ROS в тип данных изображения OpenCV и наоборот;
- `image_geometry`: содержит коллекцию методов обработки изображений и геометрию пикселей.

На следующей схеме показано, как OpenCV взаимодействует с ROS.



Интерфейс OpenCV – ROS

Типами данных изображений формата OpenCV являются `IplImage` и `Mat`. Если нам потребуется в ROS работать с OpenCV, нам необходимо передаваемые изображения преобразовать в `IplImage` или `Mat` ROS.

Пакет ROS `vision_opencv` содержит класс `CvBridge`; этот класс преобразует `IplImage` в образ ROS, и наоборот. Как только мы получим темы изображений

ROS от любого датчика технического зрения, задействуем ROS CvBridge для преобразования изображения из темы ROS в формат Mat или IplImage.

В следующем разделе показано, как создать ROS-пакет. Этот пакет содержит узел для подписки на RGB-изображения и глубину изображения, обработку RGB-изображений с целью обнаружения краев и отображение всех изображений после их преобразования в тип изображения, эквивалентный OpenCV.

Создание пакета ROS с поддержкой OpenCV

Мы можем создать пакет sample_opencv_pkg со следующими зависимостями: sensor_msgs, cv_bridge, rospy и std_msgs. Зависимость sensor_msgs определяет сообщения для часто используемых датчиков, включая камеры и сканирующий лазерный дальномер; cv_bridge является интерфейсом OpenCV ROS.

Следующая команда создаст пакет ROS с предыдущими зависимостями:

```
$ catkin-create-pkg sample_opencv_pkg sensor_msgs cv_bridge rospy std_msgs
```

После создания пакета создайте папку scripts внутри пакета и сохраните код, указанный в следующем разделе.

Вывод изображений с Kinect с помощью Python, ROS и cv_bridge

Первый раздел кода Python приведен ниже. Главным образом этот код включает в себя импортированные модули rospy, sys, cv2, sensor_msgs, cv_bridge и numpy.

Sensor_msgs импортирует тип данных ROS Image и CameraInfo. Модуль cv_bridge импортирует класс CvBridge для преобразования типа данных ROS image в тип данных OpenCV, и наоборот:

```
import rospy
import sys
import cv2
import cv2.cv as cv
from sensor_msgs.msg import Image, CameraInfo
from cv_bridge import CvBridge, CvBridgeError
import numpy as np
```

Следующий раздел кода является определением класса CvBridge в Python с функцией демонстрации. Класс называется cvBridgeDemo:

```
class cvBridgeDemo():
    def __init__(self):
        self.node_name = "cv_bridge_demo"
        # инициализировать узел ros
        rospy.init_node(self.node_name)

        # что мы делаем во время выключения
        rospy.on_shutdown(self.cleanup)

        # Создание объекта cv_bridge
        self.bridge = CvBridge()

        # Подпишитесь на темы изображения и глубины камеры и установите
        # соответствующие обратные вызовы
        self.image_sub =
```

```
rospy.Subscriber("/camera/rgb/image_raw", Image,
self.image_callback)      self.depth_sub =
rospy.Subscriber("/camera/depth/image_raw", Image,
self.depth_callback)

# обратный вызов выполняется во время паузы таймера
        rospy.Timer(rospy.Duration(0.03), self.show_img_cb)

        rospy.loginfo("Waiting for image topics...")
```

Зависимость `cv_bridge` является интерфейсом OpenCV ROS. Ниже приведен код обратного вызова для отображения фактического изображения RGB, обработанного изображения RGB и изображения глубины:

```
def show_img_cb(self,event):
    try:


        cv2.namedWindow("RGB_Image", cv2.WINDOW_NORMAL)
        cv2.moveWindow("RGB_Image", 25, 75)
        cv2.namedWindow("Processed_Image", cv2.WINDOW_NORMAL)
        cv2.moveWindow("Processed_Image", 500, 75)

        # И одно для изображения глубины
        cv2.moveWindow("Depth_Image", 950, 75)
        cv2.namedWindow("Depth_Image", cv2.WINDOW_NORMAL)

        cv2.imshow("RGB_Image",self.frame)
        cv2.imshow("Processed_Image",self.display_image)
        cv2.imshow("Depth_Image",self.depth_display_image)
        cv2.waitKey(3)
    except:
        pass
```

Следующий код создает функцию обратного вызова цветного изображения из Kinect. При поступлении в тему цветного изображения `/camera/rgb/image_color` будет вызываться эта функция, которая обработает границы цвета для обнаружения края и покажет их в сочетании с оригинальным изображением:

```
def image_callback(self, ros_image):
    # Использование cv_brige  для преобразования образа ROS в формат Opencv
    try:
        self.frame = self.bridge.imgmsg_to_cv2(ros_image, "bgr8")
    except CvBridgeError, e:
        print e
   pass

    # Преобразуйте изображение в массив Numpy, так как большинство функций cv2
    # требуют массивы Numpy.
    frame = np.array(self.frame, dtype=np.uint8)
    # Обработайте кадр с помощью функции process_image()
    self.display_image = self.process_image(frame)
```

Следующий код с помощью Kinect вызывает функцию обратного вызова глубины изображения. Функция обратного вызова будет вызвана, когда данные

по глубине изображения поступят на тему /camera/depth/raw_image. С помощью этой функции будет отображено необработанное изображение глубины:

```
def depth_callback(self, ros_image):
 # Использование cv_brige() для преобразования образа ROS в формат Opencv
 try:
     # Глубина изображения - одноканальные изображения float32
     depth_image = self.bridge.imgmsg_to_cv2(ros_image, "32FC1")
 except CvBridgeError, e:
     print e
pass
 # Преобразуйте изображение в массив Numpy, так как большинство функций cv2
 # требуют массивы Numpy.
 depth_array = np.array(depth_image, dtype=np.float32)
 # Нормализация глубины изображения между 0 (черный) и 1 (белый)
 cv2.normalize(depth_array, depth_array, 0, 1, cv2.NORM_MINMAX)
 # Обработка глубины изображения.
 self.depth_display_image = self.process_depth_image(depth_array)
```

Следующая функция называется process_image(), преобразует цветное изображение в серое, затем размывает изображение и находит границы с помощью **фильтра Кэнни**:

```
def process_image(self, frame):
    # Преобразование в оттенки серого
    grey = cv2.cvtColor(frame, cv.CV_BGR2GRAY)

    # Размытие изображения
    grey = cv2.blur(grey, (7, 7))

    # Вычислить границы через фильтр Кэнни
    edges = cv2.Canny(grey, 15.0, 30.0)

    return edges
```

Приведенная ниже функция называется process_depth_image(). Она просто возвращает глубину границы:

```
def process_depth_image(self, frame):
    # Просто верните изображение raw для этой демонстрации
    return frame
```

Функция, код которой показан ниже, закроет окно изображения, когда узел завершит работу:

```
def cleanup(self):
    print "Shutting down vision node."
    cv2.destroyAllWindows()
```

Следующий код для функции main(). Он инициализирует класс cvBridgeDemo() и вызывает функцию rospy.spin():

```
def main(args):
    try:
        cvBridgeDemo()
```

```
        rospy.spin()
    except KeyboardInterrupt:
        print "Shutting down vision node."
        cv.DestroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

Сохраните предыдущий код в `cv_bridge_demo.py` и измените разрешение узла с помощью следующей команды. Узел виден только команде `rosrun`, если мы дадим ему права доступа.

```
$ chmod +X cv_bridge_demo.py
```

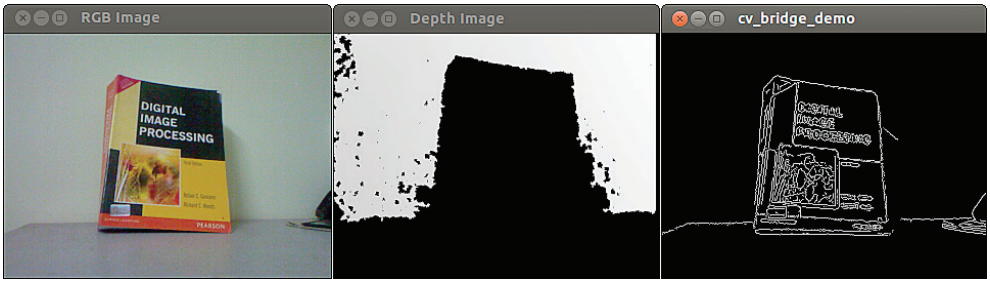
Ниже приведены команды для запуска драйвера и узлов. Запустите драйвер Kinect с помощью следующей команды:

```
$ roslaunch openni_launch openni.launch
```

Запустите узел с помощью следующей команды:

```
$ rosrun sample_opencv_pkg cv_bridge_demo.py
```

Ниже приведен скриншот вывода.



RGB, глубина и изображения края

Согласование Orbbec Astra с ROS

Одной из альтернатив Kinect является Orbbec Astra. Для Astra доступны драйверы ROS. Рассмотрим настройку этого драйвера и получим изображение, глубину и облако точек от этого датчика.

Установка драйвера Astra-ROS

Полные инструкции по настройке драйвера Astra-ROS в Ubuntu приведены на https://github.com/orbbec/ros_astra_camera и http://wiki.ros.org/Sensors/OrbbecAstra. После установки драйвер запускается следующей командой:

```
$ roslaunch astra_launch astra.launch
```

Драйвер Astra также можно установить из репозитория пакетов ROS. Ниже представлена команда для установки этих пакетов:

```
$ sudo apt-get install ros-kinetic-astra-camera
$ sudo apt-get install ros-kinetic-astra-launch
```

После установки этих пакетов необходимо обнаружить и идентифицировать камеру, чтобы с устройством можно было работать. Как это сделать, описано в http://wiki.ros.org/astra_camera. Темы ROS, созданные с помощью этого драйвера, можно проверить с помощью команды `rostopic` в терминале. Кроме того, для обработки изображений мы можем использовать тот же код Python, что был рассмотрен в предыдущем разделе.

Работа с облаками точек с помощью Kinect, ROS, OpenNI и PCL.

Трехмерное облако точек – это способ представления 3D-среды и 3D-объектов в виде геометрических координат по осям x, y и z. Мы можем получить облако точек из различных источников: облако точек можно создать программным способом либо синтезировать из данных датчика глубины или лазерных сканеров.

PCL изначально поддерживает интерфейсы OpenNI 3D. Таким образом, он может получать и обрабатывать данные с таких устройств, как 3D-камеры Prime Sensor, Microsoft Kinect или Asus Xtion Pro.

PCL включен в полную настольную установку ROS. Давайте посмотрим, как создать и сделать видимым облако точек в RViz, инструменте визуализации данных в ROS.

Открытие устройства и создание облака точек

Используя нижеприведенную команду, откройте новый терминал, запустите драйвер ROS OpenNI, узлы генераторов облака точек.

```
$ roslaunch openni_launch openni.launch
```

Эта команда активирует драйвер Kinect и обрабатывает необработанные данные в удобные выходные данные, такие как облако точек.

При использовании Orbbec Astra применяйте следующую команду:

```
$ roslaunch astra_launch astra.launch
```

Для просмотра облаков точек мы воспользуемся инструментом 3D-отображения RViz. Для запуска RViz воспользуйтесь командой

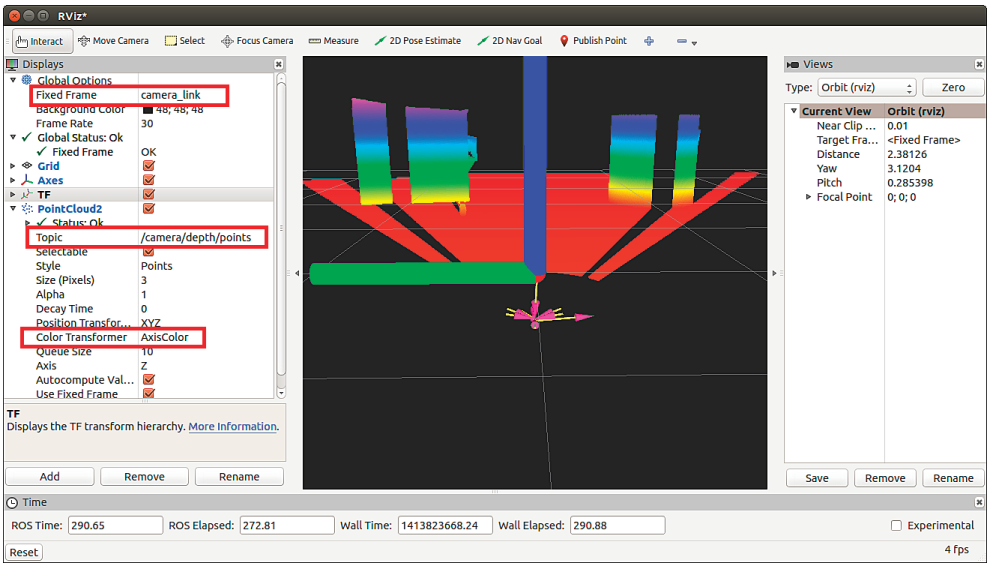
```
$ rosrun rviz rviz
```

В левой части окна RViz отображается панель для настройки глобальных параметров. Настройте в RViz параметры **Fixed Frame** – **camera_link**. Параметр **camera_link** находится в верхней части панели **Global Options**.

На левой боковой панели Global Options нажмите кнопку **Add** (Добавить) и выберите опцию отображения **PointCloud2**. Установите тему в каталог **/camera/depth/points** (это тема для Kinect; она будет отличаться для других датчиков).

Измените настройки **Color Transformer of PointCloud2** на **AxisColor**.

Все настройки показаны на следующем рисунке, в левой части окна RViz и обведены красными прямоугольниками. Кроме настроек, на следующем рисунке показан снимок экрана данных облака точек RViz. Вы можете видеть, что близлежащие объекты отмечены красным цветом, а самые дальние объекты – фиолетовым и синим. Объекты, установленные перед Kinect, оказались кубом и цилиндром.



Отображение данных облака точек в Rviz

Преобразование данных облака точек в данные лазерного сканирования

Мы в этом роботе используем Astra, чтобы дублировать функцию дорогостоящего лазерного сканера. Данные облака точек с помощью пакета ROS обрабатываются и преобразуются в эквивалент данных лазерного сканера `depthimage_to_laserscan` (http://wiki.ros.org/depthimage_to_laserscan).

Вы можете установить этот пакет из исходного кода или использовать менеджер пакетов Ubuntu. Вот команда для установки этого пакета из менеджера пакетов Ubuntu:

```
$ sudo apt-get install ros-<version>-depthimage-to-laserscan
```

Функция этого пакета состоит в том, чтобы нарезать разделы данных облака точек и преобразовать их в эквивалентный тип данных для лазерного сканирования. Тип сообщения Ros `sensor_msgs/LaserScan` используется для публикации данных лазерного сканирования. Пакет `depthimage_to_laserscan` выполнит это преобразование и сымитирует данные лазерного сканера. Формируемые данные лазерного сканирования можно увидеть в RViz. Чтобы запустить преобразование, мы должны запустить конвертор nodelets, который выполнит эту операцию. Это мы должны указать в нашем файле запуска.

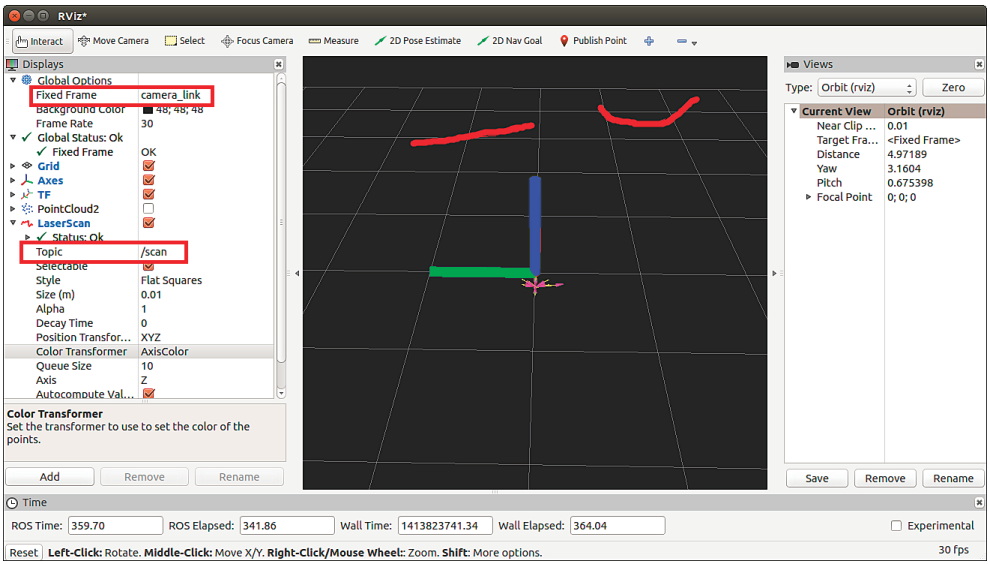
Следующий код запустит файл, который начнет процесс преобразования `depthimage_to_laserscan`:

```
  <!-- Fake laser -->
  <node pkg="nodelet" type="nodelet"
 name="laserscan_nodelet_manager" args="manager"/> <node pkg="nodelet" type="nodelet"
 name="depthimage_to_laserscan"
args="load depthimage_to_laserscan/DepthImageToLaserScanNodelet
 laserscan_nodelet_manager">
    <param name="scan_height" value="10"/>
    <param name="output_frame_id" value="/camera_depth_frame"/>
    <param name="range_min" value="0.45"/>
    <remap from="image" to="/camera/depth/image_raw"/>
    <remap from="scan" to="/scan"/>
 </node>
```

Тему изображения глубины можно изменить в каждом датчике. Для этого необходимо изменить название темы в соответствии с темой изображения глубины.

Наряду с запуском конвертера nodelet нам для более качественного преобразования нужно настроить некоторые параметры nodelet. Чтобы получить более подробное описание каждого настраиваемого параметра, см. страницу, расположенную по адресу http://wiki.ros.org/depthimage_to_laserscan.

Ниже показано отображение данных предыдущего лазерного сканирования. Для просмотра результата лазерного сканирования добавьте опцию **LaserScan**. Эта операция похожа на то, как мы добавляли опцию PointCloud2 и изменяли значение темы **LaserScan** на **/sca**.



Отображение данных лазерного сканирования в RViz

Работа в SLAM с помощью ROS и Kinect

Основная цель установки и настройки в нашем роботе датчиков технического зрения – самостоятельная навигация, обнаружение и обход препятствий. SLAM – это используемый в мобильных роботах алгоритм для создания карты неизвестной среды или обновления ранее созданной карты с помощью отслеживания текущего местоположения робота.

Карты используются для планирования траектории робота и навигации по этому пути. С помощью карты робот получит представление об окружающей среде. Основные две задачи в мобильной навигации робота – отображение и локализация.

Картографирование позволяет сформировать профиль препятствий вокруг робота. Благодаря картографированию робот поймет, как выглядит мир. Локализация – это процесс оценки положения робота на создаваемой нами карте.

SLAM извлекает данные из различных датчиков и использует их для построения карт. Датчик зрения 2D/3D-формата можно использовать как входной сигнал для SLAM. 2D-датчики зрения, как и лазерные дальномеры и 3D-датчики, такие как Kinect, в основном используются в качестве входных данных SLAM-алгоритма.

Название библиотеки SLAM – OpenSlam (http://openslam.org/gmapping.html). Эта библиотека интегрирована в ROS как пакет под названием `gmapping`. Пакет `gmapping` предоставляет узел для выполнения лазерной обработки SLAM и называется `slam_gmapping`. Этот пакет поможет создать двухмерную карту данных лазерного сканирования и расположить собранные мобильным роботом данные.

Пакет `gmapping` доступен на http://wiki.ros.org/gmapping.

Чтобы использовать `slam_gmapping`, вам понадобится мобильный робот, который передает данные по одометрии и оборудован зафиксированным в горизонтальном положении лазерным дальномером.

Узел `slam_gmapping` получает сообщения от `sensor_msgs/LaserScan` и `nav_msgs/Odometry` и создает карту (`nav_msgs/OccupancyGrid`). Сгенерированную карту можно получить с помощью темы или сервиса ROS.

Чтобы применить SLAM в нашем Chefbot, мы использовали следующий файл запуска. Этот файл запускает узел `slam_gmapping` и содержит необходимые параметры для начала отображения среды робота:

```
$ rosrun gmapping slam_gmapping scan:=base_scan
```

Итоги

В этой главе мы рассмотрели датчики зрения, которые будут использоваться в нашем роботе. Мы обсуждали использование Kinect, OpenCV, OpenNI и PCL в роботе и их применение. Мы также рассмотрели роль датчиков зрения в навигации робота, популярный метод SLAM и применение его в ROS. В следующей главе мы рассмотрим полное взаимодействие робота и научимся выполнять автономную навигацию нашего Chefbot.

Вопросы

1. Что такое 3D-сенсоры, и чем они отличаются от обычных камер?
2. Каковы основные особенности роботизированной операционной системы?
3. Каково назначение приложений OpenCV, OpenNI и PCL?
4. Что такое SLAM?
5. Что такое RGB-D SLAM, и как это работает?

Дополнительная информация

Для получения информации о пакете robotic vision в ROS перейдите по следующим ссылкам:

- http://wiki.ros.org/vision_opencv;
- http://wiki.ros.org/pcl.

Глава 8

Создание аппаратного обеспечения ChefBot и интеграция программного обеспечения

В главе 3 «Моделирование робота с дифференциальным приводом с помощью ROS и URDF» мы рассмотрели конструкцию шасси ChefBot. В этой главе мы расскажем, как, используя эти детали, данный робот собрать. Также мы рассмотрим окончательное сопряжение датчиков и других электронных компонентов робота с микропроцессором Tiva-C LaunchPad. После подключения мы объясним, как настроить взаимодействие робота с ПК и с помощью SLAM и AMCL осуществить автономную навигацию.

В этой главе будут рассмотрены следующие темы:

- оборудование для создания ChefBot;
- настройка ChefBot ПК и пакетов;
- взаимодействие датчиков ChefBot с Tiva-C LaunchPad;
- встроенный код для ChefBot;
- понимание пакетов ROS ChefBot;
- реализация SLAM на ChefBot;
- автономная навигация в ChefBot.

Технические требования

Для тестирования приложения и кодов в этой главе вам понадобится Ubuntu 16.04 LTS, ПК/ноутбук с установленным пакетом Ros Kinetic.

Вам для сборки робота понадобятся изготовленные ранее детали шасси.

У вас должны быть все датчики и другие аппаратные компоненты, которые должны быть установлены в роботе.

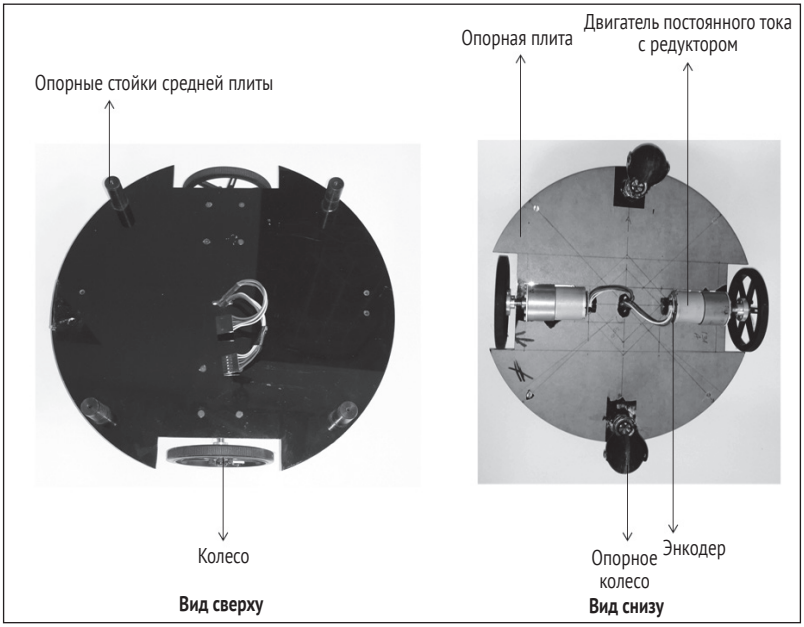
Мы уже обсуждали взаимодействие отдельных компонентов робота и датчиков с микропроцессором Tiva-C LaunchPad. В этой главе мы настроим взаимодействие между компонентами робота и датчиками ChefBot. Далее запрограммируем робот так, чтобы он получал значения от всех датчиков и обменивался с ПК информацией, позволяющей управлять роботом. LaunchPad отправит все значения, получаемые от датчиков через серийный порт, бортовому персональному компьютеру и в ответ получит управляющую информацию (команды возврата, данные о скорости и т. д.) от ПК.

После получения значений датчиков с ПК узел Ros Python получит последовательные значения и преобразует его в ROS-темы. В инсталлированном в бортовом ПК программном обеспечении имеются узлы Python, записывающие данные датчиков и одометра. Чтобы вычислить одометрию робота, данные энкодеров колеса и значения IMU объединяются. Робот обнаруживает препятствия, подписавшись к ультразвуковой теме датчика и развертке лазера и используя узел PID, контролирует скорость вращения двигателя колеса. Этот узел преобразует команду линейной скорости в команду дифференциальной скорости колеса. После запуска этих узлов мы запустим SLAM-отображения карты помещения, а после запуска SLAM будут запущены узлы AMCL для локализации и автономной навигации.

В первом разделе этой главы мы расскажем о процессе сборки оборудования ChefBot с использованием ранее изготовленных деталей его корпуса и электронных компонентов.

Сборка ChefBot из ранее изготовленных деталей и комплектующих

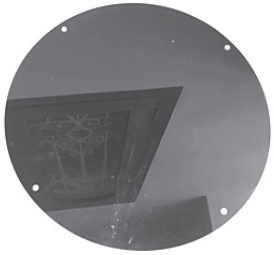
Нижнее основание робота представляет собой круглую плиту с вырезами для колес. На основании закреплены два ведущих двигателя, к валам редукторов которых через ступицы прикреплены ведущие колеса. Кроме этого, к основанию прикреплены два опорных колеса и стойки для крепления средней плиты. На следующем рисунке показан вид плиты основания сверху и снизу с установленными комплектующими.



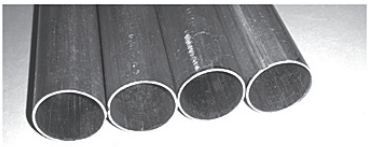
Опорная плита с двигателями, ведущими и опорными колесами

Как уже упоминалось в главе 3, радиус опорной плиты – 150 мм. Двигатели постоянного тока со встроенными редукторами закреплены с противоположных сторон опорной плиты. Под ведущие колеса в опорной плите предусмотрены вырезы. Опорные колеса установлены по краям опорной пластины, по оси, проведенной между центрами опорных колес по отношению к линии оси вращения ведущих колес под углом в 90°. Такое расположение опорных колес обеспечит хороший баланс и поддержку робота. Как уже упоминалось в главе 3, опорные колеса могут быть разных конструкций: шариковые опоры или поворотные колеса с резиновой поверхностью качения. Провода от двух двигателей и энкодеров выводятся наверх через отверстие в центре опорной плиты. Наш робот трехуровневый и состоит из опорной плиты, средней и верхней плит. Средняя плита соединяется с опорной плитой через четыре опорные стойки. В третьей главе были предложены конструкция и чертеж стоек из цельного алюминиевого прутка с резьбовыми отверстиями на торцах. Такая же конструкция у стоек, соединяющих среднюю и верхнюю плиты. Чтобы соединить пакет, состоящий из верхней части нижних опорных стоек – средней плиты – нижней части верхних опорных стоек, предлагается в верхнее резьбовое отверстие нижних стоек вкрутить резьбовые шпильки, надеть на них среднюю опорную плиту и на выступающую часть шпильки накрутить верхние опорные стойки. Вместо цельных опорных стоек можно использовать полые трубки с запрессованными втулками, в центре которых – резьбовое отверстие.

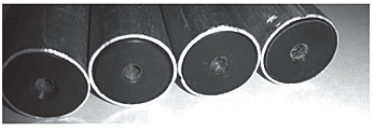
Ниже показаны: средняя плита (без отверстия в центре для прокладывания проводов от датчика Kinnect или Astra); полые трубки; полые трубки с запрессованными резьбовыми втулками. В левом нижнем углу показана собранная конструкция, состоящая из опорной плиты, стоек, средней плиты, двигателей с ведущими колесами и опорными колесами.



Средняя плита



Трубки для опорных стоек



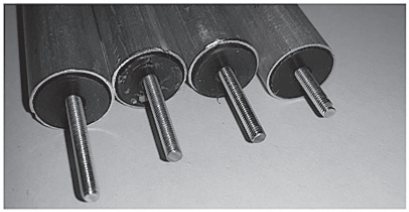
Опорные стойки с запрессованными втулками с резьбой в центре

Средняя плита установлена

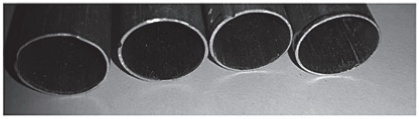
Средняя плита и опорные стойки

Полусобранный корпус

Верхняя плита



Шпилька, вкрученная во втулку

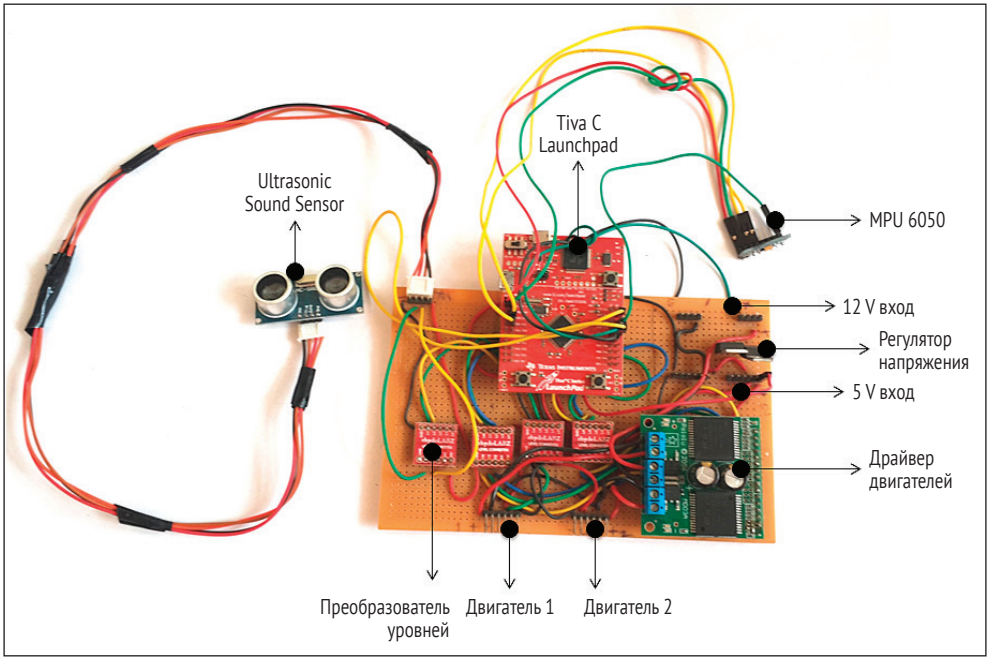


Трубки для стоек

Полностью собранный корпус

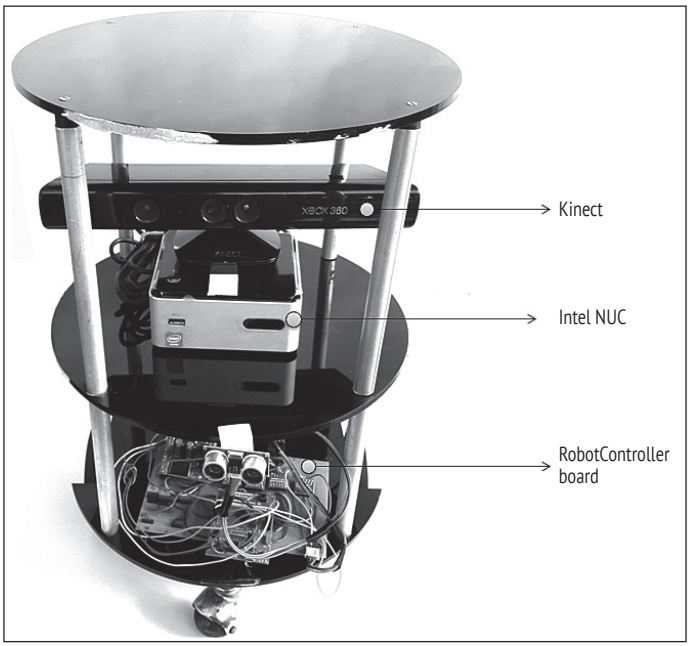
Полностью собранный корпус робота

На основании крепятся печатная плата (PCB) и аккумулятор. На среднюю плиту устанавливается блок датчиков Kinect/Orbecc и Intel NUC. При необходимости можно установить динамик и микрофон. Верхняя плита предназначена для полезной нагрузки. На следующем рисунке показан прототип печатной платы робота. На этой плате установлена плата микроконтроллера Tiva C LaunchPad, драйвера двигателя, линейных переключателей, компонентов для обеспечения управления двумя двигателями, ультразвуковым датчиком и IMU.



Прототип PCB ChefBot

Плата запитывается напряжением в 12 В, которое подается с аккумулятора, расположенного на опорной плите. Двигатели подключаются к разъемам *Двигатель 1* и *Двигатель 2*. Бортовой ПК и датчики Kinect размещены на средней плите. Плата микроконтроллера LaunchPad и датчики Kinect подключаются ПК через USB. ПК и Kinect питаются от того же аккумулятора напряжением 12 В. Можно использовать свинцово-кислотный или литий-полимерный аккумулятор. В конструкции был использован свинцово-кислотный аккумулятор. Литий-полимерный аккумулятор более эффективен, и в дальнейшем им можно заменить свинцово-кислотную батарею. На следующем рисунке показана полностью собранная конструкция робота ChefBot.



Полностью собранный робот

После сборки всех деталей робота мы приступим к разработке программного обеспечения. Встроенный код ChefBot и пакеты ROS доступны в GitHub. Мы можем клонировать код и начать работать с программой.

Конфигурирование бортового компьютера ChefBot и установка пакетов ChefBot ROS

Как уже говорилось ранее, в ChefBot в качестве бортового компьютера был использован компьютер NUC Intel. Его назначение – получение и обработка данных с датчиков и выработка команд управления роботом при движении. После того как будет куплен персональный компьютер NUC Intel, следует установить на него операционную систему Ubuntu 16.04.LTS. После установки Ubuntu установите полный дистрибутив ROS и его пакеты, о которых мы упоминали в предыдущих главах. Компьютер устанавливается на среднюю плиту робота и подключается к его электронным компонентам после предварительной настройки отдельно от робота. Ниже приведены процедуры для установки пакетов ChefBot на ПК NUC.

Скопируйте программные пакеты ChefBot из GitHub, используя следующую команду:

```
$ git clone https://github.com/qboticslabs/Chefbot_ROS_pkg.git
```

Этот код можно клонировать на вашем компьютере, после чего скопировать папку `chefbot` на бортовой ПК Intel NUC. Папка `chefbot` состоит из пакетов ROS ChefBot. Создайте в бортовом ПК NUC рабочее пространство ROS catkin, скопируйте папку `chefbot` и переместите ее в каталог `src` рабочей области catkin.

Создайте и установите исходный код ChefBot, используя нижеприведенную команду. Команда должна быть выполнена в созданном нами рабочем пространстве catkin:

```
$ catkin_make
```

Если все зависимости в бортовом компьютере NUC установлены правильно, то пакеты ChefBot будут собраны и установлены в этой системе. После установки пакетов ChefBot на бортовом ПК NUC мы можем переключиться на встроенный код для ChefBot. После этого подключите все датчики в LaunchPad. После загрузки кода в LaunchPad мы можем снова рассмотреть запуск пакетов ROS. Скопированный код из GitHub содержится в микропроцессоре Tiva C LaunchPad. Этот код мы рассмотрим в следующем разделе.

Согласование датчиков ChefBot с Tiva C LaunchPad

Мы обсудили взаимодействие отдельных датчиков, которые будут использованы в ChefBot. В этом разделе мы обсудим, как подключить и настроить датчики с микроконтроллером Tiva C LaunchPad. Код Energia для программирования Tiva C LaunchPad доступен на клонированных файлах в GitHub. Схема подключения микроконтроллера LaunchPad Tiva C с датчиками показана на следующем рисунке.

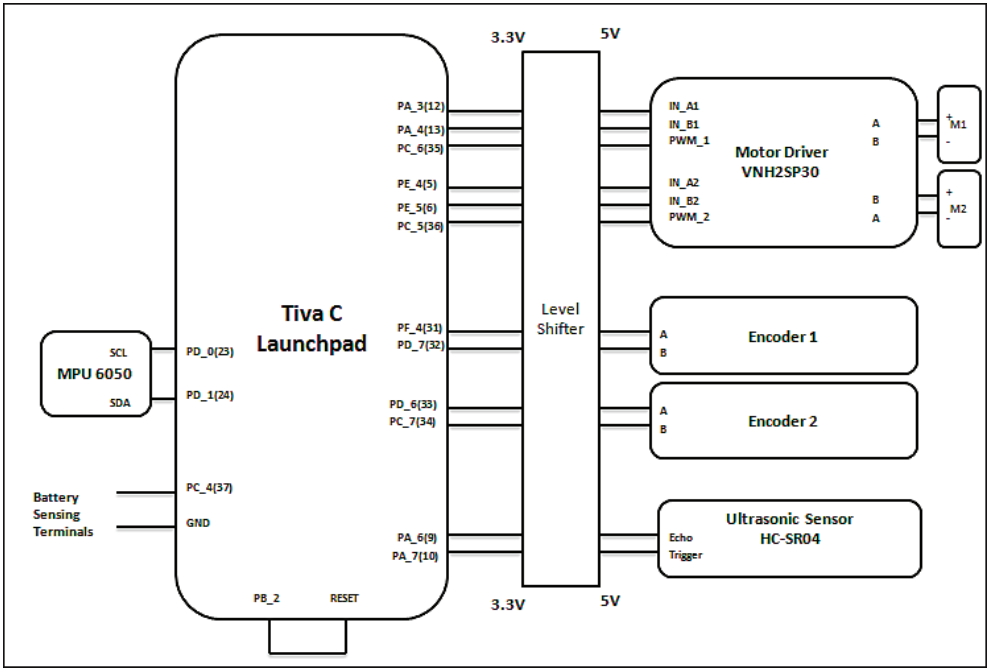


Диаграмма сопряжения датчиков ChefBot

На показанной схеме M1 и M2 обозначают два двигателя дифференциального привода. Это коллекторные двигатели постоянного тока со встроенными энкодерами и редуктором от Pololu. Двигатели подключены к драйверу VNH2SP30 от Pololu. Один из двигателей подключен к схеме в обратной полярности по отношению к другому двигателю, так как в дифференциальном управлении направлением движения один двигатель по отношению к другому вращается в противоположную сторону. Драйвер двигателя подключен к Tiva C LaunchPad через 3,3–5 В двунаправленный преобразователь уровня напряжения. Один из преобразователей, который будет использован, доступен по адресу: https://www.sparkfun.com/products/12009.

Два канала каждого энкодера соединены с LaunchPad через преобразователь уровней. В настоящее время мы используем один ультразвуковой датчик для обнаружения препятствий. В будущем число датчиков при необходимости можно увеличить. Чтобы получить качественные данные одометрии, необходимо подключить датчик MPU 6050 IMU через интерфейс I2C. Данный датчик сразу подключается к LaunchPad, так как питание MPU6050 равно 3,3 В. Для сброса в LaunchPad узлов ROS в первоначальное состояние мы выделим один вывод в качестве выхода и подключимся к контакту *reset* микроконтроллера.

Когда определенный символ отправляется в LaunchPad, он установит на этом контакте значение HIGH и сбросит устройство. В некоторых ситуациях может накапливаться ошибка, что может повлиять на навигацию робота. Чтобы сбросить эту ошибку, требуется выполнить сброс LaunchPad.

Еще один контакт будет выделен для контроля уровня заряда батареи. Пока эта функция в Energia не реализована.

Код, загруженный с GitHub, состоит из встроенного кода. Здесь мы можем рассмотреть основной раздел кода. Объяснять все его разделы нет необходимости, так как это уже было рассмотрено ранее.

Встроенный код для ChefBot

Основные разделы кода LaunchPad будут обсуждаться здесь. Ниже приведены заголовочные файлы, используемые в коде:

```
// Библиотека для связи с устройствами I2C
#include "Wire.h"
//I2C-библиотека связи для MPU6050
#include "I2Cdev.h"
// Библиотека для взаимодействия MPU6050
#include "MPU6050_6Axis_MotionApps20.h"
// Обработка входящих последовательных данных
#include <Messenger.h>
// Содержит определение максимальных пределов различных типов данных
#include <limits.h>
```

Основные библиотеки, используемые в этом коде, предназначены для связи с MPU 6050 и обрабатывают входящие в LaunchPad последовательные данные. MPU 6050 может обеспечить ориентацию в значениях кватерниона или Эйлера с помощью встроенного **Digital Motion Processor** (**DMP**). Функции доступа к DMP записываются в MPU6050_6_MotionApps 20.h. Эта библиотека имеет зависимости, такие как I2Cdev.h и Wire.h. Эти две библиотеки используются для I2C-коммуникации. Библиотека `Messenger.h` позволяет обрабатывать поток текстовых данных из любого источника и помогает извлечь из него данные. Библиотека `limits.h` содержит определения максимальных пределов различных типов данных.

После включения заголовочных файлов необходимо с помощью класса Messenger создать объект для обработки MPU6050 и входящих последовательных данных:

```
// Создание объекта MPU6050
MPU6050 accelgyro(0x68);
// Сообщение объекта
Messenger Messenger_Handler = Messenger();
```

После объявления объекта messenger основной раздел должен назначить контакты для драйвера мотора, энкодера, ультразвукового датчика, MPU 6050, reset и контакты для батареи. После назначения контактов из кода мы переходим к функции `setup()`. Определение функции `setup()` приведено здесь:

```
// Функции по заданию последовательного порта, энкодеров, ультразвука, MPU6050 и Reset
void setup()
{
  // Инициализируем последовательную связь со скоростью передачи  115200
  Serial.begin(115200);
  // Запуск энкодеров
  SetupEncoders();
  // Запуск моторов
  SetupMotors();
  // Запуск ультразвука
  SetupUltrasonic();
  // Запуск  MPU 6050
  Setup_MPU6050();
  // Запуск пина Reset
  SetupReset();
  // Настройка обработчика объектов Messenger
  Messenger_Handler.attach(OnMssageCompleted);
}
```

Предыдущая функция содержит пользовательскую процедуру для настройки и выделения контактов для всех датчиков. Эта функция инициализирует последовательную связь со скоростью передачи данных 115 200 бод и назначает контакты для энкодера, драйвера, ультразвукового датчика и MPU6050. Функция `SetupReset()` назначит pin-код для сброса устройства, как показано в схеме соединений. Мы уже в предыдущих главах рассматривали процедуры настройки каждого датчика. Поэтому повторно объяснять определение каждой функции нет никакой необходимости. Обработчик класса `Messenger` привязан к функции `Onmessagecomplete()`, которая будет вызываться при вводе данных в `Messenger_Handler`.

Ниже, в коде, приводится функция `loop()`. Основная цель этой функции – прочитать и обработать серийные данные и отправить доступные значения датчика:

```
void loop()
{
    // Чтение с порта Serial
    Read_From_Serial();
    // Отправьте данные по времени через последовательный порт
    Update_Time();
    // Отправьте значения энкодера через последовательный порт
    Update_Encoders();
    // Отправьте значения ультразвукового датчика через последовательный порт
    Update_Ultra_Sonic();
    // Обновите значения скорости двигателя с соответствующей скоростью, полученной от ПК,
    // и отправьте значения скорости через последовательный порт
    PC and send speed values through serial port
    Update_Motors();
    // Отправьте значения MPU 6050 через последовательный порт
    Update_MPU6050();
    // Отправьте значения аккумулятора через последовательный порт
    Update_Battery();
}
```

Функция Read_From_Serial() предназначена для считывания последовательных данных с бортового ПК и данных с Messenger_Handler для их обработки. Функция Update_Time() уточняет время после каждого действия на плате управления. Мы можем взять это значение времени для обработки в бортовой ПК или вместо этого значения времени использовать внутреннее время ПК.

Мы можем скомпилировать код в ENERGIA IDE и записать его в LaunchPad. После загрузки кода можно обсудить узлы ROS для обработки значения датчиков в LaunchPad.

Написание драйвера Ros Python для ChefBot

После загрузки встроенного кода в LaunchPad следующий шаг – это обработка получаемых из LaunchPad последовательных данных и их конвертирование для дальнейшей обработки в темы ROS. Драйвер узла Launchpad_node.py взаимодействует с Tiva C LaunchPad через ROS. Файл launchpad_node.py находится внутри пакета ChefBot_bringup, в папке script. Далее последует объяснение важных разделов кода launchpad_node.py:

```
# Клиент ROS Python
import rospy
import sys
import time
import math

# Этот модуль python помогает получать значения из последовательного порта, которые
# выполняются в потоке
from SerialDataGateway import SerialDataGateway
# Импорт необходимых типов данных ROS для кода
from std_msgs.msg import Int16,Int32, Int64, Float32,String, Header, UInt64
# Импорт данных ROS для IMU
from sensor_msgs.msg import Imu
```

Файл Launchpad_node.py импортирует предыдущие модули. Основной модуль – SerialDataGateway. Это пользовательский модуль, написанный для получения последовательных потоковых данных с LaunchPad. Нам также понадобятся некоторые типы данных ROS, чтобы отрегулировать данные датчиков. Основная функция узла задается в следующем фрагменте кода:

```
if __name__ =='__main__':
    rospy.init_node('launchpad_ros',anonymous=True)
    launchpad = Launchpad_Class()
    try:

        launchpad.Start()
        rospy.spin()
    except rospy.ROSInterruptException:
        rospy.logwarn("Error in main function")

    launchpad.Reset_Launchpad()
    launchpad.Stop()
```

Основной класс этого узла называется `Launchpad_Class()`. Этот класс содержит все методы запуска, остановки и преобразования последовательных данных в разделы ROS. В главной функции мы создадим объект `Launchpad_Class()`. После создания объекта можно будет вызывать метод `Start()`, который запустит последовательную связь между Tiva C LaunchPad и бортовым ПК. Если мы с помощью комбинации клавиш **Ctrl**+**C** прервем узел драйвера, он будет сброшен в LaunchPad и остановит последовательную связь между ПК и LaunchPad.

Следующий фрагмент кода содержит функции конструктора `Launchpad_Class()`. Здесь мы подключаемся к порту и настраиваем скорость передачи данных LaunchPad из параметров ROS и с помощью тех же параметров инициализируем `SerialDateGateway`. Объект `_HandleReceivedLine()` из `SerialDataGateway` – функция внутри этого класса, которая запускается при условии, когда любые входящие последовательные данные поступают на последовательный порт.

Эта функция будет обрабатывать каждую строку последовательных данных: извлекать, преобразовывать и вставлять ее в соответствующие заголовки каждого типа данных темы ROS:

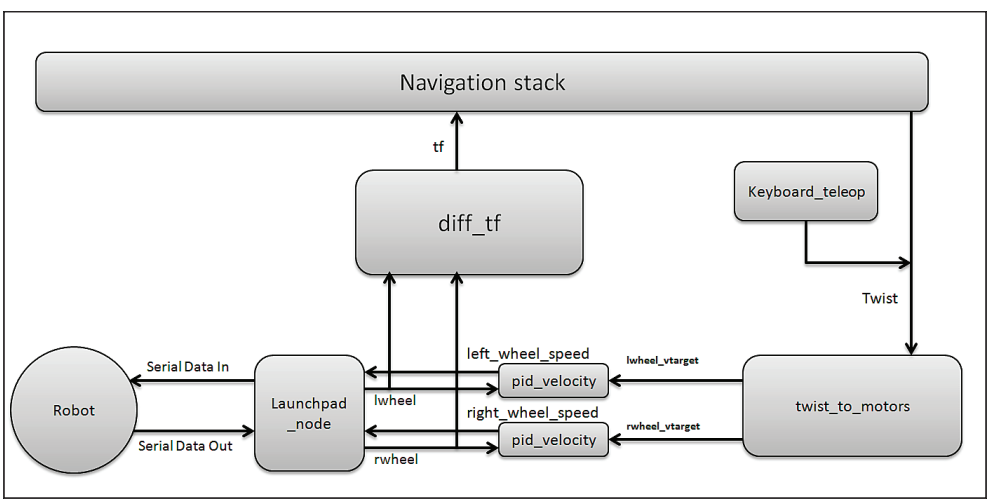
```
# Получаем  последовательный порт и скорость передачи данных Tiva C Launchpad
port = rospy.get_param("~port", "/dev/ttyACM0")
baudRate = int(rospy.get_param("~baudRate", 115200))

#################################################################
rospy.loginfo("Starting with serial port: " + port + ", baud rate: " + str(baudRate))
# Инициализация объекта SerialDataGateway с последовательным портом, скоростью передачи
# и функцией обратного вызова для обработки
self._SerialDataGateway = SerialDataGateway(port,baudRate, self._HandleReceivedLine)
rospy.loginfo("Started serial communication")

#################################################################
##Подписки и публикации

Publisher for left and right wheel encoder values
self._Left_Encoder = rospy.Publisher('lwheel',Int64,queue_size = 10)
self._Right_Encoder = rospy.Publisher('rwheel',Int64,queue_size = 10)

# Публикация уровня аккумулятора
self._Battery_Level = rospy.Publisher('battery_level',Float32,queue_size = 10)

# Публикация расстояния до ультразвукового датчика
self._Ultrasonic_Value = rospy.Publisher('ultrasonic_distance',Float32,queue_size = 10)

# Публикация значений кватерниона вращения от IMU
self._qx_ = rospy.Publisher('qx',Float32,queue_size = 10)
self._qy_ = rospy.Publisher('qy',Float32,queue_size = 10)
self._qz_ = rospy.Publisher('qz',Float32,queue_size = 10)
self._qw_ = rospy.Publisher('qw',Float32,queue_size = 10)

# Публикация последовательных  данных целиком
self._SerialPublisher = rospy.Publisher('serial', String,queue_size=10)
```

Мы создадим объект ROS publisher для таких датчиков, как энкодер, IMU и ультразвуковой датчик, а также совокупность последовательных данных для отладки. Кроме этого, подпишемся на скоростные команды для управления левым и правым двигателями робота. Когда скоростная команда поступает в тему, она создает соответствующие обратные вызовы для отправки команд скорости на микроконтроллер LaunchPad:

```
self._left_motor_speed = rospy.Subscriber('left_wheel_
speed',Float32,self._Update_Left_Speed)
self._right_motor_speed = rospy.Subscriber('right_wheel_
speed',Float32,self._Update_Right_Speed)
```

После установки узла драйвера ChefBot нам нужно подключить робота к навигационному стеку ROS для выполнения автономной навигации. Базовое требование для выполнения автономной навигации состоит в том, чтобы узлы драйвера робота получали скоростные команды из навигационного стека ROS. Робот может управляться с помощью дистанционного управления. В дополнение к этим требованиям робот должен вычислять свое местоположение и одометрические данные и генерировать данные для отправки в навигационный стек. Чтобы контролировать скорость двигателя робота, нам потребуется регулятор **пропорционального интегрального дифференциального регулирования** (**PID**). Эти функции помогает выполнять пакет ROS под названием `differential_drive`, содержащий все необходимые узлы. Мы повторно используем эти узлы в нашем пакете для реализации данных функций. Ниже приведена ссылка для пакета `differential_drive` в ROS: http://wiki.ros.org/differential_drive.

На следующем рисунке показано взаимодействие этих узлов друг с другом. Можно также обсудить использование других узлов.



Структурная схема робота, показывающая узлы ROS

Назначение каждого узла в пакете chefbot_bringup выглядит следующим образом.

twist_to_motors.py: этот узел преобразует команду Ros Twist или линейную и угловую скорости к угловой скорости колеса. Конечная скорость публикуется со скоростью ~rate (измеряется в герцах), и скорость публикации timeout_ticks времени после остановки твист-сообщения. Ниже приведены разделы и параметры, которые будут опубликованы и подписаны этим узлом.

Темы публикаций:

- wheel_target (std_msgs/Float32) – это скорость левого колеса (м/с);
- wheel_target (std_msgs/Float32) – это скорость правого колеса (м/с).

Темы подписки:

Twist (geometry_msgs/Twist) – это команда для поворота робота. Линейная скорость в направлении x и угловая скорость θ использована в сообщении Twist этого робота.

Важные параметры ROS:

- ~base_width (float, по умолчанию: 0.1): это расстояние между двумя колесами робота в метрах;
- ~rate (int, по умолчанию: 50): это скорость, с которой опубликована конечная скорость (Гц);
- ~timeout_ticks (int, по умолчанию: 2): это число сообщений о конечной скорости, опубликованной после сообщения Twist об остановке;
- pid_velocity.py: это простой регулятор PID, предназначенный для контроля скорости каждого двигателя с помощью обратной связи от шифраторов колеса. В системе дифференциального привода нам нужен один регулятор PID для каждого колеса. Он считывает данные шифратора каждого колеса и контролирует скорость каждого колеса.

Темы публикаций:

- motor_cmd (Float32) – это окончательный результат PID-регулятора, который направляется к двигателю. Мы можем изменить диапазон выходных данных PID с использованием параметров out_min и out_max ROS;
- wheel_vel (Float32) – это текущая скорость колеса робота (м/с).

Темы подписки:

- wheel (Int16) – это тема углового энкодера. Есть отдельные темы для каждого энкодера робота;
- wheel_target (Float32) – конечная скорость (м/с).

Важные параметры:

- ~Kp (float, по умолчанию: 10) – этот параметр пропорционален коэффициенту PID-регулятора;
- ~Ki (float, по умолчанию: 10): этот параметр является интегралом коэффициента усиления PID-регулятора;
- ~Kd (float, по умолчанию: 0.001): это производная коэффициента PID-регулятора;

- ~out_min (float, по умолчанию: 255): это минимальный предел значения скорости двигателя. Этот параметр ограничивает значение скорости двигателя и называется темой wheel_vel;
- ~out_max (float, по умолчанию: 255): это максимальный предел темы wheel_vel (Гц);
- ~rate (float, по умолчанию: 20): это скорость публикации темы wheel_vel;
- ticks_meter (float, по умолчанию: 20): это число импульсов энкодера колеса на метр. Это глобальный параметр, который используется и в других узлах;
- vel_threshold (float, по умолчанию: 0.001): если скорость робота падает ниже этого параметра, мы рассматриваем колесо как неподвижное. Если скорость колеса меньше vel_threshold, то будем считать ее нулевой;
- encoder_min (int, по умолчанию: 32768): это минимальное значение чтения энкодера;
- encoder_max (int, по умолчанию: 32768): это максимальное значение чтения энкодера;
- wheel_low_wrap (int, по умолчанию: 0.3 * (encoder_max - encoder_min) + encoder_min): эти значения определяют, находится ли одометрия в отрицательном или положительном направлении;
- wheel_high_wrap (int, по умолчанию: 0.7 * (encoder_max - encoder_min) + encoder_min): эти значения решают, находится ли одометрия в отрицательном или положительном направлении;
- diff_tf.py: этот узел вычисляет преобразование одометрии и передачу между блоком данных одометрии и базовым блоком данных робота.

Темы публикации:

- odom (nav_msgs/odometry) – публикует одометрию (текущая поза и поворот) робота;
- Tf – обеспечивает преобразование между блоком данных одометрии и базовой связью робота.

Темы подписки:

- lwheel (std_msgs/Int16), rwheel (std_msgs/Int16) – значения выхода от левого и правого энкодеров робота;
- chefbot_keyboard_teleop.py – этот узел передает команду Twist с клавиатуры.

Темы публикации:

- cmd_vel_mux/input/teleop (geometry_msgs/Twist) – сообщения Twist публикуются с помощью команд клавиатуры.

После обсуждения узлов пакета chefbot_bringup мы рассмотрим функции запуска файлов.

Функции исполняемого файла ChefBot ROS

Теперь мы рассмотрим функции каждого из файлов запуска пакета ChefBot_bringup:

- `robot_standalone.launch`. Основная функция этого исполняемого файла – запускать такие узлы, как `launchpad_node`, `pid_velocity`, `diff_tf` и `twist_to_motor`, чтобы получить значения датчика от робота и послать команду на управление скоростью робота;
- `keyboard_teleop.launch`: запуск дистанционного управления с клавиатуры. Запускает узел `chefbot_keyboard_teleop.py` для дистанционного управления с клавиатуры;
- `3dsensor.launch`: этот файл запускает драйверы Kinect с поддержкой OpenNI и начинает публикацию потока RGB и глубины. Он также запустит узел лазерного сканирования глубины, который преобразует данные облака точек в данные лазерного сканирования;
- `gmapping_demo.launch`: данный файл запуска запустит узлы Slam gmapping для составления карты местности, окружающей робота;
- `amcl_demo.launch`: используя AMCL, робот может определить свое место на карте. После локализации робота на карте мы можем им управлять для перехода в требуемую позицию на карте. Затем робот может самостоятельно перемещаться из текущего положения в положение цели;
- `view_robot.launch`: этот файл запуска отображает модель робота URDF в RViz;
- `view_navigation.launch`: этот файл запуска отображает все датчики, необходимые для навигации робота.

Элементы Python ChefBot и запуск файлов

Мы уже установили пакеты CHEFBOT ROS в ПК Nuc Intel и загрузили встроенный код в LaunchPad. Следующим шагом мы установим бортовой ПК NUC на среднюю плиту робота, настроим удаленное подключение ноутбука к бортовому компьютеру робота, протестируем каждый узел и начнем работать с файлами запуска ChefBot, предназначенными для автономной навигации.

Следует заметить, что для работы с ChefBot потребуется беспроводной маршрутизатор. Бортовой компьютер робота и удаленный ноутбук или настольный компьютер необходимо подключить к одной сети. Если бортовой ПК робота и удаленный ноутбук или настольный компьютер находится в одной сети, удаленный ноутбук или настольный компьютер может подключиться к бортовому ПК через SSH, используя свой IP. Перед тем как установить бортовой ПК, необходимо подключить бортовой ПК к беспроводной сети. После первого подключения к сети бортовой ПК запомнит параметры подключения. При включении робота бортовой компьютер автоматически подключится к беспроводной сети. Ниже показана схема подключения робота и удаленного ПК.

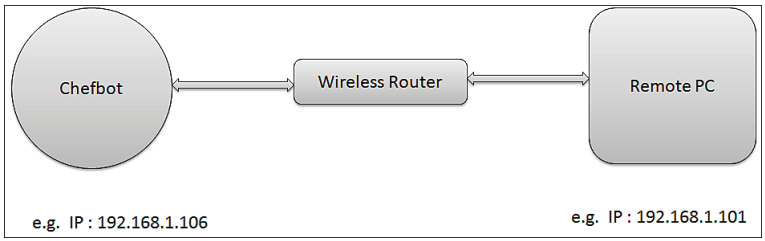


Схема беспроводного подключения робота и удаленного ПК

Предыдущий рисунок предполагает, что IP ChefBot – `192.168.1.106`, а IP удаленного ПК – `192.168.1.101`.

Мы с помощью SSH можем удаленно открыть терминал ChefBot. Для входа в ChefBot используется следующая команда, где имя робота – ChefBot PC:

```
$ ssh robot@192.168.1.106
```

При подключении CHEFBOT PC будет запрашиваться пароль бортового ПК. После ввода пароля появится доступ к терминалу бортового ПК робота. Когда вход был выполнен, можно начать испытание узлов ROS ChefBot и проверить, сможем ли мы получать последовательные данные от платы управления LaunchPad ChefBot. Заметьте, если будет запущен новый терминал, придется снова войти в CHEFBOT PC через SSH.

Если пакет `Chefbot_bringup` правильно установлен на бортовом ПК и если микроконтроллер LaunchPad подключен, тогда перед запуском узла драйвера ROS мы можем запустить инструмент `miniterm.py`, чтобы проверить, приходят ли последовательные данные к бортовому ПК через USB. Имя последовательного устройства мы можем найти с помощью команды `dmesg`.

`miniterm.py` запускается с помощью следующей команды:

```
$ miniterm.py /dev/ttyACM0 115200
```

Если появится сообщение об отказе доступа, установите доступ к USB-устройству, напишите правила для папки `udev`, как мы делали в главе 5. Мы предполагаем, что `ttyACM0` – это имя запускаемого устройства. Если на вашем компьютере устройство названо по-другому, вместо `ttyACM0` используйте фактическое имя устройства:

```
$ sudo chmod 777 /dev/ttyACM0
```

Если все работает нормально, мы получим значения, как показано на следующем рисунке.

```
b       0.00
t       66458239        0.05
e       0       0
u       10
s       0.00    0.00
i       -0.68   -0.47   -0.40   0.40
b       0.00
t       66511681        0.05
e       0       0
u       10
s       0.00    0.00
i       -0.68   -0.47   -0.40   0.40
b       0.00
t       66566051        0.05
e       0       0
u       10
s       0.00    0.00
i       -0.68   -0.47   -0.40   0.40
b       0.00
t       66620423        0.05
e       0       0
u       10
s       0.00    0.00
```

Выход `miniterm.py`

Буква `b` используется для обозначения аккумулятора робота; в настоящее время эта функция не реализована и значение равно 0. Для измерения напряжения существуют различные методы, реализованные с помощью микроконтроллера. Один из подходов описывается здесь: http://www.instructables.com/id/Arduino-Battery-Voltage-Indicator/. Буква `t` указывает общее время (в микросекундах) после запуска роботом встроенного кода. Второе значение – это время, затраченное на выполнение одной операции в LaunchPad (измеряется в секундах). Мы можем использовать это значение, если выполняем вычисления параметров робота в реальном времени. На данный момент мы это значение не используем. Но можем использовать его в будущем. Буква `e` обозначает значения левого и правого энкодеров соответственно. Робот не движется, и эти значения равны нулю. Буква `u` указывает значения от ультразвукового датчика расстояния. Значение расстояния измеряем в сантиметрах. Буква `s` указывает текущую скорость вращения колеса робота. Это значение используется для проверки. На самом деле скорость – это контроль от самого PC.

Чтобы преобразовать эти последовательные данные в темы ROS, необходимо управлять узлом двигателя, который называется `launchpad_node.py`. В следующем коде показано, как выполнить этот узел.

В первую очередь перед запуском всех узлов нам следует запустить `roscore`:

```
$ roscore
```

Далее выполните launchpad_node.py с помощью следующей команды:

```
$ rosrun chefbot_bringup launchpad_node.py
```

Если все работает нормально, мы получим следующий вывод в узле работающего терминала:

```
robot@robot-desktop:~$ rosrun chefbot_bringup launchpad_node.py
Initializing Launchpad Class
[INFO] [WallTime: 1424097603.219564] Starting with serial port: /dev/ttyACM0, bau
d rate: 115200
[INFO] [WallTime: 1424097603.220825] Started serial communication
```

Выход launchpad_node.py

После запуска launchpad_node.py мы увидим следующие созданные темы, как показано на рисунке ниже.

```
robot@robot-desktop:~$ rostopic list
/battery_level
/imu/data
/left_wheel_speed
/lwheel
/qw
/qx
/qy
/qz
/right_wheel_speed
/rosout
/rosout_agg
/rwheel
/serial
/ultrasonic distance
```

Темы, созданные launchpad_node.py

Мы можем просмотреть последовательные данные, полученные узлом драйвера, подписавшись на /serial, и использовать эти данные для отладки. Если /serial показывает те же данные, что и miniterm.py, значит, узлы работают нормально. Следующий скриншот – это вывод /serial:

```
---
data: 16266, in:  e     1       -1
---
data: 16267, in:  u     10
---
data: 16268, in:  s     0.00    0.00
---
```

Вывод /serial topic, опубликованный узлом LaunchPad

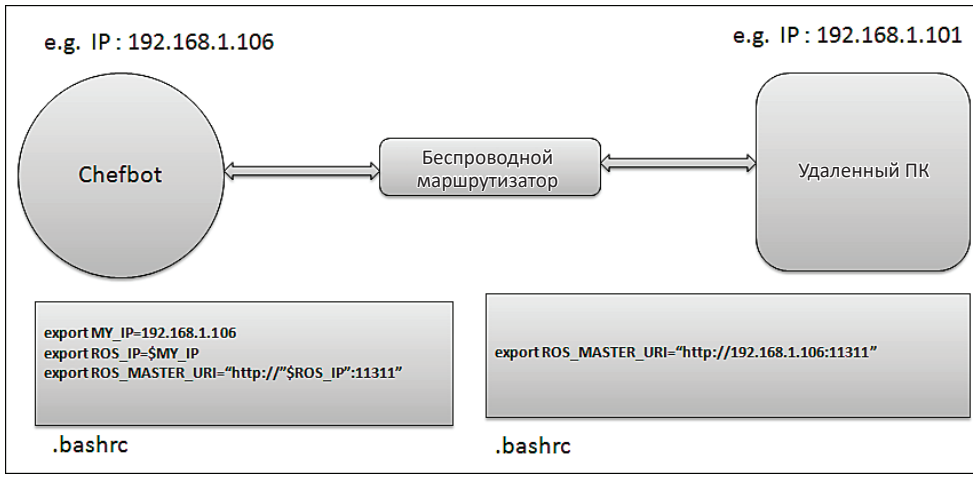
После установки пакета chefbot_bringup мы можем начать работу с автономной навигацией ChefBot. В настоящее время мы работаем только с терминалом бортового ПК ChefBot. Чтобы отобразить на экране модель робота, данные датчиков, карты и т. д., мы используем RViz, установленный в уда-

ленном ПК (пользовательский компьютер). Следует отметить, что компьютер пользователя должен иметь ту же настройку программного обеспечения, что и компьютер ChefBot.

Первое, что нам нужно сделать, – это установить ПК ChefBot в качестве мастера ROS. Для этого необходимо установить значение `ROS_MASTER_URI`. `ROS_MASTER_URI` является обязательным параметром, он информирует узлы о **едином идентификаторе ресурса** (**URI**) ROS-мастера. Когда `ROS_MASTER_URI` будут установлены на бортовом ПК ChefBot и на удаленном компьютере, можно открыть темы ChefBot ПК в удаленном компьютере. Таким образом, если мы запустим локально RViz, то он будет визуализировать темы, создаваемые бортовым ПК CHEFBOT.

Предположим, у бортового ПК ChefBot PC IP-адрес `192.168.1.106`, а IP-адрес удаленного компьютера – `192.168.1.101`.

Для установки `ROS_MASTER_URI` в каждой системе следующая команда должна быть включена в файл `.bashrc` в домашнем каталоге. На следующей схеме показана настройка, необходимая для включения `.bashrc` в бортовом ПК и удаленном ПК.



Конфигурация сети для ChefBot

Вместо указанных на рисунке IP-адресов внесите в строки ваши реальные IP-адреса и добавьте эти строки внизу `.bashrc` на каждом ПК.

После выполнения этих настроек запустите `roscore` в терминале бортового ПК ChefBot ПК и выполните команду `rostopic list` на удаленном компьютере.

Если появятся какие-то темы, значит, настройки выполнены. Используя дистанционное управление с клавиатуры, сначала запустим робот. Это делается, чтобы убедиться в работоспособности робота и получить значения с датчиков.

Используя приведенную ниже команду, запустите драйвер робота и другие узлы. Обратите внимание, команда должна выполняться в терминале бортового ПК ChefBot. Для входа в этот терминал используйте SSH:

```
$ roslaunch chefbot_bringup robot_standalone.launch
```

После запуска драйвера робота и узлов, используя следующую команду, запустите удаленную клавиатуру. Эта клавиатура должна быть запущена и в новом терминале бортового ПК ChefBot:

```
$ roslaunch chefbot_bringup keyboard_teleop.launch
```

Для активации Kinect выполните команду, приведенную ниже. Эта команда выполняется в терминале бортового ПК ChefBot:

```
$roslaunch chefbot_bringup 3dsensor_kinect.launch
```

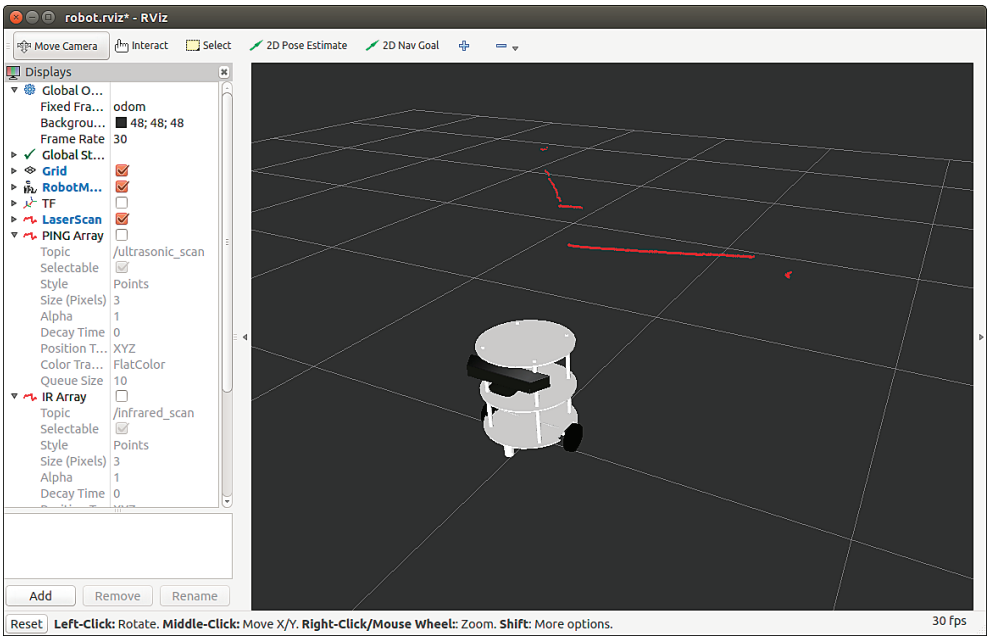
Если вы используете блок датчиков Orbecc Astra, для его запуска используйте следующую команду:

```
$ roslaunch ChefBot_bringup 3d_sensor_astra.launch
```

Для просмотра данных, поступающих с Kinect или Astra, выполните следующую команду. После выполнения этой команды на экране удаленного компьютера, в RViz, появится изображение робота. Если мы на удаленном ПК настроим пакет `ChefBot_bringup`, то сможем получить доступ к следующей команде и визуализировать модель робота и данные датчиков с бортового ПК ChefBot:

```
$ roslaunch chefbot_bringup view_robot.launch
```

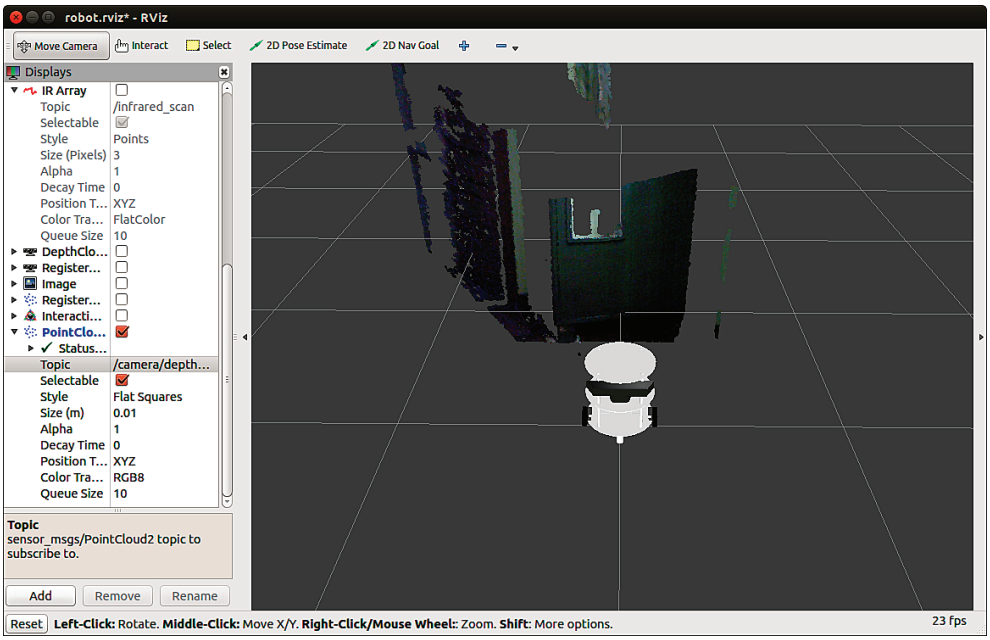
На расположенном ниже рисунке показана работа робота в RViz. Здесь приведены данные лазерного сканирования и карта облака точек.



Данные лазерного сканирования ChefBot в RViz

На предыдущем рисунке показаны данные лазерного сканирования, отображенные в RViz. Для отображения этих данных установите в левой части окна **RViz** флажок напротив темы **LaserScan**. Данные лазерного сканирования показаны на экране. Для просмотра карты облака точек Kinnect в левом нижнем углу окна **RViz** нажмите кнопку **Add**. Появится всплывающее окно. Выберите в этом окне строку **PointCloud2.**

Выберите в списке **Тема** `/camera/depth_registered`. В окне RViz появятся данные облака точек, как показано на следующем рисунке.



Данные облака точек ChefBot

Датчики работают. Теперь можно для отображения карты помещения запустить SLAM. Следующая процедура поможет нам запустить SLAM на этом роботе.

Построение карты комнаты с помощью SLAM в ROS

Чтобы запустить gmapping, мы должны выполнить следующие команды.

Следующая команда запускает драйвер робота в терминале ChefBot:

```
$ roslaunch chefbot_bringup robot_standalone.launch
```

Для запуска gmapping выполните нижеприведенную команду. Обратите внимание, команда должна выполняться в терминале бортового ПК ChefBot:

```
$ roslaunch chefbot_bringup gmapping_demo.launch
```

Gmapping будет работать только в случае, если данные одометрии будут получены без ошибок. Если значение одометра получено, появится следующее сообщение для предыдущей команды. Если это сообщение поступит, значит, gmapping работает нормально.

```
[ INFO] [1422618733.585407153]: Created local_planner dwa_local_planner/DWAPlanner
ROS
[ INFO] [1422618733.604762090]: Sim period is set to 0.20
[ INFO] [1422618735.208493249]: odom received!
```

Данные облака точек Chefbot

Для запуска дистанционной клавиатуры используйте следующую команду:

```
$ roslaunch chefbot_bringup keyboard_teleop.launch
```

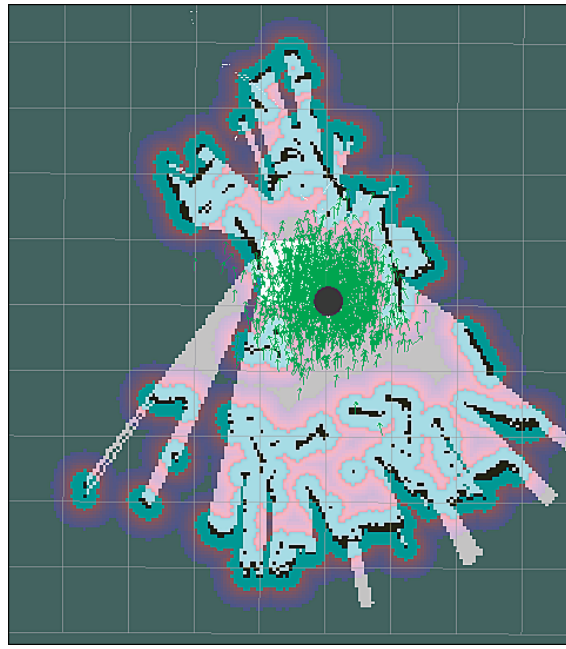
С помощью показанной ниже команды запустите на удаленном ПК RViz:

```
$ roslaunch chefbot_bringup view_navigation.launch
```

После того как в RViz появится изображение робота, перемещайте робота с помощью клавиатуры и посмотрите создаваемую карту. Когда вся область будет отображена, сохраните карту, введя в терминал бортового ПК ChefBot следующую команду:

```
$rosrun map_server map_saver -f ~/test_map
```

где `test_map` – имя карты, хранящейся в домашнем каталоге. На следующем рисунке показана созданная роботом карта комнаты.



Карта помещения

Итак, карта сохранена. Далее можно с помощью ROS начать работу с автономной навигацией и определением положения робота на карте.

ROS: локализация и навигация

После построения карты, используя следующую команду, закройте все приложения и перезапустите драйвер робота:

```
$ roslaunch chefbot_bringup robot_standalone.launch
```

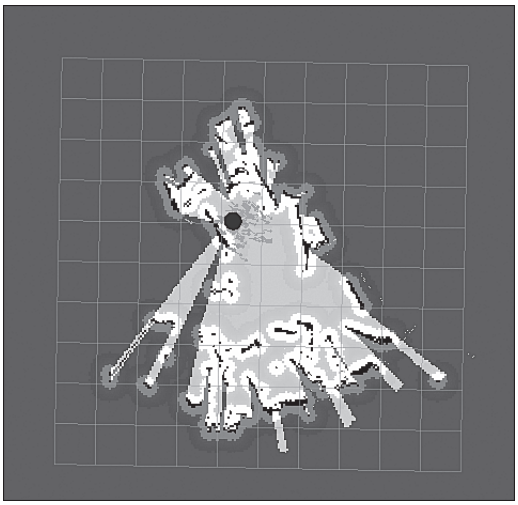
Запустите с помощью следующей команды локализацию и навигацию по сохраненной карте:

```
$ roslaunch chefbot_bringup amcl_demo.launch map_file:=~/test_map.yaml
```

Начните с помощью команды, показанной ниже, просмотр работы робота на удаленном компьютере:

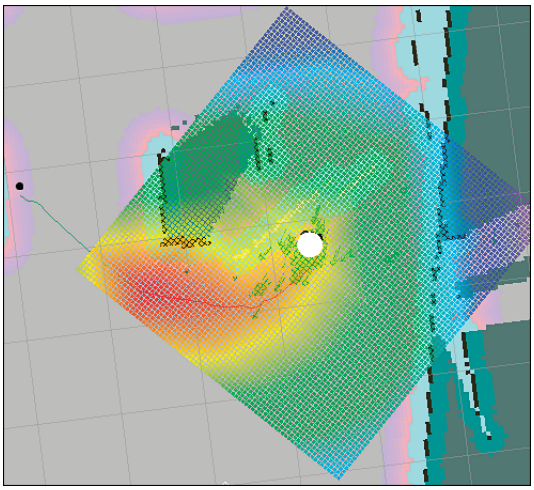
```
$ roslaunch chefbot_bringup view_navigation.launch
```

В **RViz** нам может потребоваться указать начальное положение робота. Для этого нажмите кнопку **2D Pose Estimate** и укажите на карте положение робота на данный момент. Если у робота есть доступ к карте, кнопка **2D Nav Goal** позволит переместить робот в нужное положение. В начале локализации, согласно алгоритму AMCL, вы увидите облако частиц вокруг робота.



Локализация робота с помощью AMCL

Ниже приведен снимок экрана робота, когда он самостоятельно переходит из своего текущего положения в целевую позицию. Целевая позиция отмечена черной точкой.



Автономная навигация с помощью карты

Цветная линия от робота до черной точки – это планируемый путь робота в целевую позицию. Если карта роботу не доступна, потребуется точная настройка файлов параметров в папке `chefbot_bringupparam`. Для более точной настройки зайдите в ROS-пакет AMCL по адресу: http://wiki.ros.org/amcl?distro=indigo.

Итоги

Эта глава была посвящена сборке аппаратного обеспечения ChefBot и интеграции встроенного ROS-кода, с помощью которого выполняется автономная навигация. Мы рассмотрели отдельные части робота, которые были изготовлены с использованием конструкций, рассмотренных в главе 6 «Согласование приводов и датчиков с контроллером робота». Был собран отдельный узел робота и подключен разработанный для него прототип PCB. Этот узел состоит из микроконтроллера LaunchPad, драйвера двигателя, левого переключателя, ультразвукового датчика расстояния и IMU (гироскопа-акселерометра). Далее заменили встроенный код микропроцессора LaunchPad, благодаря чему микроконтроллер получил возможность взаимодействовать со всеми датчиками робота и обмениваться данными с бортовым ПК. После обсуждения встроенного кода мы написали узел драйвера Ros Python для сопряжения последовательных данных с LaunchPad. После согласования микроконтроллера LaunchPpad мы рассчитали данные одометрии и управления дифференциальным приводом. Для этого мы использовали узлы из пакета `differential_drive`, из репозитория ROS. Далее мы подключили робот к навигационному стеку ROS, что позволило выполнить SLAM и AMCL для автономной навигации. Мы также обсудили SLAM, AMCL, создание карты и выполнение автономной навигации робота.

Вопросы

1. Для чего в роботе используется узел драйвера ROS?
2. Какова роль контроллера PID в навигации?
3. Как преобразовать данные энкодера в данные одометрии?
4. Какова роль SLAM в навигации роботов?
5. Какова роль AMCL в навигации роботов?

Дополнительная информация

Чтобы узнать больше об автоматизированном пакете технического зрения в ROS, перейдите по следующим ссылкам:

- http://wiki.ros.org/gmapping;
- http://wiki.ros.org/amcl.

Глава 9

Разработка графического интерфейса для робота с использованием Qt и Python

В последней главе мы обсудили аппаратную сборку, интеграцию компонентов роботизированного оборудования и программные комплексы для выполнения задач автономной навигации. Далее необходимо создать графический интерфейс для управления роботом. Мы создаем графический интерфейс, который может действовать как триггер для базовых команд ROS. Вместо выполнения всех команд в терминале пользователь может управлять роботом с помощью графических кнопок. Графический интерфейс, который мы собираемся разработать, предназначен для типичного гостиничного номера с девятью столами. Пользователь может установить положение стола на карте гостиничного номера и командовать роботом для доставки еды к определенному столу. После того как еда будет доставлена, робот получит команду для возвращения в исходное положение.

В этой главе будут рассмотрены следующие темы:

- установка Qt на Ubuntu;
- введение в PyQt и PySide;
- введение в Qt Designer;
- сигналы и слоты Qt;
- преобразование файла Qt UI в файл Python;
- работа с ChefBot из графического интерфейса;
- введение в rqt и его особенности.

Технические требования

Для тестирования приложения и кода вам понадобится Ubuntu 16.04 LTS PC/laptop с установленным ROS Kinetic.

Вы должны знать, как установить Qt, PyQt и rqt.

В настоящее время наиболее популярны два GUI-фреймворка: Qt (http://qt.digia.com) и GTK+ (http://www.gtk.org/). QT и GTK+ имеют открытые исходные коды, кроссплатформенные наборы инструментов пользовательского интерфейса и платформы разработки. Эти две программные платформы широко используются в настольных средах Linux, в оболочках GNOME и KDE.

В этой главе для реализации GUI мы будем использовать привязку Python к платформе Qt. Это объясняется тем, что привязка Python Qt более стабильна, чем другие привязки UI Python. Мы рассмотрим, как разработать графический интерфейс с нуля и запрограммировать его с помощью Python. После обсуждения основы программирования на Python и Qt мы обсудим интерфейсы ROS Qt, которые уже доступны в ROS. Но вначале следует познакомиться с QT UI framework и как его установить на ПК.

Установка Qt на Ubuntu 16.04 LTS

Qt – это кроссплатформенный фреймворк приложений, широко использующийся для разработки прикладного обеспечения с интерфейсом GUI и инструментов командной строки. Qt доступен почти во всех операционных системах, таких как Windows, macOS X, Android и т. д. Основной язык программирования, используемый для разработки приложений QT, – C++. Но есть привязки, доступные для таких языков, как Python, Ruby, Java и т. д. Давайте посмотрим, как установить QT SDK на Ubuntu 16.04. Мы установим Qt со **средством предварительной упаковки** (**APT**) в Ubuntu. APT входит в репозиторий Ubuntu. Итак, для установки Qt/Qt SDK мы можем просто использовать следующую команду, которая установит QT SDK и все необходимые зависимости из репозиториев Ubuntu. Мы, используя следующую команду, установим Qt версии 4:

```
$ sudo apt-get install qt-sdk
```

Эта команда установит весь QT SDK и его библиотеки, необходимые для нашего проекта. Пакеты, доступные в репозиториях Ubuntu, могут быть не самыми последними версиями. Получить самую последнюю версию Qt можно, загрузив онлайн- или автономный установщик Qt для различных платформ ОС по следующей ссылке: http://qt-project.org/downloads.

После установки Qt в вашей системе мы рассмотрим, как с помощью Qt и Python можно разработать графический интерфейс.

Взаимодействие Python и Qt

Давайте посмотрим, как мы можем взаимодействовать с Python и Qt. В общем, есть два модуля, доступных в Python для подключения к пользовательскому интерфейсу Qt. Это два наиболее популярных фреймворка:

- PyQt;
- PySide.

PyQt

PyQt является одним из самых популярных привязок Python для кроссплатформенной Qt. PyQt разработан и поддерживается Riverbank Computing Limited. Обеспечивает привязку для Qt 4 и Qt 5 и поставляется с GPL (версия 2 или 3) вместе с коммерческой лицензией. PyQt доступен для Qt версий 4 и 5 и называется PyQt4 или PyQt5 соответственно. Эти два модуля совместимы с Python версий 2 и 3. PyQt содержит более 620 классов, охватывающих пользовательский интерфейс, XML, сеть сообщений, веб и т. д.

PyQt доступен в Windows, Linux и Mac OS X. Это обязательное условие для установки Qt SDK и Python и установки PyQt. Бинарники для Windows и Mac OS доступны по следующей ссылке: http://www.riverbankcomputing.com/software/pyqt/download.

Теперь можно рассмотреть, как установить PyQt4 на Ubuntu 16.04 с помощью Python 2.7.

Установка PyQt в Ubuntu 16.04 LTS

Для установки PyQt на Ubuntu/Linux используется следующая команда, которая установит библиотеку PyQt, ее зависимости и инструменты Qt:

```
$ sudo apt-get install python-qt4 pyqt4-dev-tools
```

PySide

PySide – это проект программного обеспечения с открытым исходным кодом, который предоставляет привязку Python для фреймворка Qt. Проект PySide был инициирован Nokia и предлагает полный набор Qt с привязкой к нескольким платформам. Техника, используемая в PySide для переноса библиотеки Qt, отличается от PyQt, но API обоих похож. PySide в настоящее время не поддерживается на Qt 5. PySide доступен для Windows, Linux и Mac OS X. Следующая ссылка поможет вам настроить PySide на Windows и Mac OS X: http://qt-project.org/wiki/Category:LanguageBindings::PySide::Downloads.

Предварительные настройки для PySide такие же, как и для PyQt. Давайте посмотрим, как мы можем установить PySide на Ubuntu 16.04 LTS.

Установка PySide в Ubuntu 16.04 LTS

Пакет PySide доступен в репозиториях пакетов Ubuntu. Следующая команда установит модуль PySide и инструменты Qt на Ubuntu:

```
$ sudo apt-get install python-pyside pyside-tools
```

Давайте начнем работу с этими модулями, чтобы увидеть различия между ними.

Установка PyQt в Ubuntu 16.04 LTS

Если вы хотите установить PyQt в Ubuntu/Linux, используйте следующую команду. Эта команда установит библиотеку PyQt, ее зависимости и некоторые инструменты Qt:

```
$ sudo apt-get install python-qt4 pyqt4-dev-tools
```

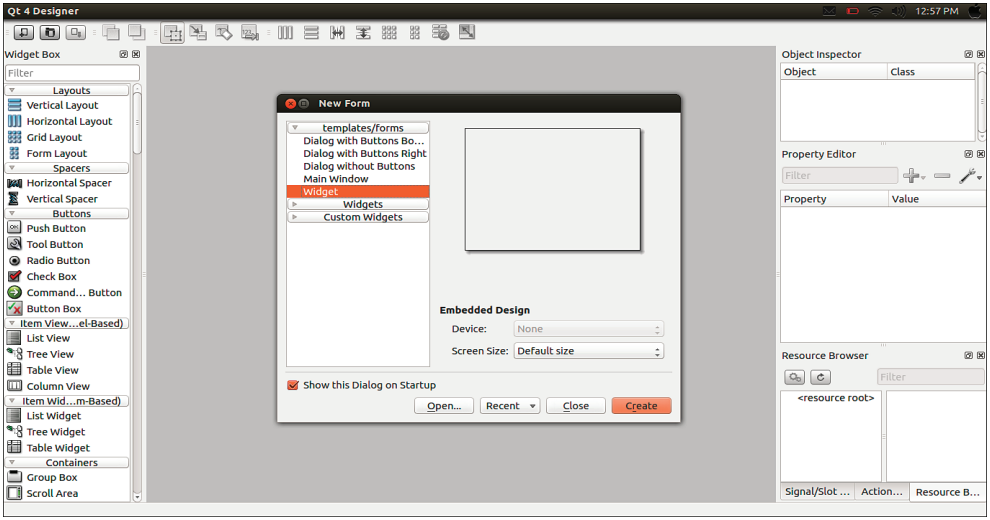
Работа с PyQt и PySide

После установки пакетов PyQt и PySide мы увидим надпись Hello World. Основное различие между PyQt и PySide только в некоторых командах; большинство шагов тождественно. Давайте посмотрим, как сделать графический интерфейс Qt и преобразовать его в код Python.

Представляем Qt Designer

Qt Designer – это инструмент для проектирования и вставки элементов управления в QT GUI. Графический интерфейс Qt – в основном это XML-файл, который содержит информацию о его компонентах и элементах управления. Первый шаг для работы с графическим интерфейсом – это его проектирование. Инструмент Qt Designer предоставляет различные варианты для качественного проектирования приложения.

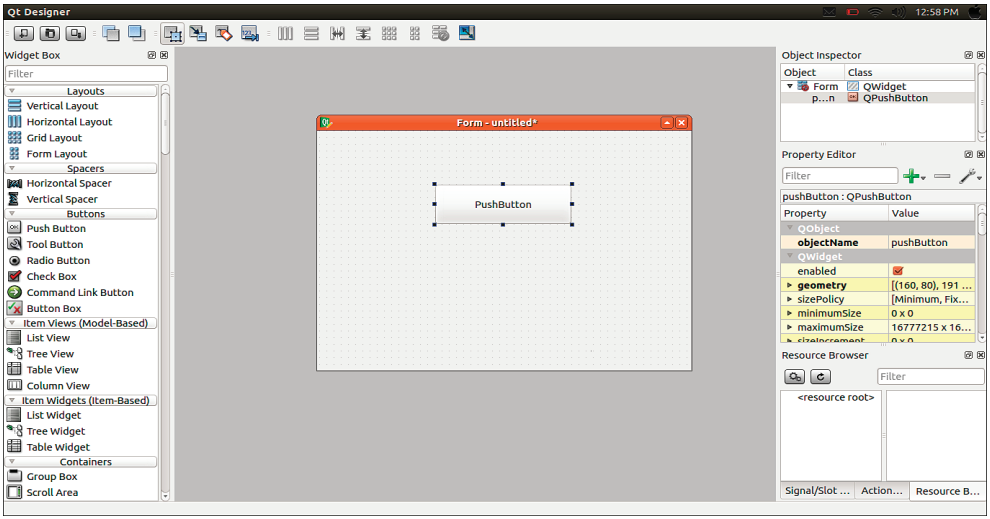
Запустите Qt Designer, введя в терминал команду `designer-qt 4`. На следующем рисунке показано, что появится на экране после выполнения данной команды.



Qt 4 Designer

На предыдущем рисунке показан интерфейс QT Designer. Выберите в окне **New Form** опцию **Widget** и нажмите на кнопку **Create** (Создать). Будет создан новый пустой виджет. Мы можем перетащить в окно нового виджета расположенные в левой части окна **QT 4 Designer** различные элементы управления GUI. Виджеты Qt являются основными строительными блоками графического интерфейса Qt.

На следующем рисунке показана форма с кнопкой, которую перетащили из набора элементов управления в левой части окна **Qt Designer**.



Форма виджета Qt Designer

Приложение **Hello World** содержит кнопку **PushButton**. При нажатии на эту кнопку в терминале появится сообщение **Hello World**. Прежде чем создать приложение **Hello World**, нам нужно понять, что такое сигналы и слоты Qt, так как эти функции будут использованы для создания приложения **Hello World**.

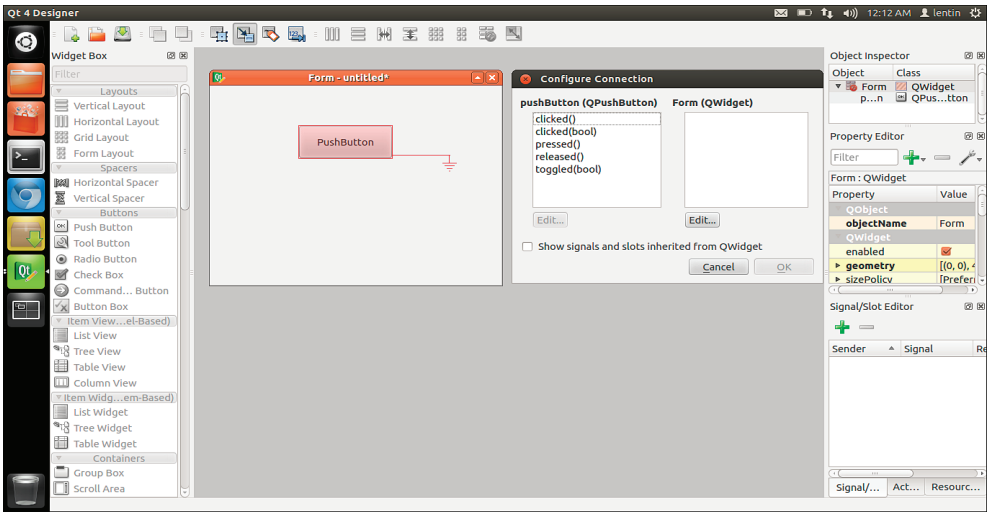
Сигналы и слоты Qt

В Qt события GUI обрабатываются с помощью сигналов и слотов. Когда происходит событие, GUI генерирует сигнал. Виджеты Qt имеют много ранее определенных сигналов, и пользователи могут добавлять пользовательские сигналы для событий GUI. Слот – это функция, вызываемая в ответ на конкретный сигнал. В этом примере мы используем сигнал `clicked()` для **PushButton** и создадим для него слот. Можно написать свой собственный код для этой пользовательской функции. Давайте посмотрим, как мы можем создать кнопку, подключить сигнал к слоту и преобразовать весь GUI в Python.

Вот шаги, связанные с созданием приложения **Hello World GUI**.

1. Перетащите и создайте кнопку **PushButton** из **Qt Designer** в пустую форму.
2. Назначьте слот для события нажатия кнопки, которое выдает `clicked()`.
3. Сохраните разработанный файл пользовательского интерфейса с расширением `.ui`.
4. Конвертируйте файлы интерфейса в Python.
5. Напишите определение пользовательского слота.
6. Распечатайте сообщение **Hello World** внутри определенного слота/функции.

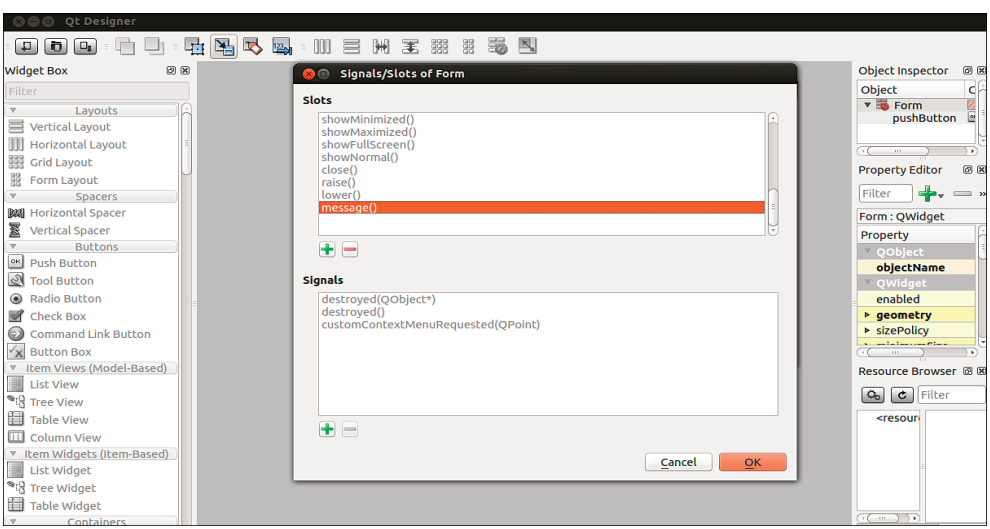
Мы уже перетащили кнопку из **Qt Designer** в пустую форму. Нажмите клавишу **F4** для вставки слота на кнопку. При нажатии клавиши **F4** редактируемая кнопка становится красной, и мы можем перетащить линию от кнопки и поместить символ земли (ground symbol) в основном окне. Этот процесс показан на следующем рисунке.



Назначение слотов и сигналов в Qt 4 Designer

Выберите с левой стороны сигнал `clicked()` и нажмите кнопку **Edit...** для создания нового пользовательского слота. При нажатии кнопки **Edit...** появится еще одно окно для создания пользовательской функции. Пользовательскую функцию можно создать, щелкнув мышью на символе +.

Мы создали пользовательский слот под названием `message()`, как показано на расположенном ниже рисунке.



Назначение слотов и сигналов в Qt 4 Designer

Нажмите на кнопку **OK** и сохраните в файл UI как `***.ui`, выйдите из QT Designer. После сохранения файла UI давайте посмотрим, как мы можем преобразовать файл QT UI в файл Python.

Преобразование UI-файла в код Python

После проектирования UI-файла мы можем его преобразовать в эквивалент Python-кода. Преобразование выполняется с помощью компилятора `pyuic`. Мы уже установили этот инструмент при установке PyQt/PySide. Ниже приведены команды для преобразования пользовательского интерфейса Qt-файла в файл Python. Необходимо использовать разные команды для PyQt и PySide.

Следующая команда преобразует UI в его PyQt-эквивалентный файл:

```
$ pyuic4 -x hello_world.ui -o hello_world.py
```

`pyuic4` – это компилятор UI, который преобразовывает файл UI в его эквивалент в Python-коде. Мы должны упомянуть имя интерфейса после аргумента `-x` и отметить на выходе `filename` после аргумента `-o`.

В команде PySide не так много изменений. Вместо pyuic4 PySide использует `pyside-uic` для преобразования файлов UI в файлы Python. Остальные аргументы одинаковы:

```
$ pyside-uic -x hello_world.ui -o hello_world.py
```

Предыдущая команда создаст эквивалентный код Python для UI-файла. Если мы запустим этот код Python, появится пользовательский интерфейс, разработанный в Qt Designer. Созданный скрипт не будет содержать функции `message()`. Мы должны добавить эту пользовательскую функцию для генерации кода. Следующая процедура поможет добавить функцию. И когда вы нажмете на кнопку, созданная функция `message()` запустится.

Определение и добавление слота в код PyQt

Ниже приведен сгенерированный из PyQt код Python. Код, сгенерированный `pyuic4` и `pyside-uic`, не отличаются друг от друга, за исключением импорта имен модулей. Все другие части одинаковы. Объяснение кода, сгенерированного с помощью PyQt, также применимо и к коду PySide. Код, сгенерированный из приведенного выше преобразования, выглядит следующим образом. Код, структура и параметры могут меняться в зависимости от созданного вами UI-файла:

```
from PyQt4 import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s

class Ui_Form(object):
    def setupUi(self, Form):
```

```
        Form.setObjectName(_fromUtf8("Form"))
        Form.resize(514, 355)
        self.pushButton = QtGui.QPushButton(Form)
        self.pushButton.setGeometry(QtCore.QRect(150, 80, 191, 61))
        self.pushButton.setObjectName(_fromUtf8("pushButton"))
        self.retranslateUi(Form)
        QtCore.QObject.connect(self.pushButton, QtCore.SIGNAL(_fromUtf8("clicked()")),
Form.message)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        Form.setWindowTitle(QtGui.QApplication.translate("Form", "Form", None, QtGui.
QApplication.UnicodeUTF8))
        self.pushButton.setText( QtGui.QApplication.translate("Form", "Press", None, QtGui.
QApplication.UnicodeUTF8))

if __name__ == "__main__":
    import sys
        app = QtGui.QApplication(sys.argv)
        Form = QtGui.QWidget()
        ui = Ui_Form()
        ui.setupUi(Form)
        Form.show()
        sys.exit(app.exec_())
```

Предыдущий код является эквивалентным скриптом Python UI файла Qt, который мы разработали в приложении QT designer. Ниже приведена пошаговая процедура работы данного кода.

1. Выполнение кода начнется с `if __name__ == "__main__":`. Первый в коде PyQt создает объект `QApplication`. Класс `QApplication` содержит основной цикл обработки событий, в котором обрабатываются и отправляются все события из системы Windows и других источников. Он также обрабатывает инициализацию и завершение приложения. Класс `QApplication` находится внутри модуля `QtGui`. Этот код создает объект `QApplication` с именем `app`. Основной код мы должны добавить вручную.
2. Строка `Form = QtGui.QWidget()` создает объект с именем `Form` из класса `QWidget`, который присутствует внутри модуля `QtGui`. Класс `QWidget` – это базовый класс всех объектов пользовательского интерфейса Qt. Он может получить события мыши и события клавиатуры от основной системы Windows.
3. Строка `ui = Ui_Form()` создает объект с именем `ui` из класса `Ui_Form()`, определяемый в данном коде. Объект `Ui_Form()` может принимать класс `QWidget`, который мы создали в предыдущей строке. Он может добавлять кнопки, текст, кнопку управления и другие компоненты пользовательского интерфейса в объекте `QWidget`. В классе `Ui_Form()` содержатся две функции: `setupUi()` и `retranslateUi()`. Можно передать объекту `QWidget` функции `setupUi()`. Эта функция добавит такие компоненты пользовательского интерфейса для этого объекта виджета, как кнопки, назначение слотов

для сигналов и т. д. При необходимости в функции `retranslateUi()` будет переводиться язык UI на другие языки. Например, если нам нужно перевести с английского на испанский, можно в этой функции отметить соответствующее испанское слово.

4. Строка `Form.show()` отображает конечный результат – окно с кнопками и текстом. Далее нужно создать слот функции, которая печатает сообщение **Hello World**. Определение слота создается внутри класса `Ui_Form()`. Следующие шаги вставят слот с именем `message()` в класс `Ui_Form()`.

Определение функции `message()` выглядит следующим образом:

```
def message(self):
    print "Hello World"
```

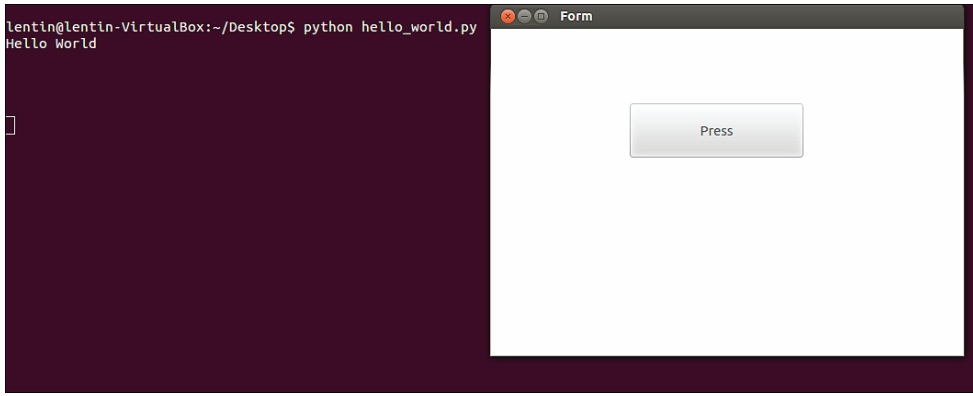
Это должно быть вставлено как функция внутри класса `Ui_Form()`. Кроме того, измените следующую строку в функции `setupUi()` внутри класса `Ui_Form()`:

```
QtCore.QObject.connect(self.pushButton, QtCore.SIGNAL(_fromUtf8("clicked()")), Form.message)
```

Параметр `Form.message` должен быть заменен на параметр `self.message`. Предыдущая строка соединяет сигнал `PushBbutton` функции `clicked()` со слотом `self.message()`, который мы уже вставили в класс `Ui_Form()`.

Работа с приложением Hello World GUI

После замены параметра `Form.message` параметром `self.message` мы можем выполнить код, и результат будет выглядеть следующим образом:



Запуск приложения Pyqt4

Когда мы нажимаем на кнопку **Press**, приложение будет печатать в терминале сообщение **Hello world**. Это все, что нужно знать о настройке GUI с Python и Qt.

В следующем разделе мы рассмотрим GUI, который нужно разработать для робота.

ChefBot – управление с помощью графического интерфейса

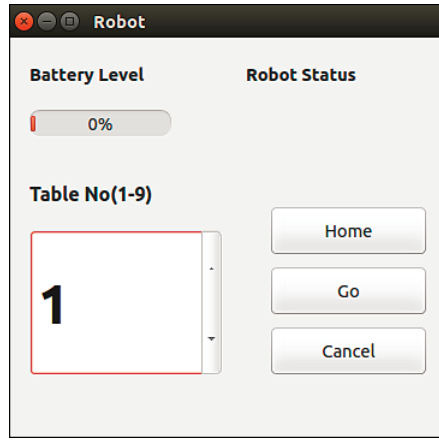
После завершения работы приложения **Hello World** в PyQt обсудим графический интерфейс для управления ChefBot. Основное использование построения GUI – это создание более простого способа управления роботом. Человек, управляющий этим роботом, не должен знать комплекс команд для запуска и остановки этого устройства. Создание GUI для ChefBot облегчит управление роботом. Мы планируем создать графический интерфейс с помощью PyQt, ROS и интерфейса Python. Пакет Chef BotROS доступен на GitHub по следующей ссылке: https://github.com/qboticslabs/Chefbot_ROS_pkg.git.

Если вы еще не скопировали код, то можете сделать это с помощью следующей команды:

```
$ git clone https://github.com/qboticslabs/Chefbot_ROS_pkg.git
```

Код GUI с именем `robot_gui.py` помещается в папку `scripts`, которая находится внутри пакета `chefbot_bringup`.

Следующий рисунок показывает интерфейс, который был разработан для ChefBot.



Запуск приложения Pyqt4

Графический интерфейс имеет следующие функции:

- контроль состояния батареи робота. В поле **Robot status** (Состояние робота) будут показаны обнаруженные роботом ошибки;
- в поле ввода **Table** (Стол) указывается номер стола, к которому требуется подвезти еду. В данном приложении предусмотрено наличие девяти столов, но это значение можно изменить, исходя из фактического количества столов в помещении. После ввода номера стола дается команда

на движение по маршруту от фактического положения робота к целевому столу. Для этого достаточно нажать кнопку **Go** (Ехать). Чтобы вернуть робота в исходную позицию, следует нажать кнопку **Home** (Домой). Для отмены команды нажмите кнопку **Отмена** (Cancel).

Это приложение работает следующим образом.

Прежде всего необходимо создать карту помещения. После создания карту следует сохранить в памяти бортового ПК. Робот создает карту один раз. Далее запускаются процедуры локализации и навигации. Пакет ChefBot ROS поставляется с картой и имитационной моделью помещения гостиницы. Теперь, чтобы протестировать GUI, мы можем запустить моделирование и локализацию. Позже обсудим, как управлять роботом с помощью графического интерфейса. Если вы установите пакеты ChefBot ROS местной системы, то сможете сымитировать окружающую среду и проверить работу GUI.

Для запуска имитации ChefBot в помещении используйте следующую команду:

```
$roslaunch chefbot_gazebo chefbot_hotel_world.launch
```

После запуска моделирования ChefBot мы можем запустить процедуры локализации и навигации, используя уже построенную карту. Карта находится внутри пакета карт `chefbot_bringup`. Мы воспользуемся этой картой для выполнения данного теста. Для загрузки программы локализации и навигации выполните следующую команду:

```
$ roslaunch chefbot_gazebo amcl_demo.launch map_file:=/home/lentin/catkin_ws/src/chefbot/
chefbot_bringup/map/hotel1.yaml
```

В вашей системе путь к файлу карты может быть другим. Поэтому в вашей системе необходимо использовать фактический путь сохранения файла.

Если вы указали правильный путь, робот начнет выполнение стека ROS-навигации. Чтобы увидеть положение робота на карте или вручную установить начальное положение робота, запустите RViz, используя следующую команду:

```
$ roslaunch chefbot_bringup view_navigation.launch
```

В RViz с помощью кнопки **2D Nav Goal** мы можем приказать роботу перейти к любым координатам карты.

Мы можем приказать роботу перейти к любым координатам карты, используя инструменты программирования. Стек навигации ROS работает с помощью библиотеки ROS `actionlib`. Библиотека ROS `actionlib` предназначена для выполнения выделенных задач, это имитатор ROS Services. Преимущество над услугами ROS заключается в том, что мы можем отменить запрос в любое время.

В графическом интерфейсе мы можем приказать роботу перейти по карте по координатам, используя библиотеку `actionlib` Python. Применяя данный метод, мы можем получить положение требуемого стола на карте.

После запуска симулятора и узлов AMCL запустите удаленное управление и с помощью клавиатуры перемещайте робот рядом с каждым столом. Используйте следующую команду, чтобы перемещать и разворачивать робот:

```
$ rosrun tf tf_echo /map /base_link
```

После нажатия кнопки **Go** эта позиция подается в стек навигации. Робот планирует свой путь и достигает заданной цели. Задание можно отменить в любое время. Так, графический интерфейс ChefBot действует как клиент `actionlib`, который отправляет координаты карты на сервер `actionlib`, то есть как стек навигации.

Мы можем запустить GUI для управления роботом, используя следующую команду:

```
$ rosrun chefbot_bringup robot_gui.py
```

Можно выбрать номер стола и для перемещения робота нажать на кнопку **Go**.

Мы предполагаем, что файлы были скопированы, а файл `robot_gui.py` открыт. В этом случае мы можем обсудить главные слоты, которые были добавлены в класс `Ui_Form()` для клиента `actionlib` и получения значения уровня заряда батареи и состояния робота.

Для этого нам нужно импортировать следующие модули Python для GUI-приложений:

```
import rospy
import actionlib
from move_base_msgs.msg import *
import time
from PyQt4 import QtCore, QtGui
```

Дополнительные модули мы вызываем из клиента ROS Python: `rospy` и `actionlib` для отправки значений в стек навигации. Модуль `move_base_msgs` содержит определение сообщения цели, которое должно быть отправлено в стек навигации.

Позиция робота возле каждого стола упоминается в словаре Python. Следующий код показывает значения hardcode позиции робота возле каждого стола:

```
table_position = dict()
table_position[0] = (-0.465, 0.37, 0.010, 0, 0, 0.998, 0.069)
table_position[1] = (0.599, 1.03, 0.010, 0, 0, 1.00, -0.020)
table_position[2] = (4.415, 0.645, 0.010, 0, 0, -0.034, 0.999)
table_position[3] = (7.409, 0.812, 0.010, 0, 0, -0.119, 0.993)
table_position[4] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
table_position[5] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
table_position[6] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
table_position[7] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
table_position[8] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
table_position[9] = (1.757, 4.377, 0.010, 0, 0, -0.040, 0.999)
```

Мы можем получить доступ к позиции робота возле каждого стола, обратившись к этому словарю.

В настоящее время для демонстрации мы вставили только четыре значения. Вы можете добавить дополнительные значения, определив положение других столов.

Далее необходимо назначить некоторые переменные для обработки: номер стола, положение робота и клиента `actionlib` внутри класса `Ui_Form()`.

```
#Номер стола
self.table_no = 0
#Сохраняет текущее положение робота у стола
self.current_table_position = 0
#Создание клиента Actionlib
self.client = actionlib.SimpleActionClient('move_base',MoveBaseAction)
#Создание сообщения цели
self.goal = MoveBaseGoal()
#Запустите эту функцию для обновления состояния батареи и робота
self.update_values()
```

В следующем коде показаны назначение сигналов и слотов в этом коде для кнопок и счетчик виджета:

```
#Обработчик сигнала и назначение слота  set_table_number()
QtCore.QObject.connect(self.spinBox, QtCore.SIGNAL(_fromUtf8("valueChanged(int)")), self.
set_table_number)
#Обрабатывать сигнал кнопки Home и назначить слот Home()
QtCore.QObject.connect(self.pushButton_3, QtCore.SIGNAL(_fromUtf8("clicked()")), self.Home)
#Обрабатывать сигнал  кнопки Go и назначить слот Go()
QtCore.QObject.connect(self.pushButton, QtCore.SIGNAL(_fromUtf8("clicked()")), self.Go)
#Обрабатывать сигнал кнопки Cansel () и назначить слот Cancel()
QtCore.QObject.connect(self.pushButton_2, QtCore.SIGNAL(_fromUtf8("clicked()")), self.Cancel)
```

Следующий слот обрабатывает значение спин-бокса из пользовательского интерфейса и назначает номер стола. Также он преобразовывает номер стола в соответствующие координаты положения робота:

```
def set_table_number(self):
    self.table_no = self.spinBox.value()
    self.current_table_position = table_position[self.table_no]
```

Ниже дано определение слота Go для кнопки **Go**. Эта функция вставит в положение робота положение выбранного стола в заголовок сообщения цели и отправит его в стек навигации:

```
def Go(self):
    #Присвоение X, Y и Z позиции и ориентации на target_pose сообщением self.goal.target_
pose.pose.position.x=float(self.current_table _position[0])
    self.goal.target_pose.pose.position.y=float(self.current_table _position[1])
    self.goal.target_pose.pose.position.z=float(self.current_table _position[2])
    self.goal.target_pose.pose.orientation.x = float(self.current_table_position[3])
    self.goal.target_pose.pose.orientation.y= float(self.current_table_position[4])
    self.goal.target_pose.pose.orientation.z= float(self.current_table_position[5])
```

```
    # ID фрейма
    self.goal.target_pose.header.frame_id= 'map'
    # Отметка времени
    self.goal.target_pose.header.stamp = rospy.Time.now()
    # Отправка цели в стек навигации
    self.client.send_goal(self.goal)
```

Приведенный ниже код является определением слота `Cancel()`. Он отменит всю задачу робота, которая планировалась к выполнению.

```
def Cancel(self):
    self.client.cancel_all_goals()
```

Следующий код определяет функцию `Home()`. Данная функция обнуляет положение стола и вызывает функцию `Go()`. При этом текущим положением робота считается позиция обнуленного стола:

```
def Home(self):
    self.current_table_position = table_position[0]
    self.Go()
```

Следующие определения относятся к функциям `update_values()` и `add()`. Метод `update_values()` начнет обновление уровня заряда батареи и состояния робота в потоке. Функция `add()` извлекает параметры ROS состояния батареи и состояния робота и устанавливает их на индикаторе выполнения и индикаторе батареи:

```
def update_values(self):
    self.thread = WorkThread()
    QtCore.QObject.connect( self.thread, QtCore.SIGNAL("update(QString)"), self.add )
    self.thread.start()

def add(self,text):
    battery_value = rospy.get_param("battery_value")
    robot_status = rospy.get_param("robot_status")
    self.progressBar.setProperty("value", battery_value)
    self.label_4.setText(_fromUtf8(robot_status))
```

Ниже приведен класс `WorkThread()`, используемый в предыдущей функции, Класс `WorkThread()` наследуется от `QThread`, предоставляемого Qt для организации многопоточности. Поток с определенной задержкой выдает обновление сигнала (`Qstring`). Предшествующая функция `update_values()` сигнала `update(QString)` подключена к слоту `self.add()`, и когда обновление сигнала (`QString`) испускается из потока, будет вызван слот `add()` и начнется обновление значения состояния аккумулятора:

```
class WorkThread(QtCore.QThread):
    def __init__(self):
        QtCore.QThread.__init__(self)
    def __del__(self):
        self.wait()
    def run(self):
```

```
        while True:
            time.sleep(0.3) # artificial time delay
            self.emit( QtCore.SIGNAL('update(QString)'), " " )
            return
```

Мы обсудили, как сделать графический интерфейс для ChefBot, но этот GUI только для пользователя, управляющего роботом ChefBot. Для отладки и проверки данных робота следует применить другие инструменты. ROS предоставляет отличный инструмент отладки для визуализации данных робота.

Инструмент rqt – это популярный инструмент ROS. Он основан на Qt-based framework для GUI для ROS. Давайте обсудим инструмент rqt, процедуру установки и то, как мы можем проверить данные датчика от робота.

Установка и работа с RQT в Ubuntu 16.04 LTS

Инструмент **rqt** – это программный фреймворк в ROS, реализующий различные графические инструменты в виде подключаемых модулей. Мы можем добавить плагины в качестве закрепляемых окон в rqt.

Установка **rqt** в Ubuntu 16.04 может выполняться с помощью следующей команды. Перед установкой убедитесь, что у вас установлена полная версия ROS Kinetic.

```
$ sudo apt-get install ros-<ros_version>-rqt
```

После установки пакетов rqt мы можем получить доступ к реализации GUI rqt под названием `rqt_gui`, в котором мы можем закрепить плагины `rqt` в одном окне.

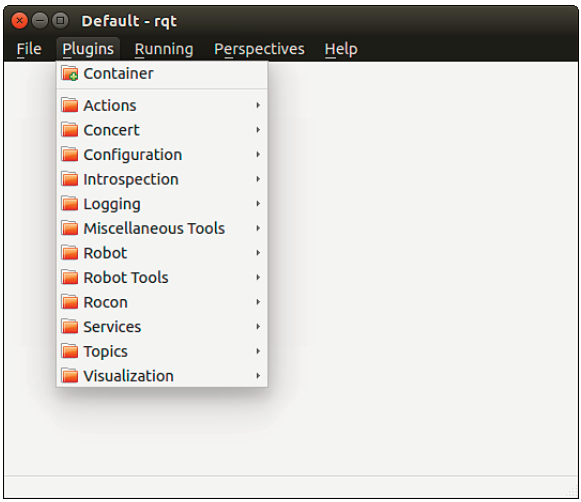
Перед запуском `rqt_gui` выполните в терминале команду roscore:

```
$ roscore
```

Выполните следующую команду, чтобы запустить `rqt_gui`:

```
$ rosrun rqt_gui rqt_gui
```

При правильной работе команд мы получим следующее окно:



Запуск rqt

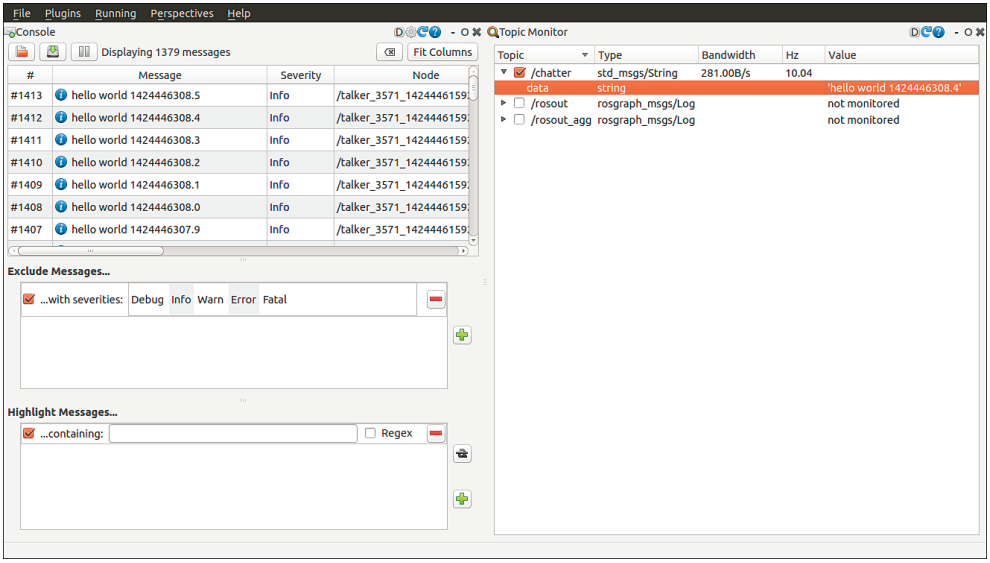
Плагины мы можем загружать и выгружать во время выполнения. Для анализа сообщений ROS следует загрузить плагин **Console** из **Plugins** → **Logging** → **Console**. В следующем примере мы загружаем подключаемый модуль консоли и запускаем узел `talker` внутри `rospy_tutorials`, который отправит сообщение **Hello World** в тему (**Topic**) под названием `/chatter`.

Для запуска узла **talker.py** выполните следующую команду:

```
$rosrun rospy_tutorials talker.py
```

На расположенном ниже рисунке `rqt_gui` загружается с двух плагинов под именем **Console** и **Topic Monitor**. Плагин **Topic Monitor** загружается из **Plugins** → **Topics** → **Topic Monitor**. Плагин **Console** отслеживает печать сообщений на каждом узле и степень их важности. Это очень полезно для отладки. На этом рисунке левая часть `rqt_gui` загружается с плагином **Console**, а правая часть – с **Topic Monitor**. **Topic Monitor** перечислит доступные темы и проконтролирует их значения.

На расположенном ниже рисунке подключаемый модуль **Console** отслеживает узел сообщения `talker.py` и их уровень серьезности (важности), тогда как **Topic Monitor** отслеживает значение внутри темы `/chatter`.



Запуск rqt с различными плагинами

Мы также можем визуализировать такие данные, как изображения и графики на `rqt_gui`. Для навигации и проверки робота существуют плагины для встраивания RViz в `rqt_gui`. Плагин **Navigation viewer** просматривает тему `/map`. Визуализация плагина доступна в **Plugin → Visualization**.

Итоги

В этой главе мы обсудили создание GUI для управления ChefBot обычным пользователем, который не знает, как работает этот робот. Мы использовали привязку Python Qt под названием PyQt для создания данного графического интерфейса. Прежде чем мы перешли к проектированию GUI, мы, чтобы облегчить понимание PyQt, в качестве примера рассмотрели создание приложения Hello World. Пользовательский интерфейс создавался с помощью QT Designer, а файл UI с помощью компилятора UI Python был преобразован в эквивалентный скрипт Python. После проектирования основного GUI в QT Designer мы конвертировали файл UI в скрипт Python и вставили в него необходимые слоты. Для запуска робота следует в графическом интерфейсе задать номер стола, к которому должен переместиться робот, и отдать приказ на движение. В карте отмечено положение каждого стола. Карта зашита в скрипте Python и может быть использована для тестирования. После выбора целевого стола мы определяем положение робота на карте, и когда нажмем кнопку Go, робот начнет движение в целевую точку. Пользователь может отменить операцию в любое время, и робот после отмены должен вернуться в исходное положение. При

необходимости GUI может в реальном времени получать данные о состоянии аккумуляторной батареи и самого робота. После обсуждения графического интерфейса был рассмотрен инструмент отладки GUI в ROS. Этот инструмент называется rqt. Также были рассмотрены некоторые плагины, используемые для отладки данных, получаемых от робота.

Вопросы

1. Какие популярные наборы инструментов пользовательского интерфейса доступны на платформе Linux?
2. Каковы различия между привязками PyQt и PySide Qt?
3. Как преобразовать файл Qt UI в Python script?
4. Что такое сигналы и слоты Qt?
5. Что такое инструмент rqt, и каковы его основные приложения?

Дополнительная литература

Дополнительно о системе технического зрения роботов с операционной системой ROS вы можете прочитать, перейдя по следующим ссылкам:

- http://wiki.ros.org/rqt/UserGuide;
- http://wiki.ros.org/rqt/Tutorials.

Подводим итоги

Глава 1 «Начало работы с операционной системой робота ROS»

1. Три главные функции ROS:
   - интерфейс передачи данных между разными программами;
   - стандартный алгоритм для быстрой разработки роботов;
   - программные инструменты для визуализации и отладки данных.
2. Различные уровни концепций в ROS – это уровень файловой системы ROS, уровень графа вычислений ROS и уровень сообщества ROS.
3. Система сборки Catkin построена с использованием сценариев CMake и Python. Этот инструмент помогает нам создавать пакеты ROS.
4. Тема ROS – это именованная шина, в которой один узел может взаимодействовать с другим узлом. Тип сообщения, используемый в темах, – сообщения ROS.
5. Различное понятие графа вычисления ROS – узлы ROS, ROS-темы, сообщения ROS, ROS Master, ROS Services и ROS Bags.
6. Ros Master действует как программа-посредник для соединения двух узлов ROS для начала взаимодействия друг с другом.
7. Важные особенности Gazebo:
   - динамическое моделирование: включает движок, такой как ODE, Bullet, Simbody и Dart;
   - улучшенная 3D-графика: для создания высококачественного освещения, теней и структур используется структура OGRE;
   - поддержка плагинов: данная возможность позволяет добавлять роботу новые функции и датчики, контролирующие состояние окружающей среды;
   - протокол TCP/IP: управление Gazebo с помощью пользовательского интерфейса передачи данных.

Глава 2 «Основные понятия о роботах с дифференциальным приводом»

1. Голономные роботы могут свободно перемещаться в любом направлении, а управляемые степени свободы равны суммарным степеням свободы. Колесные роботы Omni являются примером голономных роботов. Голономные роботы ограничены в движении. Поэтому количество управляемых степе-

ней свободы не будет равно количеству полных степеней свобод. Дифференциальный привод является примером неголономной конфигурации.
2. Кинематика робота рассматривает движение робота без учета массы и инерции, тогда как динамика робота – отношения между массой и инерцией.
3. ICC – это мгновенный центр искривления, воображаемая точка на оси вращения колеса робота. Вокруг этой точки поворачивается робот.
4. Это процесс нахождения текущего положения робота от скорости колеса.
5. Нахождение скорости колеса для достижения цели.

Глава 3 «Моделирование робота с дифференциальным приводом»

1. Моделирование робота – процесс создания 2D- и 3D-модели робота с наличием всех параметров, в которые входят кинематические и динамические параметры.
2. В двухмерной модели учитываются точные геометрические размеры всех деталей, из которых состоит робот. Эти данные позволяют вычислить кинематику робота и изготовить все необходимые детали.
3. 3D-модель робота – точная копия аппаратных средств робота, со всеми физическими параметрами робота, разработанного с использованием программного обеспечения CAD.
4. Если вы знаете API-сценарии Blender, создать 3D-модель с помощью сценариев Python намного проще и точнее, чем ручное моделирование.
5. URDF является 3D-модель робота в ROS. Модель имеет кинематические и динамические параметры робота.

Глава 4 «Моделирование дифференциального привода робота, управляемого операционной системой ROS»

1. Модель датчика может быть создана в Gazebo с помощью плагинов Gazebo. Модель датчика может быть написана на языке программирования C++, который интегрирован в симулятор.
2. ROS соединяется с Gazebo с использованием плагина ROS. Когда мы загружаем этот плагин в Gazebo, мы можем управлять Gazebo через интерфейс ROS.
3. Важные признаки `<inertia>`, `<collision>` и `<gazebo>`.
4. Пакет Gmapping в ROS – это реализация алгоритма Fast SLAM, который можно использовать в роботе для отображения среды и локализации. Использование Gmapping в ROS – это простой процесс, включающий узел отображения с необходимыми параметрами и такими темами, как одометрия и лазерное сканирование.
5. Узел `Move_base` предназначен для работы с различными подсистемами навигации в роботе. Он получает координаты для обработки глобального и ло-

кального планировщиков и карты робота. Как только узел получает целевую позицию, эти данные подаются в навигационную подсистему для достижения данной целевой позиции.

6. AMCL означает адаптивную локализацию по методу Монте-Карло, которая представляет собой алгоритм локализации робота на заданной карте. В ROS есть пакет ROS для развертывания AMCL в нашем роботе. Мы можем запустить узел amcl с правильным вводом и необходимыми параметрами.

Глава 5 «Проектирование оборудования и схем ChefBot»

1. Это процесс поиска соответствующих проекту аппаратных компонентов. В проектирование входят разработка схемы и вычисление электрического тока и напряжения для каждого компонента.
2. Коммутационная цепь для управления скоростью вращения и направления вращения ведущих валов двигателей.
3. Главные компоненты – энкодер колеса, позволяющий вычислять скорость вращения и направление вращения колеса и датчики для обнаружения препятствий. Это такие датчики, как лазерный сканер и датчик глубины.
4. Необходимо проверить, соответствует ли он спецификации робота.
5. Картографирование, обнаружение и отслеживание препятствий.

Глава 6 «Согласование приводов и датчиков с контроллером робота»

1. Коммутационная схема для контроля скорости и направления вращения ведущих валов двигателей.
2. Энкодер, позволяющий отслеживать скорость вращения и направление вращения колеса.
3. 4X-кодирование сигналов энкодера для получения максимального количества импульсов на один оборот.
4. Зная полученное количество импульсов от энкодера, мы легко можем вычислить смещение колеса.
5. Умный привод с обратной связью между двигателем и микроконтроллером, который взаимодействует с бортовым ПК и всеми датчиками.
6. Ультразвуковой датчик с ультразвуковым излучателем и приемником эхо-сигнала. Расстояние до препятствия рассчитывается по разнице времени между излученным и принятым сигналами.
7. Диапазон = Время между моментом излучения ультразвукового сигнала и приемом эхо-сигнала * Скорость (340 м/с) / 2.
8. Излучается ультразвуковой импульс, и принимается эхо-сигнал. Расстояние высчитывается с использованием следующего уравнения:

   Range = (6787 / (Volt – 3)) – 4.

Глава 7 «Согласование датчиков зрения с ROS»

1. У большинства 3D-датчиков глубины есть дополнительные датчики зрения для обнаружения глубины. Работа этих датчиков основана на стереоскопическом зрении.
2. Интерфейс передачи сообщений, инструменты для визуализации и отладки всех систем робота, стандартные алгоритмы робота.
3. OpenCV обладает алгоритмом машинного зрения, алгоритмы OpenNI реализуют приложения NI и PCL для обработки облака точек.
4. Одновременная локализация и отображение. Данный алгоритм используется для отображения окружающей среды и одновременной локализации в реальном времени.
5. Это алгоритм для нанесения окружающей среды в 3D-карту робота.

Глава 8 «Сборка робота ChefBot и интеграция программного обеспечения»

1. Преобразователь уровня напряжений между драйвером двигателя, датчиками, микроконтроллером и бортовым ПК. Он преобразует низкоуровневые данные в эквивалентные данные ROS.
2. PID-механизм обратной связи в управлении для достижения целевого положения робота.
3. Используя данные энкодера, мы, применяя кинематические уравнения робота, можем вычислить пройденное роботом расстояние. Это данные одометра.
4. В основном используется составленная карта окружающей среды.
5. Чаще всего используется для локализации робота на статической карте.

Глава 9 «Разработка графического интерфейса для робота с использованием Qt и Python»

1. QT и GTK.
2. Обе привязки почти одинаковы, разница только в названии. PyQt имеет лицензию GPL и PySide, поставляемую с LGPL. Кроме того, имеется большое количество документации PySide о своих API.
3. Использование компилятора Py UI с названием pyuic.
4. Слоты Qt – это функции в программе, которые могут быть вызваны сигналом Qt. Например, clicked – это сигнал, который может вызвать функцию с именем hello.
5. Rqt является одним из полезных инструментов GUI в ROS. Мы можем создавать плагины rqt и вставлять их в графический интерфейс rqt. Существуют плагины для визуализации, отладки и т. д. в rqt.

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Лентин Джозеф

Изучение робототехники с использованием Python

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Редактор *Герасименко А. С.*
Перевод *Корягин А. В.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 20,31. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**