# Содержание

**Составители** ..... 11

**Введение** ..... 12

**Глава 1. Начало работы с операционной системой для робота (ROS)** ..... 17

Технические условия ..... 17
Введение в ROS ..... 18
Концепции ROS ..... 20
Установка ROS на Ubuntu ..... 23
Введение в catkin ..... 28
Создание пакета ROS ..... 29
Введение в Gazebo ..... 36
Итоги ..... 39
Вопросы ..... 39

**Глава 2. Основные понятия роботов с дифференциальным приводом** ..... 40

Математическое моделирование робота ..... 40
Введение в систему дифференциального привода и кинематику робота ..... 41
Прямая кинематика дифференциального робота ..... 43
Объяснение уравнений прямой кинематики ..... 43
Обратная кинематика ..... 48
Итоги ..... 49
Вопросы ..... 49
Дополнительная информация ..... 49

**Глава 3. Моделирование робота с дифференциальным приводом** ..... 50

Технические требования ..... 51
Требования к сервисному роботу ..... 51
Приводной механизм ходовой части робота ..... 51
Выбор двигателей и колес ..... 52
Результаты проектирования ..... 53

Конструкция шасси робота ..... 53
Установка LibreCAD, Blender и MeshLab ..... 55
Установка LibreCAD ..... 56
Установка Blender ..... 56
Установка MeshLab ..... 56
Создание 2D CAD-чертежа робота с помощью LibreCAD ..... 56
Конструкция опорной плиты робота ..... 59
Конструкция нижней и верхней стоек ..... 61
Конструкция колеса и крепления для колеса и мотора ..... 62
Конструкция опорного колеса ..... 65
Конструкция средней плиты ..... 66
Конструкция верхней плиты ..... 67
Работа с 3D-моделью робота с использованием Blender ..... 68
Скрипты Python в Blender ..... 69
Введение в API Blender Python ..... 70
Скрипт Python модели робота ..... 72
Создание модели URDF-робота ..... 79
Создание пакета описания ChefBot в ROS ..... 80
Итоги ..... 84
Вопросы ..... 85
Дополнительная информация ..... 85

**Глава 4. Моделирование дифференциального привода робота, управляемого операционной системой ROS** ..... 86

Технические условия ..... 87
Начало работы с симулятором Gazebo ..... 87
Графический интерфейс пользователя Gazebo ..... 88
Работа с симулятором TurtleBot 2 ..... 92
Перемещение робота ..... 97
Создание симуляции в Chefbot ..... 99
Преобразование глубины изображения с помощью лазерного сканера ..... 100
Теги и плагины URDF для моделирования Gazebo ..... 101
Визуализация данных датчика робота ..... 106
Начало работы с одновременной локализацией и картографированием ..... 108
Создание карты с помощью SLAM ..... 109
Начало работы с адаптивным методом локализации Монте-Карло ..... 110
Реализация AMCL в среде Gazebo ..... 112
Автономная навигация Chefbot в отеле с использованием Gazebo ..... 114
Итоги ..... 115
Вопросы ..... 115
Дополнительная информация ..... 115

**Глава 5. Проектирование оборудования и схем ChefBot** ..... 116
Технические условия ..... 117
Спецификации оборудования Chefbot ..... 117
Структурная схема робота ..... 117
Двигатель и энкодер ..... 118
Драйвер двигателя ..... 120
Встроенный контроллер ..... 123
Ультразвуковые датчики ..... 124
Инерциальный блок измерения (акселерометр и гироскоп) ..... 126
Kinect/Orbbec Astra ..... 127
Центральный процессор ..... 128
Динамики/микрофон ..... 129
Источник питания/аккумулятор ..... 130
Как работает оборудование ChefBot? ..... 131
Итоги ..... 132
Вопросы ..... 133
Дополнительная информация ..... 133

**Глава 6. Согласование приводов и датчиков с контроллером робота** ..... 134
Технические условия ..... 135
Согласование редукторного двигателя постоянного тока с Tiva C LaunchPad ..... 135
Дифференциальный привод колесного робота ..... 137
Установка Energia IDE ..... 139
Код взаимодействия с двигателями ..... 143
Интерфейс квадратурного энкодера с Tiva C Launchpad ..... 146
Обработка данных энкодера ..... 147
Код согласования квадратурного энкодера ..... 150
Работа с приводом Dynamixel ..... 154
Работа с ультразвуковыми датчиками расстояния ..... 157
Согласование HC-SR04 с Tiva C LaunchPad ..... 157
Работа с ИК-датчиком расстояния ..... 163
Работа с инерционным измерительным модулем ..... 166
Инерциальная навигация ..... 166
Взаимодействие MPU 6050 с Tiva C LaunchPad ..... 167
Код согласования в Energia ..... 170
Итоги ..... 173
Вопросы ..... 173
Дополнительная информация ..... 173

**Глава 7. Согласование датчиков зрения с ROS** ..... 174
Технические требования ..... 174
Список робототехнических датчиков зрения и библиотек для работы с изображением ..... 175
Pixy2/CMUcam5 ..... 175
Веб-камера Logitech C920 ..... 176
Kinect 360 ..... 176
Intel RealSense серии D400 ..... 177
Датчик глубины изображения Orbbec Astra ..... 179
Введение в OpenCV, OpenNI и PCL ..... 180
Что такое OpenCV? ..... 180
Что такое OpenNI? ..... 184
Что такое PCL? ..... 185
Программирование Kinect с использованием Python ROS, OpenCV и OpenNI ..... 186
Как запустить драйвер OpenNI ..... 186
Интерфейс ROS в формате OpenCV ..... 187
Согласование Orbbec Astra с ROS ..... 191
Установка драйвера Astra-ROS ..... 191
Работа с облаками точек с помощью Kinect, ROS, OpenNI и PCL ..... 192
Открытие устройства и создание облака точек ..... 192
Преобразование данных облака точек в данные лазерного сканирования ..... 193
Работа в SLAM с помощью ROS и Kinect ..... 195
Итоги ..... 196
Вопросы ..... 196
Дополнительная информация ..... 196

**Глава 8. Создание аппаратного обеспечения ChefBot и интеграция программного обеспечения** ..... 197
Технические требования ..... 197
Сборка ChefBot из ранее изготовленных деталей и комплектующих ..... 198
Конфигурирование бортового компьютера ChefBot и установка пакетов ChefBot ROS ..... 203
Согласование датчиков ChefBot с Tiva C LaunchPad ..... 204
Встроенный код для ChefBot ..... 206
Написание драйвера Ros Python для ChefBot ..... 208
Функции исполняемого файла ChefBot ROS ..... 212
Элементы Python ChefBot и запуск файлов ..... 213
Построение карты комнаты с помощью SLAM в ROS ..... 220

ROS: локализация и навигация ..... 221
Итоги ..... 223
Вопросы ..... 224
Дополнительная информация ..... 224

**Глава 9. Разработка графического интерфейса для робота с использованием Qt и Python** ..... 225
Технические требования ..... 226
Установка Qt на Ubuntu 16.04 LTS ..... 226
Взаимодействие Python и Qt ..... 227
PyQt ..... 227
PySide ..... 227
Работа с PyQt и PySide ..... 228
Представляем Qt Designer ..... 228
Сигналы и слоты Qt ..... 230
Преобразование UI-файла в код Python ..... 232
Определение и добавление слота в код PyQt ..... 232
Работа с приложением Hello World GUI ..... 234
ChefBot – управление с помощью графического интерфейса ..... 235
Установка и работа с rqt в Ubuntu 16.04 LTS ..... 240
Итоги ..... 242
Вопросы ..... 243
Дополнительная литература ..... 243

**Подводим итоги** ..... 244

**Предметный указатель** ..... 248

# 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.

ПЕРЕВОД

## 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.

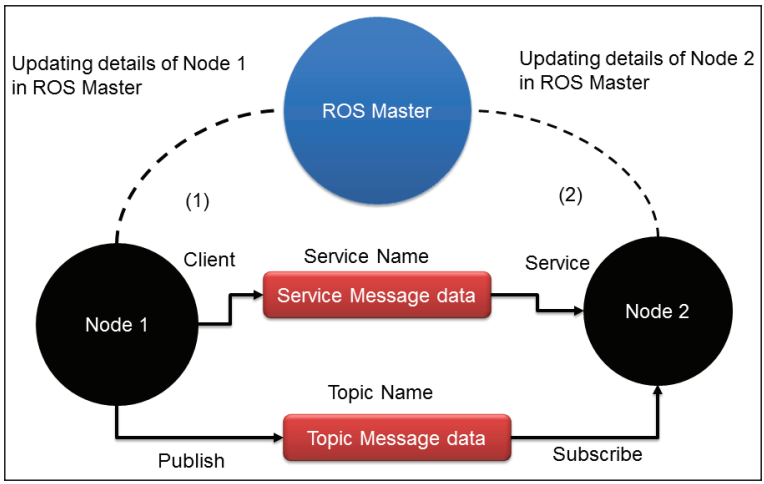
- **Узлы** – это вычислительные процессы в ROS. Каждый узел выполняет обработку определенных данных. Например, один узел обрабатывает данные с лазерного сканера, другой публикует эти данные и т. д. Узел ROS создается с помощью клиентской библиотеки ROS (например, roscpp и rospy). Более подробно мы познакомимся 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:

ПЕРЕВОД




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 |

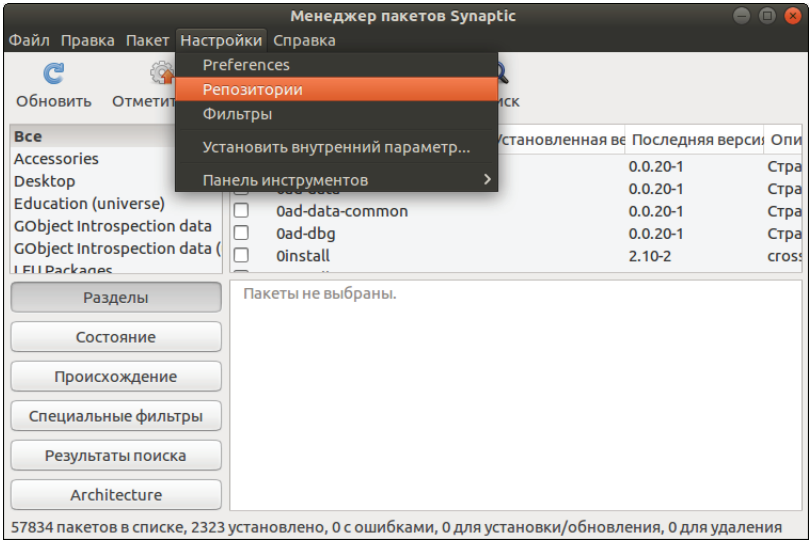
Теперь рассмотрим процедуру установки стабильной версии дистрибутива ROS Kinetic с долгосрочной поддержкой (LTS) на Ubuntu 16.04.3 LTS. Разработчики ROS Kinetic Kame в первую очередь ориентировались на Ubuntu 16.04 LTS. Вы также можете найти инструкции по настройке ROS LTS Melodic Morenia на Ubuntu 18.04 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.

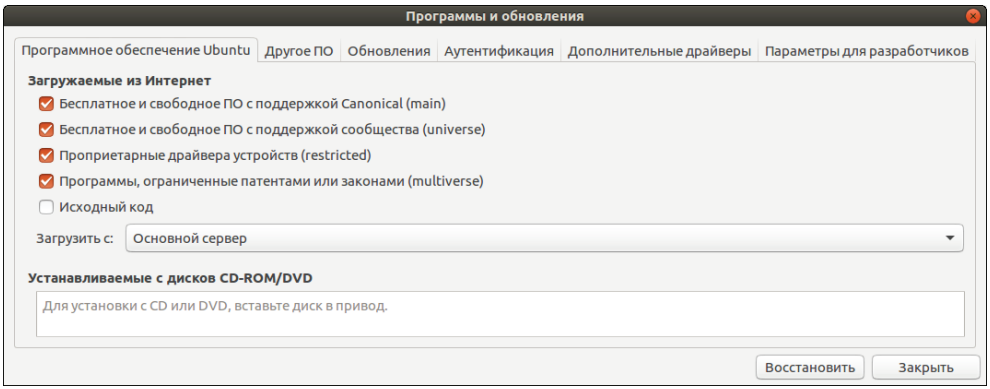
Чтобы установить ROS, выполните следующие действия.

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



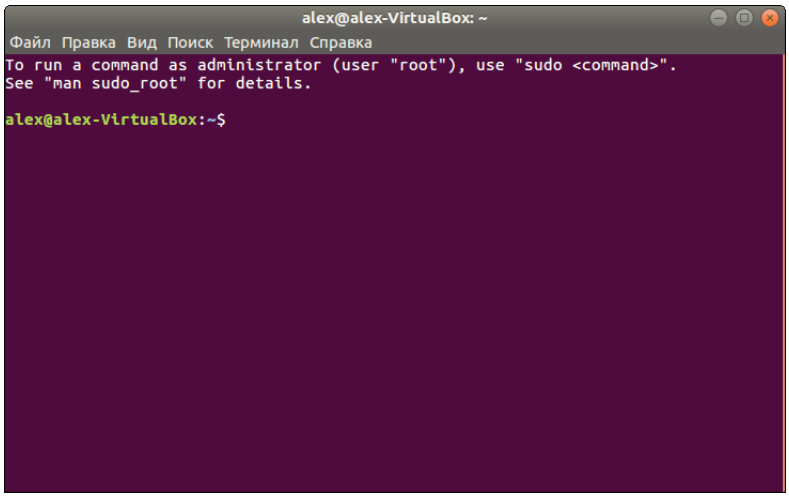

Команда **Настройки → Репозитории**

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
```

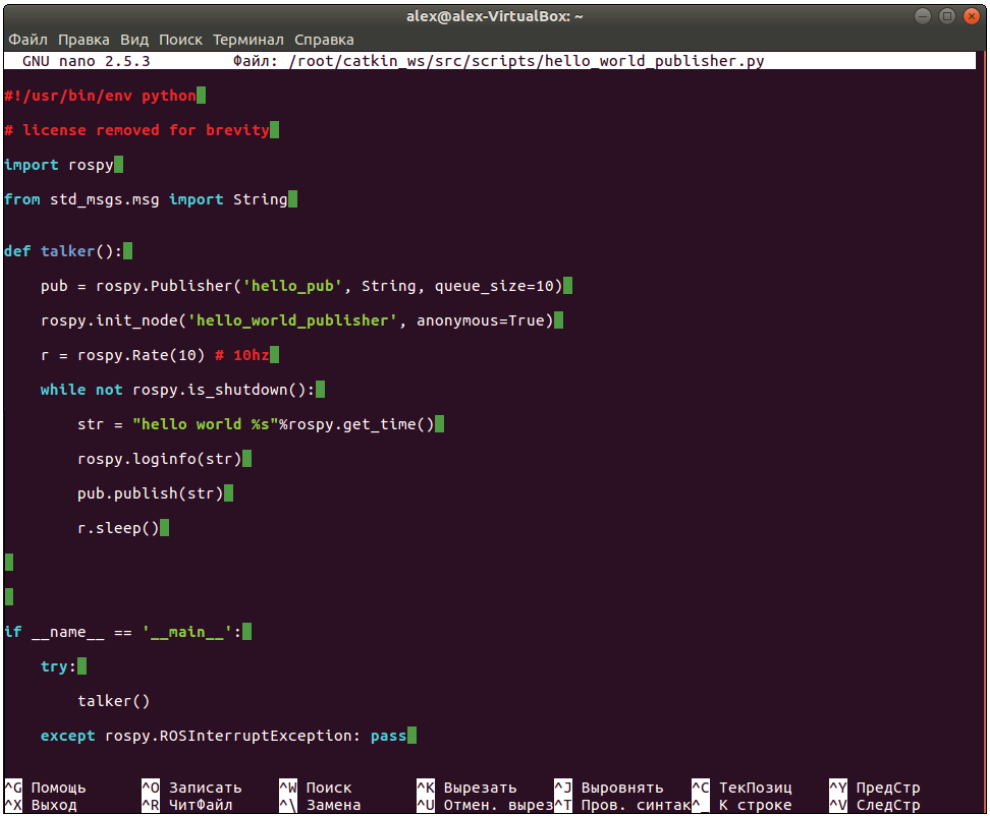
В ответ мы получим список файлов, хранящихся в данной папке:

```
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:

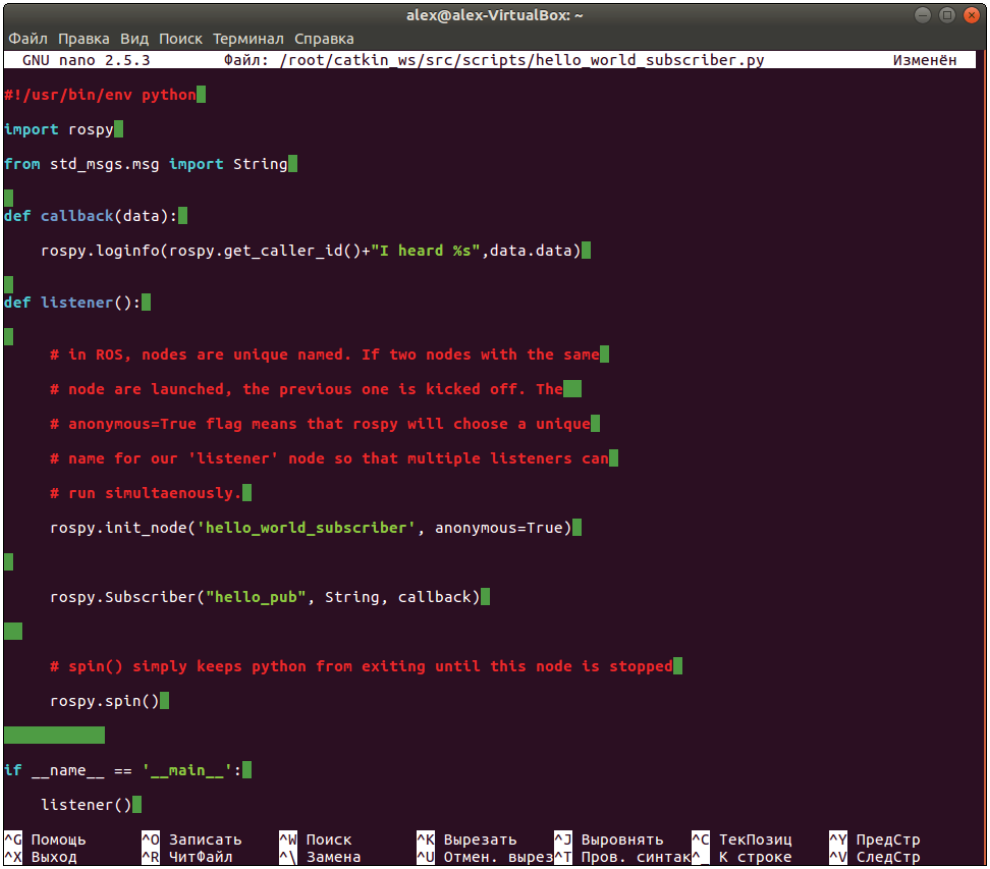



Скрипт `hello_world_publisher.py`

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`.

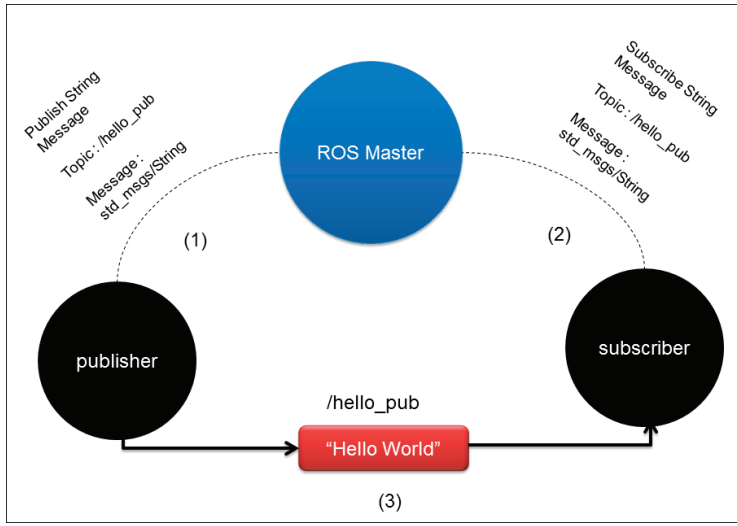
Далее с помощью введенных в Терминал команд `chmod` будут изменены права доступа к исполняемому файлу. Благодаря команде `catkin_make` мы создадим пакет, после чего запустим скрипты на исполнение.

### *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:

ПЕРЕВОД




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
   ```

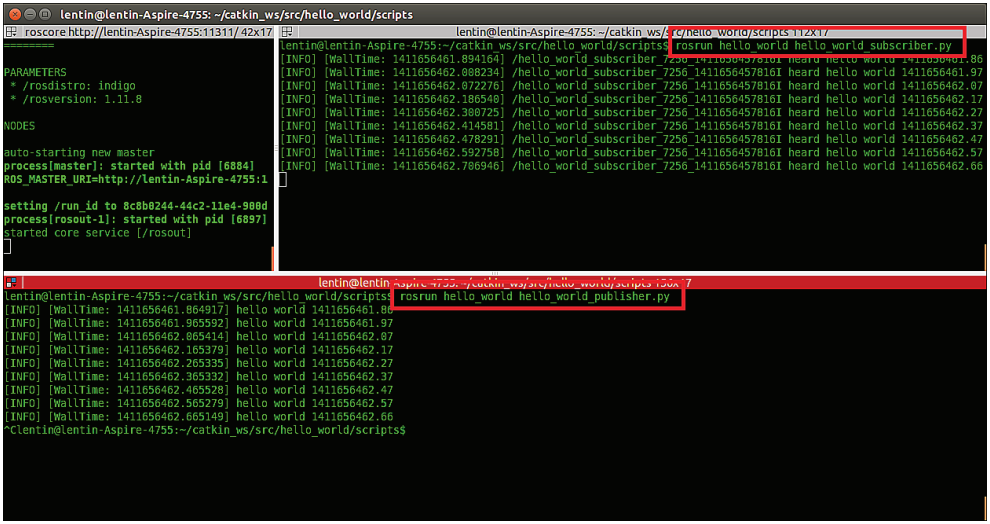
5. После того как права доступа будут изменены, необходимо с помощью команды catkin_make создать пакет:

   ```
   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:

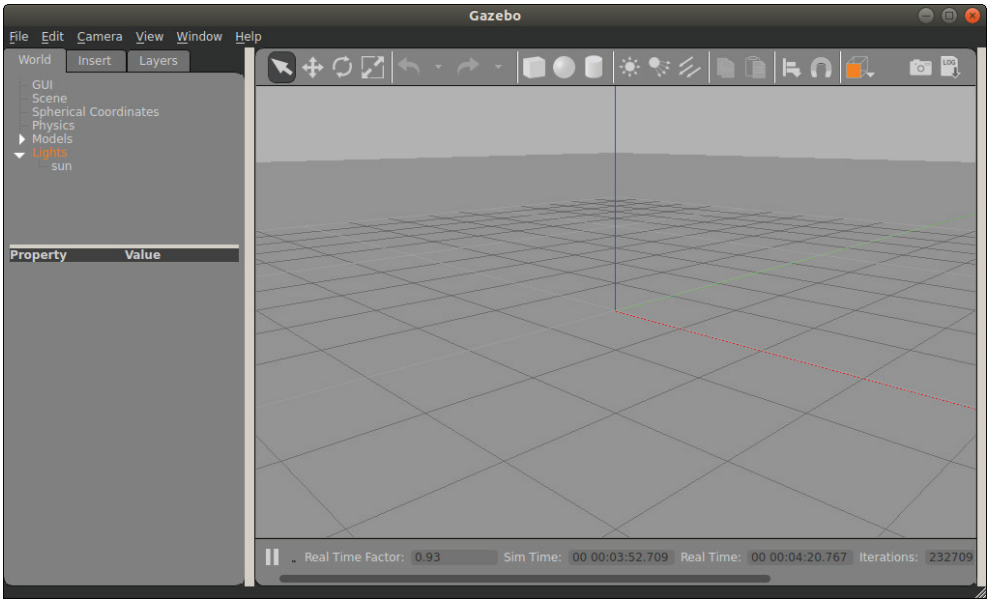
```
$ roscore
```

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?