КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И
ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ
Кафедра интеллектуальной робототехники

Р.О. ЛАВРЕНОВ, Е.А. МАГИД

ОСНОВЫ ROS1 и ROS2

учебно-методическое пособие

Казань
2024

УДК 007.52, 519.878, 519.1, 004.942
ББК 32.81

Рецензент:
кандидат технических наук, доцент кафедры
программной инженерии ИТИС КФУ
В.В. Кугуракова

Лавренов Р.О., Магид Е. А. Основы ROS1 и ROS2: учебно-методическое пособие. – Казань: Казан. ун-т, 2024. - 43 с.

Учебно-методические пособие предназначено для студентов, обучающихся по направлению «Программная инженерия» и профилю подготовки «Интеллектуальная робототехника». Материалы, представленные в пособии, также могут быть полезны преподавателям робототехники. С помощью пособия читатель сможет самостоятельно освоить основные понятия первой и второй версий фреймворка ROS, изучить сторонние программные библиотеки ROS и связи между ними. Кроме того, в пособии описаны средства исследования ROS-процессов как из командной строки, так и с помощью средств с графическим интерфейсом.

Пособие будет полезно для начинающих изучение Робототехнической Операционной Системы ROS и может служить удобной подсказкой для тех, кто уже знаком с ROS.

Содержание

Введение

Программное обеспечение является центральным элементом любого робота. Для обеспечения функциональных возможностей для связи с исполнительными механизмами и датчиками робота чаще всего используются операционные системы. Операционная система на основе Linux может обеспечить большую гибкость взаимодействия с низкоуровневым оборудованием и обеспечить настройку операционной системы в соответствии с конкретными задачами робота. Преимуществами операционной системы Ubuntu в этом контексте являются его отзывчивость, легкость и активная поддержка сообщества. Ubuntu также имеет релизы долгосрочной поддержки (LTS), которые обеспечивают поддержку пользователей на срок до пяти лет. Эти факторы заставили разработчиков ROS ориентироваться на ОС Ubuntu, и это единственная операционная система, которая полностью поддерживается ROS. Комбинация Ubuntu-ROS является идеальным выбором для программирования роботов.

Таким образом первое требование к начинающим изучать ROS — наличие базовых умений работы с linux-системами. В частности, знакомство читателя с терминалом (командной строкой) linux и методами установки программ в linux-системах.

Начнем наше изучение Робототехнической Операционной Системы!

1. Программирование роботов

1.1 Что такое программирование роботов?

Как вы знаете, робот — это киберфизическое устройство с датчиками, исполнительными механизмами (двигателями или, как их еще называют, актуаторами) и вычислительным блоком, который работает на основе пользовательских команд или может принимать свои собственные решения на основе данных с датчиков. Вычислительные операции производит микроконтроллер или персональный компьютер (ПК). Принятие решений и действия робота полностью зависят от программы, управляющей роботом. Управление может осуществляться встроенным программным обеспечением, работающим на микроконтроллере, или кодом C/C++ или Python, выполняющемся на ПК или одноплатном компьютере, например Raspberry Pi. Программирование робота — это процесс написания программы, контролирующей работу робота. На рисунке 1.1 демонстрируется общая схема работы робототехнических устройств.

Основными компонентами любого робота являются исполнительные механизмы и датчики. Приводы перемещают суставы робота, обеспечивая вращательное или линейное движение. Редукторы с сервоприводом, шаговые двигатели и двигатели постоянного тока являются видами исполнительных механизмов. Датчики обеспечивают данные о положении робота и его частей, предоставляют информацию об окружающей среде и внутреннем состоянии робота. Примеры датчиков робота: энкодеры колес, ультразвуковые датчики, лазерные дальномеры и камеры.

Актуаторы (или двигатели) управляются специальными контроллерами и взаимодействуют с микроконтроллером/ПК. Некоторые приводы напрямую управляются через USB-порт ПК. Датчики также взаимодействуют с микроконтроллером или ПК. Ультразвуковые и инфракрасные датчики соединяются с системой управления при помощи специальных контроллеров. Более сложные с точки зрения объема и детализации предоставляемой информации датчики, такие как камеры и лазерные сканеры, могут напрямую взаимодействовать с ПК. Для питания всех роботизированных компонентов обязательно наличия блока питания или аккумулятора. У всех роботов должна присутствовать как минимум од-

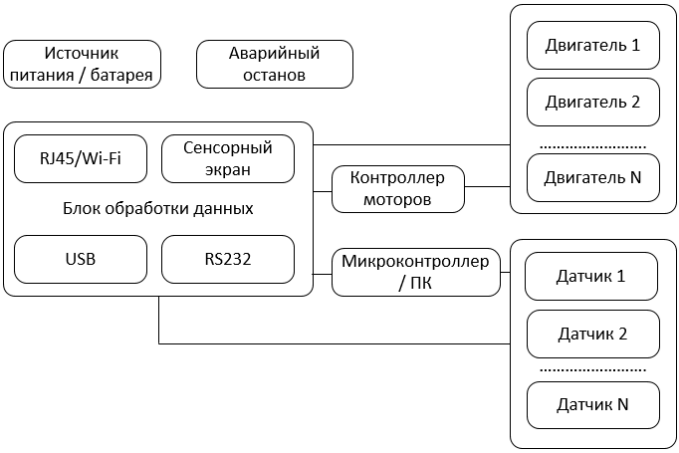

Рис. 1.1: Общая схема работы роботов

на кнопка аварийного останова для немедленной остановки робота. Две основные части робота, используемые для его программирования и расположенные внутри робота, — это ПК и микроконтроллер/ПЛК (программируемый логический контроллер). ПЛК в основном используются в промышленных роботах.

Таким образом, программирование робота — это программирование ПК/микроконтроллера/ПЛК внутри робота для выполнения им определенного действия с использованием исполнительных механизмов и обратной связи от различных датчиков. Для программирования роботов с ПК могут использоваться различные языки программирования: C/C++, Python, Java, C# и другие. Для микроконтроллеров используют Embedded C, язык Wiring (на основе C++), который используется в Arduino, и Mbed (https://os.mbed.com). В приложениях для промышленных роботов используются SCADA или собственные языки программирования, разработанные производителями (например, ABB или KUKA). Программирование робототехнических систем создает программные элементы искусственного интеллекта роботов для самостоятельного принятия решений, автоматизации повторяющихся задач и роботизированного зрения.

1.2 Почему программирование роботов так различается?

Программирование роботов является подмножеством компьютерного программирования. Различия между программированием робота и обычным программированием заключаются в устройствах ввода и вывода. Устройства ввода включают в себя датчики роботов, сенсорные экраны, а устройства вывода включают ЖК-дисплеи и исполнительные механизмы. Любой из языков программирования может использоваться для роботов, но C++ и Python являются наиболее часто используемыми из-за высокой производительности и меньшего времени, необходимого для создания прототипов программ.

Ниже приведены некоторые характеристики, важные при программировании робота.

- *Многопоточность:* Как видно на схеме (рисунок 1.1), в составе роботехнической системы может быть несколько датчиков и исполнительных механизмов. Следовательно, понадобится многопоточный язык программирования для работы с разными датчиками в разных потоках. Потоки могут общаться друг с другом для обмена данными.

- *Высокоуровневое объектно-ориентированное программирование:* Все объектно-ориентированные языки программирования (ООП) являются модульными, и качественный код можно легко использовать повторно. Поддержка кода также гораздо легче по сравнению языками, не поддерживающими ООП.

- *Низкоуровневое управление устройствами:* Языки программирования высокого уровня могут также получать доступ к устройствам низкого уровня, таким как пины GPIO (универсальные порты ввода/вывода), последовательные порты, параллельные порты, USB, SPI и I2C. Языки программирования, такие как C/C++ и Python, могут работать с устройствами низкого уровня, поэтому эти языки предпочтительнее на одноплатных компьютерах (например, Raspberry Pi или Odroid).

- *Простое прототипирование:* Простота создания прототипа алгоритма робота, безусловно, является аргументом при выборе языка программирования. Python — хороший выбор для быстрого прототипирования алгоритмов роботов.

- *Межпроцессное взаимодействие:* У типового робота имеется несколько датчиков и исполнительных механизмов. Мы можем использовать многопоточную архитектуру или написать независимую программу для выполнения каждой задачи; например, одна програм-

ма берет изображения с камеры и обнаруживает лицо, а другая программа отправляет данные на встроенную плату. Эти две программы могут общаться друг с другом для обмена данными.

- *Производительность:* Если мы работаем с датчиками с высокой пропускной способностью, такими как камеры глубины и лазерные сканеры, вычислительные ресурсы, необходимые для обработки получаемых с них данных, очевидно, высоки. Некоторые языки программирования позволяют напрямую контролировать работу с памятью (выделять и загружать необходимые объемы), и язык C++ — хороший выбор для обработки подобных сценариев.

- *Поддержка сообщества:* При выборе языка для программирования роботов убедитесь, что для этого языка достаточно поддержки сообщества, включая форумы и блоги.

- *Наличие сторонних библиотек:* Наличие сторонних библиотек может существенно облегчить вашу разработку; например, для обработки изображений можно использовать готовые библиотеки, такие как OpenCV. Если ваш язык программирования имеет поддержку OpenCV, вам будет проще создавать приложения для обработки изображений.

- *Поддержка существующего программного обеспечения для робототехники:* Существуют уже готовые и достаточно популярные программные среды для программирования роботов и их виртуальных моделей, такие как ROS. Если ваш язык программирования имеет поддержку ROS, создать прототип нового приложения для робота будет намного проще.

2. Робототехническая операционная система (ROS)

2.1 Что такое ROS?

Робототехническая операционная система (ROS) — это свободно распространяемый фреймворк с открытым исходным кодом для робототехники, который используется как в коммерческих, так и в исследовательских приложениях [1]. Платформа ROS предоставляет следующие возможности программирования роботов:

- *Интерфейс передачи сообщений между процессами.* ROS обеспечивает интерфейс передачи сообщений для связи между двумя программами или процессами. Как уже упоминалось, это одна из ключевых функций, необходимых для программирования робота.

- *Элементы операционной системы.* ROS не является реальной операционной системой: это фреймворк с некоторым функционалом операционной системы, включая многопоточность, низкоуровневое управление устройствами, управление пакетами и аппаратную абстракцию. Управление пакетами помогает пользователям организовывать программное обеспечение в единицах, называемых пакетами. Каждый пакет имеет исходный код, файлы конфигурации или файлы данных для конкретной задачи. Эти пакеты могут быть установлены на других компьютерах, имеющих необходимое программно-аппаратное обеспечение.

- *Поддержка языков программирования высокого уровня.* Преимущество ROS состоит в том, что данная система поддерживает популярные языки программирования, используемые в программировании роботов, включая C, Python и Lisp. Существует экспериментальная поддержка языков C#, Java, Node.js и некоторых других. С полным списком поддерживаемых языков можно ознакомиться по адресу http://wiki.ros.org/. ROS предоставляет клиентские библиотеки для этих языков. Например, если программист хочет реализо-

вать приложение Android, использующее функциональность ROS, можно использовать клиентскую библиотеку rosjava.

- *Наличие сторонних библиотек.* Платформа ROS интегрирована с большинством популярных сторонних библиотек; например, библиотека OpenCV (https://opencv.org) интегрирована для машинного зрения, а PCL (http://pointclouds.org) интегрирована для реконструкции и манипуляции с 3D окружением робота.

- *Готовые алгоритмы.* В ROS реализованы многие популярные робототехнические алгоритмы, например, управление при помощи PID-контроллеров (wiki.ros.org/pid); алгоритмы одновременной локализации и картографирования (SLAM) (wiki.ros.org/gmapping); планировщики пути, включая алгоритмы поиска по графу A и Dijkstra (wiki.ros.org/global_planner); адаптивная локализация Монте-Карло AMCL (wiki.ros.org/amcl). Готовые алгоритмы позволяют существенно сократить время разработки робототехнических программ.

- *Поддержка сообщества.* Пользователи ROS во всем мире активно разрабатывают и поддерживают пакеты ROS. Большая поддержка сообщества включает разработчиков, задающих вопросы, связанные с ROS: ROS Answers — это платформа для вопросов о ROS (answers.ros.org/questions/); ROS Discourse — это онлайн-форум, на котором пользователи ROS обсуждают различные темы и публикуют связанные с ROS новости (discourse.ros.org).

- *Утилиты и среды моделирования.* ROS состоит из множества инструментов, работающих как из командной строки, так и с графическим интерфейсом, для отладки, визуализации и моделирования робототехнических приложений. Например, инструмент Rviz (wiki.ros.org/rviz) используется для визуализации данных с камер, лазерных сканеров, инерциальных датчиков, камер глубины и других датчиков. Для моделирования окружающей среды, в которой работает робот, используют симуляторы, например, Gazebo (gazebosim.org).

До появления ROS не было единой платформы и сообщества для разработки робототехнических приложений: каждый разработчик создавал программное обеспечение для своего собственного робота, которое в большинстве случаев не могло быть использовано для какого-либо другого робота. ROS фактически установила новые принципы программирования робототехнических систем и стала общей платформой разработки для робототехнических приложений. Исходный код является бесплатным и открытым для коммерческих и исследовательских целей [2].

2.2 История возникновения и развития ROS. Появление ROS2.

За последние года ROS вырос и вырос в несколько раз. В настоящий момент он имеет огромный список пакетов, где каждый пакет решает проблему либо частично, либо полностью, что исключает концепцию изобретения колеса заново. Созданные сообществом пакеты привели к тому, что был создан совершенно новый подход к робототехнике и предложена интеллектуальная начинка для существующих систем.

Ниже приведены исторические вехи проекта ROS (ROS1).

- Проект ROS был запущен в Стэнфордском университете в 2007 году под руководством Моргана Куигли. Сначала это был набор программного обеспечения, разработанный для роботов в Стэнфорде.
- Позже, в 2007 году, стартап по исследованию робототехники под названием Willow Garage взял на себя проект и придумал название ROS, что означает «Робототехническая операционная система».
- В 2010 году была выпущена ROS 1.0. Многие из его функций все еще используются.
- В 2012 году ROS переходит к OSRF (Open Source Robotics Foundation).
- В 2014 году был выпущен ROS Indigo Igloo (восьмой релиз), это был первый выпуск с долгосрочной поддержкой (LTS).
- В 2016 году была выпущена ROS Kinetic Kame. Вторая LTS версия ROS. Начиная с этой версии ROS поделился на ROS1 и ROS2.
- В мае 2018 года была выпущена третья версия ROS1 с долгосрочной поддержкой - ROS Melodic Morenia.
- В мае 2020 года была выпущена последняя версия ROS1 - ROS Noetic Ninjemys которая имеет поддержку до мая 2025 года.

Несмотря на то, что ROS1 дал определенную свободу общения со сложными аппаратными средствами и программными компонентами, имеются некоторые сложности, связанные с использованием ROS1 в конечном продукте. Данные сложности начинают появляться, когда существует целый парк разнородных роботов (мобильные роботы, роботы-манипуляторы и так далее). Установить связь между ними довольно сложно из-за того, что существуют архитектурные ограничения ROS1. Первое ограничение состоит в том, что ROS1 строго централизован и не поддерживает концепцию мультимастера. Второе ограничение ROS1 заключается в том, что, нет поддержки шифрования сообщений, которыми обмениваются программы, написанные с использованием ROS1. Любой,

кто имеет возможность подключиться к мастеру (master-node), может получить доступ к топикам и далее либо эксплуатировать или изменять их. Поэтому основное предназначение ROS1 — проверка концепции или прототипирование для научных целей, а также для обучения. Третье ограничение ROS1 — данная система не является real-time системой. Возможны задержки в получении данных и их потеря. И, наконец еще одно важное ограничение — это то, что ROS1 фактически мог использоваться только на операционной системе Ubuntu. Все вышеизложенные недостатки подтолкнули к созданию ROS2.

ROS2 находится в стадии активной разработки. На настоящее время (начало 2024 года) его история такова:

- В августе 2015 года был открыт исходный код альфа-версии ROS2.
- В декабре 2017 года была выпущена первая полноценная версия — ROS2 Ardent Apalone.
- В июне 2020 года вышла первая версия с долгосрочной поддержкой — ROS2 Foxy Fitzroy. Поддержка этой версии завершилась в 2023 году.
- В 2022 году была выпущена вторая LTS версия — ROS2 Humble Hawksbill с поддержкой до мая 2027 года.
- Следующая LTS версия выходит в мае 2024 года с поддержкой до 2029 года — ROS2 Jazzy Jalisco.

Версии ROS1 и ROS2 и более подробную историю можно найти на сайте docs.ros.org. Каждая версия ROS называется дистрибутивом ROS. Если вы ищете новейшие функции ROS, вы можете выбрать новые дистрибутивы, а если вы ищете стабильные пакеты, вы можете выбрать LTS. В данном пособии примеры ROS1 используют версию Noetic Ninjemys. В настоящее время обе версии фреймворка ROS разрабатываются и поддерживаются Open Robotics, ранее известной как Open Source Robotics Foundation.

2.3 Сообщество ROS

Ниже приведены ресурсы, используемые для обмена программным обеспечением и утилитами ROS.

- В ROS wiki есть учебники по настройке и программированию ROS.
- Ответы ROS (https://answers.ros.org/questions/) содержат вопросы и решения по проблемам ROS, аналогично ресурсу Stack Overflow.

- Дискурс ROS (https://discourse.ros.org) — это форум, на котором разработчики могут делиться новостями и задавать вопросы, связанные с ROS.
- Российское сообщество ROS в Телеграм (https://t.me/rosrussia/1) - место сбора всех ROS разработчиков России. Тут происходит обмен новостями, отвечают на вопросы, предлагают вакансии.

2.4 Установка ROS1

Установить ROS1 на ПК несложно. Перед установкой вы должны знать о различных платформах, которые поддерживают ROS1. На рисунке 2.1 показаны различные операционные системы, в которых вы можете установить ROS1. Как уже говорилось, ROS1 является фреймворком, и для его работы нужна операционная система.

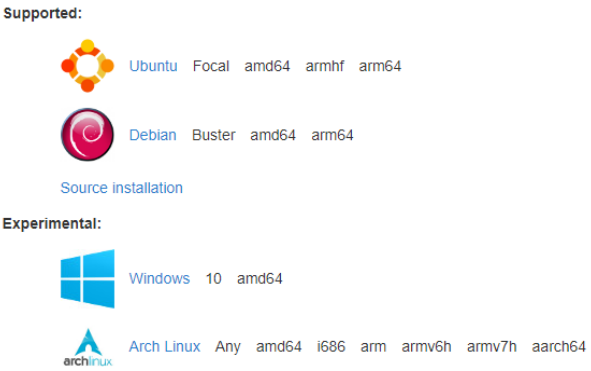

Рис. 2.1: Операционные системы, которые поддерживают ROS1.

Ubuntu Linux является наиболее предпочтительной ОС для установки ROS1. Как вы можете видеть на рис. 2.1, ROS1 поддерживает 32-битную, 64-битную Ubuntu, 32-битную ARM и 64-битную ARM. ROS1 может работать на ПК и на одноплатных компьютерах, таких как Raspberry, Odroid и NVIDIA TX1/TX2. Debian Linux также поддерживает ROS. В OS-X и других операционных системах ROS1 находится в экспериментальной фазе, что означает, что функциональные возможности ROS1 не доступны.

Давайте перейдем к установке. Если вы используете ПК или плату ARM, на которой работает Ubuntu armhf или arm64, вы можете выполнить процедуры со страницы: wiki.ros.org/ROS/Installation. Когда вы заходите в эту вики, она спрашивает, какую версию ROS вам нужно уста-

новить. В настоящее время (начало 2024г.) доступна только ROS Noetic Ninjemys.

Мы можем установить ROS1 двумя способами: с помощью готовых бинарных файлов или путем компиляции исходного кода. В этом пособии мы устанавливаем ROS Noetic с помощью готовых бинарных файлов на Ubuntu 20.04 LTS. На другие дистрибутивы Ubuntu данная версия ROS может быть установлена только компиляцией исходного кода.

Ниже описаны этапы установки.

1. Настройка файла *sources.list*. На этом шаге мы добавляем в пакетный менеджер ОС информацию о хранилище ROS, где хранятся двоичные файлы. Для этого выполните следующую команду в терминале.

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

   Эта команда создает новый файл /etc/apt/sources.list.d/ros-latest.list и добавляет к нему следующую строку.
   "deb http://packages.ros.org/ros/ubuntu xenial main". Обратите внимание, что если вы выполните

   ```
   lsb_release -sc
   ```

   в терминале, вы получите вывод 'focal'.

2. Добавление ключей. В Ubuntu, если мы хотим загрузить двоичный файл или пакет, мы должны добавить безопасный ключ в нашу систему, чтобы аутентифицировать процесс загрузки. Пакет, который аутентифицируется с использованием этих ключей, является доверенным. Проверкой ключей занимается утилита curl. Если эта утилита не установлена, то её нужно установить.

   ```
   sudo apt install curl
   ```

3. Далее с помощью этой утилиты взять по ссылке ключ на список дистрибутивов ROS и сохранить его.

   ```
   curl -s
   https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc |
   sudo apt-key add -
   ```

4. Обновление списка доступных пакетов Ubuntu. Делается это с помощью команды *update* программы *apt*, которая контролирует пакеты в Ubuntu.

sudo apt update

5. Устанавливаем пакеты ROS Noetic, используя следующую команду.

   sudo apt install ros-noetic-desktop-full

   Эта команда устанавливает все необходимые пакеты в ROS, включая необходимые для работы библиотеки, симуляторы и основные алгоритмы для роботов. Для загрузки и установки всех этих пакетов требуется время.

6. После установки всех пакетов нам нужно установить инструмент под названием *rosdep*, который полезен для установки зависимых пакетов пакета ROS. Например, какой-нибудь пакет ROS может иметь несколько зависимых пакетов для правильной работы. rosdep проверяет, доступны ли зависимые пакеты, и, если нет, автоматически устанавливает их. Следующие две команды устанавливают rosdep и обновляют его конфигурацию.

   sudo rosdep init
   rosdep update

7. Настройка окружения ROS. Как обсуждалось ранее, ROS поставляется с необходимыми утилитами и библиотеками. Чтобы получить доступ к этим инструментам и пакетам командной строки, нам нужно настроить среду ROS. Следующая команда добавляет строку в файл *.bashrc* в вашей домашней папке, которая подключает среду ROS в каждом новом окне терминала.

   echo "source /opt/ros/noetic/setup.bash»> /.bashrc

   Затем введите следующую команду, чтобы добавить среду ROS в текущее окно терминала.

   source /.bashrc

   Установка завершена.

8. Установка дополнительных пакетов, полезных для сборки остальных пакетов.

   sudo apt install python3-rosdep python3-rosinstall
   python3-rosinstall-generator python3-wstool build-essential

   Поздравляем, вы закончили с установкой. Вы можете проверить правильность установки, используя следующую команду.

   rosversion -d

   Если в качестве вывода будет слово *'noetic'* то и установка и настройка прошли успешно.

2.5 Установка ROS2

Перед установкой вы должны знать о различных платформах, которые поддерживают ROS2. Как уже говорилось, ROS2 не является операционной системой, но для работы ей нужна операционная система. Установить ROS2 можно двумя способами: из бинарных установочных файлов или из исходного кода.

Последняя (на текущий момент) версия ROS2 - ROS 2 Humble Hawksbill. Установить из бинарных файлов можно на операционных системах Windows (если есть VS 2019), Ubuntu 22.04, RedHat 8. Если же примите решение собирать из файлов с исходным кодом, то, кроме этих операционных систем, можете также использовать MacOS.

В этом пособии мы устанавливаем ROS 2 Humble Hawksbill с помощью готовых бинарных файлов на Ubuntu 22.04 LTS.

Ниже описаны этапы установки.

1. Установим пакет для контроля свойств программного обеспечения

   sudo apt install software-properties-common

2. Убедимся, что репозиторий Ubuntu Universe включен в пакетный менеджер операционной системы.

   sudo add-apt-repository universe

3. Добавление ключей. В Ubuntu, если мы хотим загрузить двоичный файл или пакет, мы должны добавить безопасный ключ в нашу систему, чтобы аутентифицировать процесс загрузки. Пакет, который аутентифицируется с использованием этих ключей, является доверенным. Проверкой ключей занимается утилита curl. Если эта утилита не установлена, то её нужно установить.

   sudo apt install curl

4. Далее с помощью этой утилиты взять по ссылке ключ на список дистрибутивов ROS и сохранить его.

   sudo curl -sSL
   https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
   /usr/share/keyrings/ros-archive-keyring.gpg

5. Добавим репозиторий в список источников

   echo "deb [arch=$(dpkg –print-architecture)
   signed-by=/usr/share/keyrings/ros-archive-keyring.gpg]
   http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
   $UBUNTU_CODENAME) main sudo tee
   /etc/apt/sources.list.d/ros2.list > /dev/null

Эта команда добавит ссылку *packages.ros.org/ros2/ubuntu* в список мест, откуда пакетный менеджер будет брать обновления для пакетов.

6. Обновление списка доступных пакетов Ubuntu. Делается это с помощью команды *update* программы *apt*, которая контролирует пакеты в Ubuntu. Команда *upgrade* обновит все имеющиеся в операционной системе программные пакеты.

   sudo apt update && sudo apt upgrade

7. Устанавливаем пакеты ROS 2 Humble Hawksbill, используя следующую команду.

   sudo apt install ros-humble-desktop

   Эта команда устанавливает все необходимые пакеты в ROS2, включая необходимые для работы библиотеки, симуляторы и основные алгоритмы для роботов. Для загрузки и установки всех этих пакетов требуется время.

8. Устанавливаем дополнительные пакеты для программирования во фреймворке ROS2.

   sudo apt install ros-dev-tools

9. Настройка окружения ROS2. Как обсуждалось ранее, ROS2 поставляется с необходимыми утилитами и библиотеками. Чтобы получить доступ к этим инструментам и пакетам командной строки, нам нужно настроить среду ROS2. Следующая команда добавляет строку в файл *.bashrc* в вашей домашней папке, которая подключает среду ROS2 в каждом новом окне терминала.

   echo "source /opt/ros/humble/setup.bash»> /.bashrc

   Затем введите следующую команду, чтобы добавить среду ROS в текущее окно терминала.

   source /.bashrc

   Установка завершена.

Поздравляем, вы закончили с установкой. Вы можете проверить правильность установки, запустив в одном окне терминала

ros2 run demo_nodes_cpp talker

а в другом

ros2 run demo_nodes_py listener

Если в качестве вывода во втором окне будут отображаться сообщения из первого окна, то это означает, что установка прошла успешно.

3. Архитектура ROS

3.1 Основные понятия ROS

Робот может иметь множество датчиков и исполнительных механизмов. Как мы можем управлять несколькими приводами и обрабатывать данные с большого количества датчиков? Лучший выход, написать небольшие независимые программы для обработки данных датчиков и управления приводами, и обмен данными между этими программами. Именно в такой ситуации нужно использовать ROS.

По сути, фреймворк ROS является основой для взаимодействия между двумя или более программами или процессами. Например, если программа A хочет отправить данные в программу B, а B хочет отправить данные в программу A, мы можем легко реализовать это с помощью ROS.

Термины, которые используются в Робототехнической Операционной Системе:

- *Nodes – ноды.* Программа ROS, выполняющая какие-либо действия (обработка данных с датчиков, отправка данных в другие ноды, и т.д.), в которой используется API ROS, собирается из одного или нескольких файлов с исходным кодом. Является базовым элементом ROS. Могут запускаться независимо друг от друга.

- *Topic – топик или тема.* Именованный поток данных, в которых передаются данные определенного типа (или структуры). Ноды ROS могут публиковать сообщения в топики или подписываться на них для получения данных. Ноды могут публиковать или подписываться на любое количество топиков.

- *Message – сообщение.* Сообщения передаются через топики. Они могут быть примитивными типами данных или сложными структурами данных. Пользователи могут формировать собственные структуры для отправки их в качестве сообщений.

- *Service – сервис.* Реализация в ROS механизма "запрос/ответ" (request/responce). Нода, которая реализует работу сервиса, назы-

вается сервером, а нода, которая вызывает сервис, называется клиентом. Выполнение сервисной функции осуществляется только по запросу от клиентской ноды.

- *ROS Master – ROS-Мастер.* Первоначально запускаемая программа, которая соединяет ноды ROS, для межпроцессорного взаимодействия между ними (только в ROS1).

- *Parameter server — сервер параметров.* Программа, которая запускается вместе с ROS-Мастером. Пользователь может хранить различные параметры или значения на этом сервере, и все ноды могут получить к нему доступ. Пользователь также может установить конфиденциальность параметров. Если это публичный (public) параметр, все ноды имеют к нему доступ; если это частный (private), то только конкретная нода будет иметь доступ к параметру (только в ROS1).

Давайте посмотрим, как происходит связь между двумя программами (нодами) в ROS1. Рисунок 3.1 иллюстрирует базовую блок-схему ROS1.

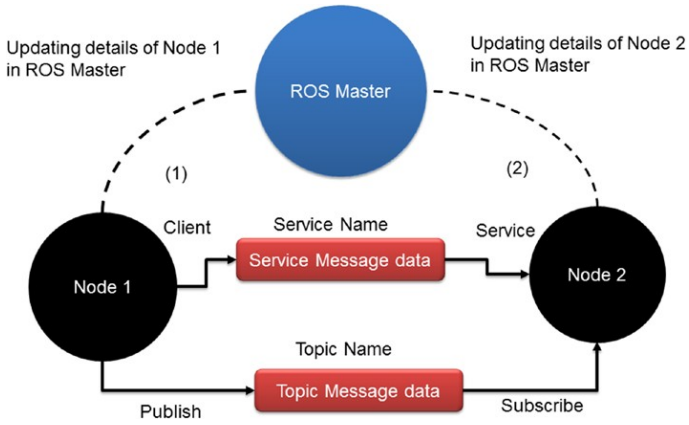

Рис. 3.1: Базовая блок-схема ROS1.

На рисунке 3.1 показаны две программы, помеченные как Нода 1 и Нода 2. Нода (node — с англ. узел) — название программы в ROS. Когда запускается любая из программ, нода связывается с программой ROS1, называемой ROS-Мастер (ROS Master). Нода отправляет всю свою информацию ROS-Мастеру, включая типы данных, которые она отправляет или получает. Ноды, которые отправляют данные, называются публикующими (publish — с англ. издавать, выпускать), а ноды, которые

принимают данные, называются подписчиками (subscribe — с англ. подписаться). ROS-Мастер имеет всю информацию о запущенных нодах, что они публикуют и на что подписаны. Если нода 1 отправляет конкретные данные, называемые «A», и ноде 2 требуются те же данные, то ROS-Мастер отправляет информацию нодам, чтобы они могли обмениваться данными друг с другом.

Ноды ROS могут отправлять друг другу разные типы и структуры данных, которые включают в себя примитивные типы данных, такие как целое число, число с плавающей запятой, строки и т.д. Различные типы отправляемых данных называются сообщениями (messages — с англ. сообщения) ROS. С помощью сообщений ROS мы можем отправлять данные одного типа или данные одной конкретной структуры. Эти сообщения отправляются через поток сообщений, называемый топиком ROS (topic — с англ. тема). У каждого топика есть имя. Например, в топике с именем «cmd_vel» отправляются команды скорости на автономное устройство.

На рисунке 3.1 нода 1 публикует в топик, а нода 2 подписана на него. За взаимообмен данными между нодами в первой версии фреймворка ROS отвечает ROS-Мастер.

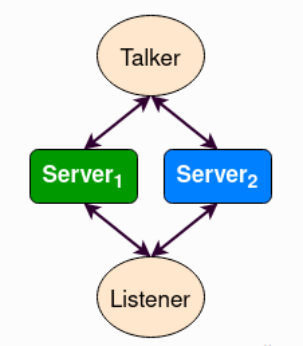

Рис. 3.2: Блок-схема ROS2 с несколькими DDS v2 серверами.

В ROS2 общие принципы работы остались прежними. Есть программы (ROS-ноды или узлы) и есть каналы связи (ROS-топики), с помощью которых они обмениваются данными. Отличие от ROS1 заключается в том, что обязательно необходимой программы ROS-мастер в ROS2 нет, как и сервера параметров, который был частью ROS-Мастера. Для обнаружения нодами друг друга и соединения их топиками используется сервер, реализующий сервис Fast DDS v2 [6]. Клиенты (узлы ROS) могут подключаться к любому количеству серверов, что позволяет иметь резервную сеть, которая будет работать, даже если некоторые серверы

или узлы неожиданно отключатся. На рисунке 3.2 показана простая архитектура с двумя нодами (Talker и Listener) и двуми обслуживающими их серверами.

3.2 Файловая система ROS

Основная единица файловой системы ROS - это пакеты [3]. Пакеты ROS — это отдельные единицы или атомные единицы программного обеспечения ROS. Весь исходный код, файлы данных, файлы сборки, зависимости и другие файлы организованы в пакеты. Метапакет ROS группирует набор похожих пакетов для конкретного приложения. В метапакете ROS нет исходных файлов или файлов данных. Он может иметь зависимости от других ROS-пакетов.

Приведу пример. Есть пакет turtlebot3-description [4], в котором хранится кинематическая схема робота Turtlebot3. Есть еще один пакет turtlebot3-navigation, в котором хранятся необходимые файлы для самостоятельной навигации этого робота. Вместе с еще несколькими пакетами для этого робота они образуют метапакет. Один или несколько метапакетов могут лежать в репозитории. ROS репозиторий — это набор пакетов ROS, которые используют общую систему контроля версий.

3.2.1 Пакеты в ROS1

Файл описания пакета — это файл XML, размещенный внутри пакета ROS. Он содержит всю основную информацию о пакете ROS, включая имя пакета, описание, автора, зависимости и так далее. Типичный *package.xml* показан ниже.

```
<?xml version="1.0"?>
<package>
<name>test_pkg</name>
<version>0.0.1</version>
<description>The test package</description>
<maintainer email="lirs_itis_kfu@gmail.com">robot</maintainer>
<license>BSD</license>
<buildtool_depend>catkin</buildtool_depend>
.................. ............... .
<exec_depend>catkin</run_depend>
....... . ............ .
</package>
```

Внутри каждого пакета ROS1 обычно находится несколько директорий. В директории *scripts* хранятся python файлы. В директории *src* хранятся файлы исходного кода c++, в *include* — заголовочные файлы C++. В папке *msg* пользователи могут написать собственный тип

сообщений, который будет передаваться в топиках. А в директории *srv* — собственный сервис. Кроме того, пользователь может создать папку *launch* в которой будут лежать файлы запуска нод (.launch файлы) с определенными, прописанными параметрами. В файле *CMakeLists.txt*, обязательно имеющемся в каждом пакете, находятся настройки сборки пакета, а также указываются все необходимые для сборки зависимости. На рисунке 3.3 демонстрируется пример содержимого пакета ROS1.

Рис. 3.3: Структура пакета ROS1.

Файлы запуска (.launch файлы) в ROS1 представляют собой xml файлы, в которых перечислены запускаемые ноды и их параметры. Внутри .launch файла также можно указать использование других .launch файлов. Однако, нельзя задать какое-либо условие запуска или запуск в цикле. Поэтому в ROS2 .launch файлы были существенно переработаны.

3.2.2 Пакеты в ROS2

Пакеты в ROS2 немного отличаются от пакетов из ROS1 [5]. Для разработки в фреймворке ROS2 используется только два языка программирования: C++ и Python. Соответственно, пакеты делятся на те, в которых будут программы, написанные на C++, и те, где используется Python.

Пакеты, где используется C++ обычно содержат следующие директории и файлы:

- Файл *package.xml*, содержащий метаинформацию о пакете.
- Файл *CMakeLists.txt*, описывающий, как создать код внутри пакета.
- Каталог *include/<имя_пакета>*, содержащий общедоступные заголовочные файлы.
- Каталог *src*, содержащий файлы исходного кода пакета

Структура файла *CMakeLists.txt* идентична тем, что используются в ROS1.

Пакеты, где используется Python содержат другие директории и файлы:

- Файл *package.xml*, содержащий метаинформацию о пакете.
- Файл *resource/<package_name>*, в котором хранятся настройки использования ресурсов.
- Файл *setup.cfg* используется, когда в пакете есть исполняемые файлы, чтобы командой *ros2 run* можно было их запускать.
- Файл *setup.py*, содержащий инструкции по установке пакета.
- Каталог *<имя_пакета>*, содержащий файлы исходного кода на языке Python, в том числе обязательный файл *__init__.py*.

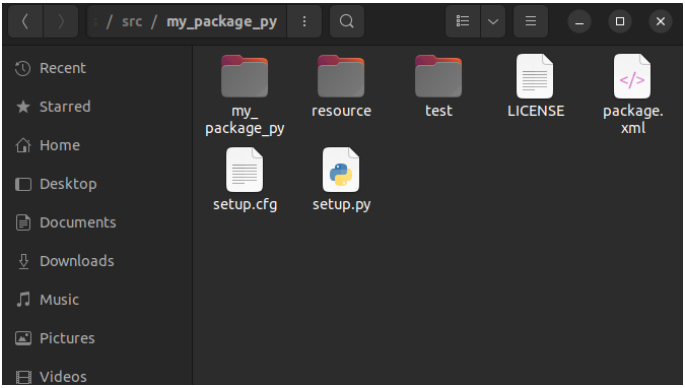

Рис. 3.4: Структура Python пакета ROS2.

Структура файла *package.xml* в C++ и в Python идентична тем, что используются в ROS1.

Также в C++ и Python пакетах ROS2 могут присутствовать файлы запуска. Они как в ROS1 могут запускать несколько нод, задавать их параметры и подключать другие файлы запуска. Однако, в отличии от ROS1, теперь это Python файлы с расширением *.launch.py*. В ROS2 можно задавать условие запуска нод или запускать ноды в цикле. Более гибкие файлы запуска - это наиболее важное изменение ROS2 по сравнению с ROS1.

На рисунке 3.4 демонстрируется пример содержимого пакета ROS2. В данном случае есть еще необязательные каталог с тестами для кода и файл лицензии.

4. Базовое использование ROS

4.1 Инструменты командной строки

В этом разделе рассматриваются инструменты (или команды) командной строки ROS. Существуют различные команды ROS, которые мы можем использовать для исследования различных аспектов ROS. Мы можем реализовать практически все возможности ROS с помощью этих команд. Инструменты командной строки выполняются в терминале Linux.

4.1.1 Консольные команды ROS1

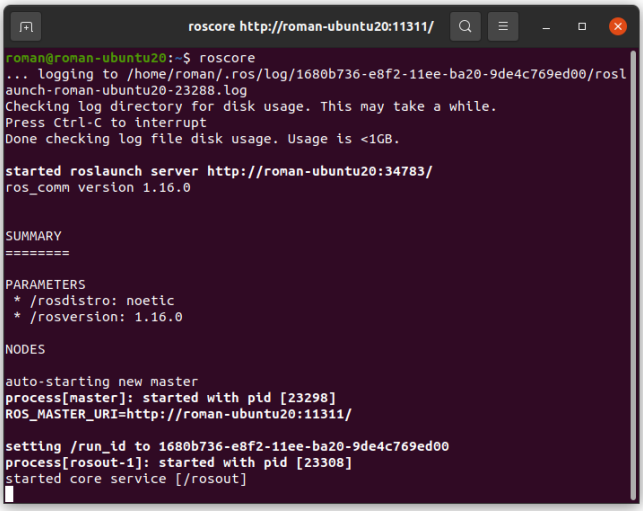

Рис. 4.1: Окно терминала после вызова *roscore*.

1. Команда ***roscore*** — очень важный инструмент в ROS. Когда мы запускаем эту команду в терминале, она запускает ROS-Мастер, сервер параметров и ноду ведения записей (логирования — logging). Мы можем запустить любую другую ноду ROS после выполнения этой команды. Если вы запустите *roscore* в окне терминала, вы можете получить сообщения, подобные тем, которые показаны на рисунке 4.1. В терминале вы можете видеть сообщения о запуске ROS-Мастера и сервера параметров. Также можно увидеть основной адрес ROS для удаленного подключения.

2. С помощью команды ***rosnode*** можно исследовать все свойства и настройки ноды ROS. Обратите внимание, что при выполнении этой и последующих команд, в отдельном окне должен быть запущен ROS-Мастер (*roscore*).

   rosnode list – вывод списка нод ROS, запущенных на текущий момент.

   rosnode info /NodeName – получение информации о ноде: на какой топик подписана нода и в какой публикует, какие у нее есть сервисы и с какими нодами она связана. Ниже, на рисунке 4.2 приведен пример получения информации о ноде.

   rosnode ping /NodeName – проверить соединение с нодой *NodeName*.

   rosnode kill /NodeName – закрытие ноды *NodeName* и очистка всей памяти выделенной на неё.

3. Команда ***rostopic*** предоставляет информацию о топиках, которые публикуются или на которые подписаны ноды в текущий момент. Эта команда очень полезна для анализа существующих топиков, просмотра данных в них, и публикации вручную данных в топики.

   rostopic list – вывод списка всех существующих в системе топиков.

   rostopic echo /TopicName – вывод в консоль данных с топика *TopicName*.

   rostopic type /TopicName – вывод в консоль информацию о типе данных, передаваемого по топику *TopicName*.

   rostopic find /TypeName – поиск топиков в которые публикуются сообщения типа *TypeName*.

   rostopic pub TopicName msg_ type Data – публикация данных (*Data*), имеющих тип *msg_ type* в топик *TopicName*.

4. Инструменты ***rosmsg*** и ***rossrv*** предоставляют пользователю информацию о сообщениях и сервисах, соответственно. Набор команд у них одинаковый и приведен далее на примере *rosmsg*:

   rosmsg show MsgName – вывод информации о типе с названием *MsgName*, если это структура, то выведется содержимое.

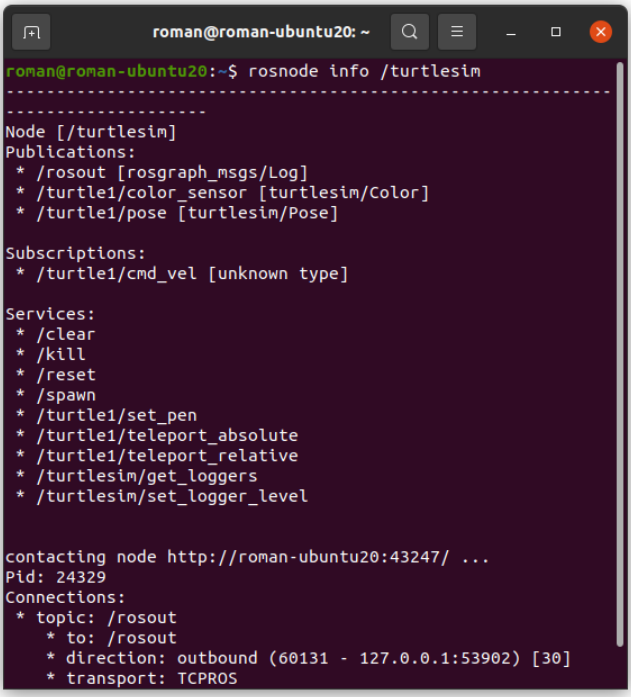

Рис. 4.2: Окно терминала после вызова *rosnode info*.

rosmsg package PckgName – вывод информации о всех используемых типах в пакете с названием *PckgName*.

5. Кроме того, для исследования сервисов есть специальный инструмент ***rosservice***. Примеры его использования на рисунке 4.3.

   rosservice list – информация о всех, доступных для вызова сервисах.

   rosservice node /SrvName – вывод названия ноды, которая осесчивает работу сервиса с названием *SrvName*.

   rosservice call /SrvName ... – вызов сервиса с аргументами.

   rosservice find /TypeName ... – поиск сервиса выдающего в качестве ответа сообщение типа *TypeName*.

6. Инструмент ***rosparam*** содержит команды для получения и установки параметров ROS в сервере параметров с использованием файлов в кодировке YAML (YAML-файлов).

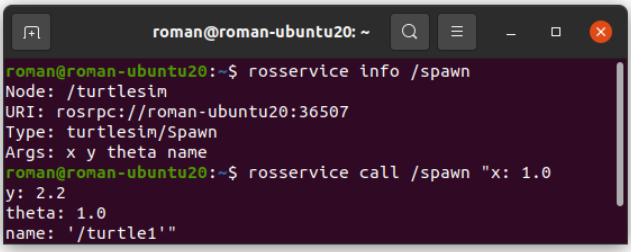

Рис. 4.3: Примеры использования *rosservice*.

rosparam list /namespace – получить список параметров из пространства имен *namespace*.

rosparam get /PrmName – получить значение параметра *PrmName*.

rosparam set /PrmName ... – задание значения параметру *PrmName*.

rosparam load – получить значение параметра из файла.

rosparam dump – записать значение параметра в файл.

rosparam delete – удалить параметр из файла.

7. Инструмент ***rosrun*** позволяет запускать ноду из заданного пакета. После написания команды *rosrun* через пробел в окне терминала пишется название пакета, а еще через пробел — название ноды из этого пакета. Пример на рисунке 4.4.

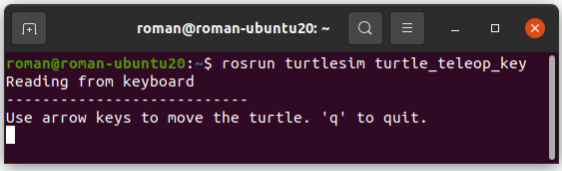

Рис. 4.4: Пример использования *rosrun*.

8. Инструмент ***roslaunch*** позволяет не запускать для каждой ноды отдельное окно терминала. Данный инструмент позволяет воспользоваться .launch файлами, которые имеют структуру XML. В данных файлах можно записать вызов нескольких нод и параметров, с

которыми они будут вызваны. Ноды будут вызваны в порядке упоминания в .launch файле. Кроме того, при использовании *roslaunch*, перед запуском первой ноды, будет автоматически запущен *roscore*. Таким образом используя *roslaunch* можно обойтись одним окном терминала, и запускать одновременно несколько нод, отвечающих за разные процессы робототехнической системы и симуляции.

roslaunch package filename.launch – пример запуска файла *filename.launch* из пакета *package*.

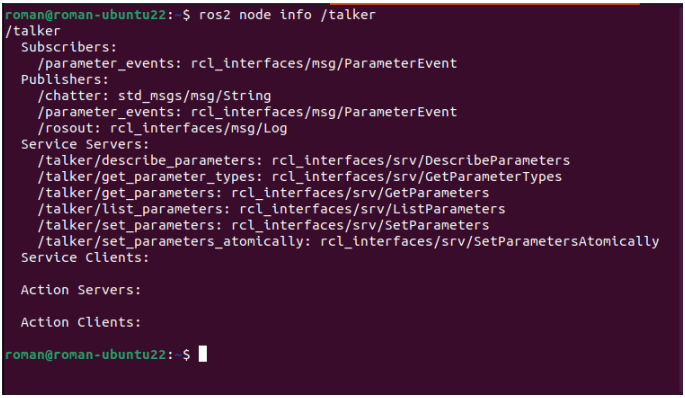

Рис. 4.5: Окно терминала после вызова *ros2 node info*.

4.1.2 Консольные команды ROS2

Как уже было сказано выше, в ROS2 нет ROS-мастера, который является основном узлом в ROS1. Поэтому в ROS2 нет аналога команды *roscore*. Все остальные команды по управлению и получению данных о топиках, нодах в ROS2 присутствуют.

Чтобы запустить отдельную ноду, нужно использовать команду *ros2 run*. Далее через пробелы указываются название пакета и название файла программы (ноды). Например, *ros2 run demo_ node_ cpp talker*.

Чтобы запустить несколько нод, используя *.launch.py* файл, нужно использовать команду *ros2 launch*. Далее так же через пробелы указываются название пакета и название файла запуска. Например, *ros2 launch demo_ node_ cpp add_ two_ ints.launch.py*.

Как вы, наверное, уже поняли, вызов всех команд в ROS2 начинается с команды *ros2*.

Так, командой *ros2 node info <название ноды>* можно получить подробную информацию о ноде (Пример на рисунке 4.5).

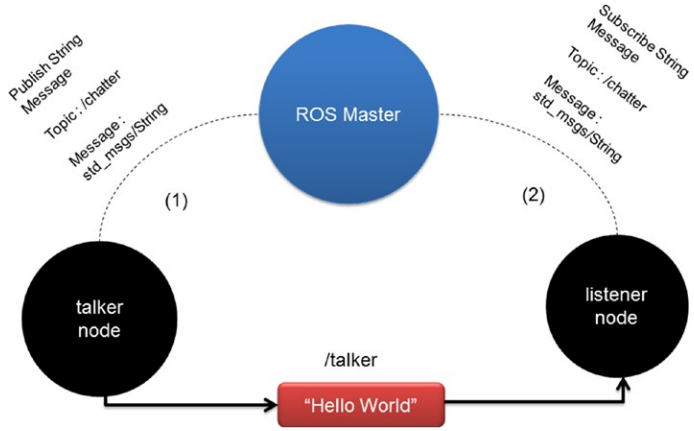

Рис. 4.6: Связь между нодами *talker* и *listener*.

Командой *ros2 node list* запрашивается список запущенных ROS программ.

С помощью команд *ros2 topic*, так же как в ROS1 можно контролировать топики. *ros2 topic info <название топика>* выдает информацию о топике. *ros2 topic pub <название топика> <содержимое>* позволяет вручную опубликовать сообщение в топик. Из нового, по сравнению с ROS1, стоит отметить команду *ros2 topic hz <название топика>* с помощью которой можно получить частоту публикаций соощений в топике.

Командами семейства *ros2 msg* можно получить справку о ссобщениях ROS2, так же как и в ROS1. А с помощью команд *ros2 service* аналогично узнаются подробности о сервисах.

По любой команде можно получить дополнительную информацию, вызвав её с флагом *–help*.

4.2 Запуск программ "Hello world"в ROS1 и ROS2

Разберем базовый пример, который прилагается к установленному ROS Noetic из пакета *roscpp_tutorials*.

Есть две ноды: *talker* и *listener*. Нода *talker* публикует строковое сообщение. Нода *listener* подписывается на него. В рассматриваемом примере *talker* публикует сообщение Hello World, а *listener* получает его и выводит на экран. Рисунок 4.6 демонстрирует схему взаимодействия двух нод. Как обсуждалось ранее, обе ноды вначале соединяются с ROS-мастером, а уже он соединяет их между собой создавая поток данных — топик */talker*.

Вначале запустим наш пример с помощью запуска нод по отдельности. Каждую команду следует запускать в новом окне терминала. Вначале нужно запустить ROS-Мастер с помощью команды:

roscore

Запустим ноду *talker* с помощью команды *rosrun*.

rosrun roscpp_tutorials talker

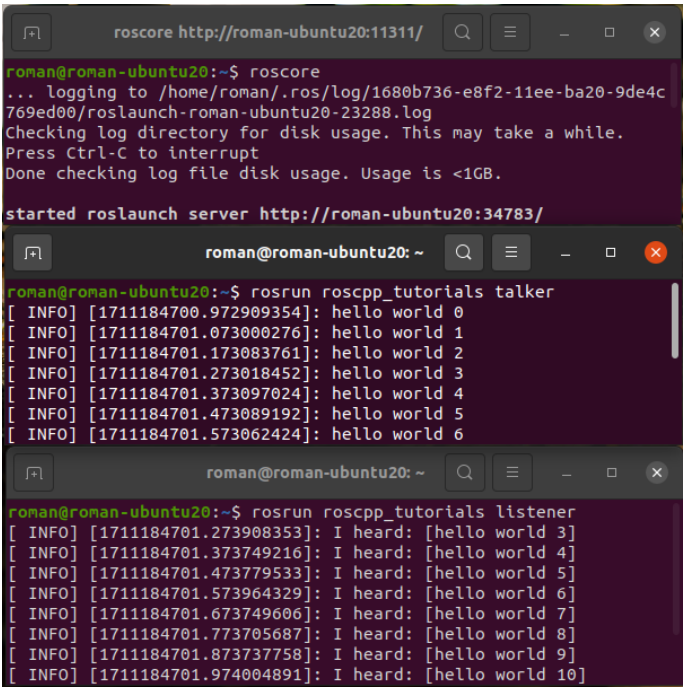

Рис. 4.7: Окна терминала при запуске нод *talker* и *listener*.

После запуска ноды вы увидите, что она с частотой 1 секунда выводит на экран на экран сообщение "hello world"и номер сообщения. В новых окнах терминала вы можете поэкспериментировать с различными инструментами ROS. Например, если вы выполните команду

rostopic list,

то в выводе увидите топик */chatter*. Именно в этот топик отправляет сообщения нода *talker*. Теперь запустите ноду, слушающую топик, используя следующую команду.

rosrun roscpp_tutorials listener

Нода *listener* начнет выводить сообщения, которые получает от ноды *talker* (рисунок 4.7).

Если вы хотите запустить две ноды в одном окне терминала, используйте команду roslaunch. Так как в пакете *roscpp_tutorials* уже есть готовый .launch файл, запускающий обе ноды, то можно закрыть все открытые окна терминала и в новом окне терминала запустить:

roslaunch roscpp_tutorials talker_listener.launch

Аналогичный пример для запуска двух нод в ROS2 был представлен выше, где проверялась работоспособность ROS2 после установки. Там нам потребовалось запустить две команды. Первая запустит ноду, которая с частотой 1 секунда выводит на экран на экран сообщение "hello world"и номер сообщения.

ros2 run demo_nodes_cpp talker

Кроме того, эта команда публикует сообщения в топик, который можно считывать и отображат полученные строки в консоли, если запустить программу *listener*:

ros2 run demo_nodes_py listener

4.3 Программа "Turtlesim"в ROS1

В этом разделе приводится пример интересного приложения для демонстрации концепций ROS 1. Приложение называется turtlesim, это 2D симулятор черепашки в нем. Вы можете перемещать черепаху, получать текущее положение черепахи, посылать на неё угловые и линейные скорости. При этом используются топики, сервисы и параметры ROS1. Работая с turtlesim, вы получите хорошее представление о том, как управлять роботом с помощью ROS. Черепашка используется только для обучения, но, фактически, моделирует любого всенаправленного робота, типа роботов-пылесоса.

Пакет turtlesim предустановлен в ROS. Для запуска, используйте следующие команды:

roscore
rosrun turtlesim turtlesim_node

Вы должны увидеть экран с расположенной в центре черепашкой (рисунок 4.8).

Далее, поэкспериментируйте с командами ROS, выводя различную информацию. Например можно вывести список всех топиков или сервисов (Рисунок 4.9).

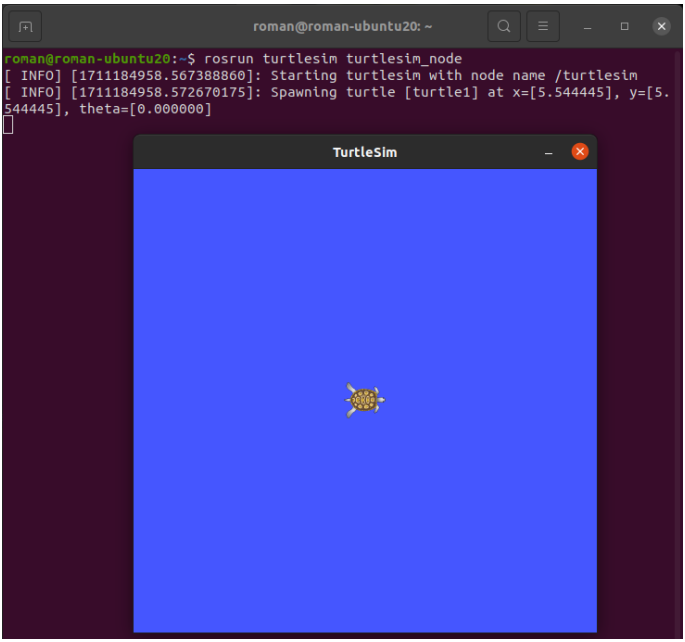

Рис. 4.8: Нода Turtlesim.

Далее, начнем подвигаем черепаху. Для этого запустим ноду ROS с помощью следующей команды. Эта команда должна запускаться в другом окне терминала.

rosrun turtlesim turtle_teleop_key

После этого, вы сможете управлять роботом с помощью клавиш со стрелками на клавиатуре.

Таким образом, мы запустили новую ноду, которая считывает нажатия клавиш, в зависимости от нажатых клавиш формирует сообщения типа *geometry_msgs/Twist*, хранящее шесть значений типа double. Из них три отвечают за линейные скорости по осям x, y, z, и три за скорости поворота вокруг этих осей. Далее, сформированные сообщения публикуются в топик */turtle1/cmd_vel*. Нода *turtlesim_node* подписана на этот топик, и после получения сообщений обрабатывает их и перемещает с заданной скоростью черепашку-робота, рисуя траекторию пройденного пути. Схема взаимодействия между нодами представлена на рисунке 4.10.

Вы можете поэкспериментировать, отправляя значения скорости вручную с нового окна терминала. Для этого нужно сформировать сообщение типа *geometry_msgs/Twist* и отправить его в топик */turtle1/cmd_vel*

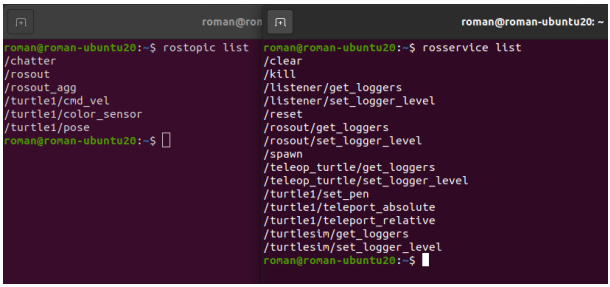

Рис. 4.9: Список запущенных топиков и сервисов Turtlesim.

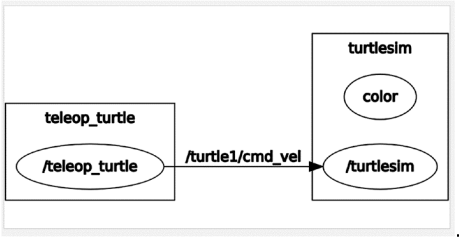

Рис. 4.10: Схема взаимодействия между нодами.

с помощью упомянутой выше функции *rostopic pub*:

rostopic pub -1 /turtle1/cmd_ vel geometry_ msgs/Twist – '[3.0, 0.0, 0.0]' '[0.0, 0.0, 2]'

Если мы хотим очистить поле черепашки от нарисованных линий, мы можем вызвать сервис под названием /reset.

rosservice call /reset

4.4 Программная совместимость ROS1 и ROS2

За период существования ROS1 на этом фреймворке было написано много полезного программного обеспечения. Это и реализации различных алгоритмов, и полноценный софт по управлению роботами с графическим интерфейсом. Для того, чтобы не потерять возможность использовать эти программы при переходе на ROS2, требовалось иметь в нем обратную совместимость с ROS1. Эту обратную совместимость обеспечивает пакет *Ros1_ bridge*. Получить данный пакет можно, собрав его из файлов исходного кода или установив его командой

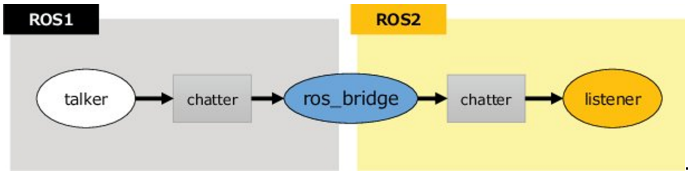

Рис. 4.11: Ros1_bridge соединяет ноды ROS1 и ROS2.

sudo apt install ros-humble-ros1-bridge

где *humble* - это название текущей версии ROS2.

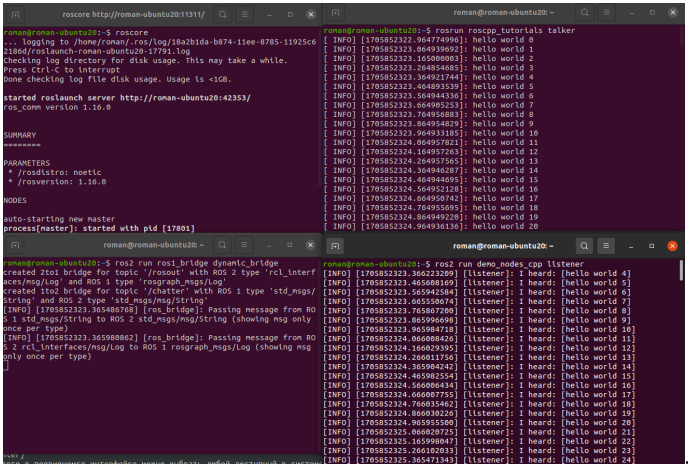

Рис. 4.12: Демонстрация работы Ros1_bridge.

Данный пакет, как можно судить по его названию (англ. bridge - мост), служит мостом, соединяя топики и сервисы между нодами из разных версий ROS. Когда пользователь запускает ноду из пакета *Ros1_bridge*, она динамически проверяет какие ноды запущены, какие топики публикуются и ожидаются. Ros1_bridge преобразует сообщения, чтобы они были доступны нодам из разных версий фрейворка (рис 4.11).

Для того чтобы проверить работу Ros1_bridge, сделаем следующее. На одной машине с Ubuntu 20.04 установим и ROS1 Noetic и ROS2 Foxy. Далее, с помощью команд

roscore

rosrun roscpp_tutorials talker

запустим ноду talker из ROS1. Эта нода будет публиковать сообщения "Hello world ..."в топик
chatter. После чего запустим Ros1_bridge:

ros2 run ros1_bridge dynamic_bridge

затем уже можно запускать ноду listener из ROS2

ros2 run demo_nodes_cpp listener

и убедиться, что она будет получать сообщения из ROS1. Работа всех этих команд продемонстрирована на рисунке 4.12.

5. Графические средства ROS

Представленные в этот разделе средства визуализации работают одинаково с различными версиями ROS1 и ROS2.

5.1 Инструменты фреймворка rqt

rqt — это программный фреймворк внутри ROS, который реализует различные аспекты ROS в удобном для пользователей виде, с использованием графического интерфейса. Любые готовые плагины с графическим интерфейсом можно запустить (при работающем ROS-Мастере) командой:

rosrun rqt_gui rqt_gui

После этого в появившемся интерфейсе можно выбрать любой доступный в системе плагин. Также, пользователи могут создавать свои собственные плагины для rqt с помощью Python или C++. На 2018 существовало более 20 плагинов. rqt заменяет прежние графические инструменты ROS rxtools, которые, начиная с версии ROS Groovy считаются устаревшими.

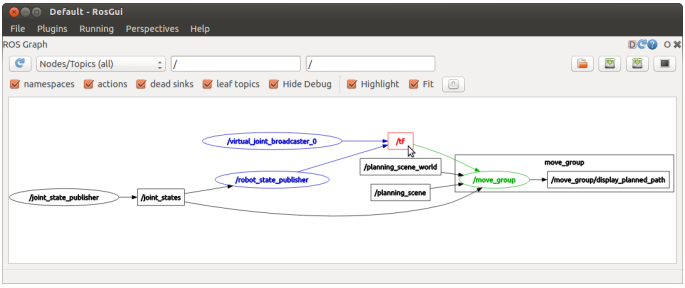

Рис. 5.1: Запущенный плагин rqt_graph, отображающий информацию о нодах и топиках.

Далее, перечислим несколько наиболее часто используемых плагинов, которые нужны для визуализации процессов ROS.

1. ***rqt_ graph***. Отображает ноды и взаимодействие между ними в виде графа. Пример на рисунке 5.1. Является интерактивным графом. То есть наводя курсором на ноды или на связывающие их топики можно получить различную дополнительную информацию. Запускается командой:

rosrun rqt_ graph rqt_ graph

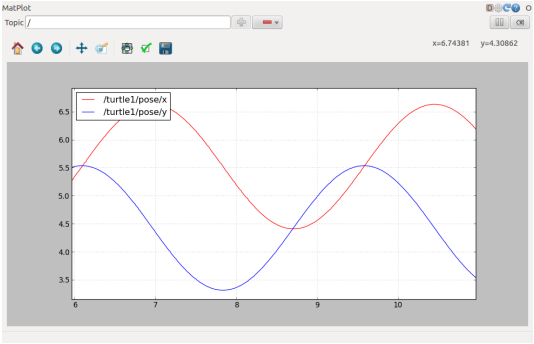

Рис. 5.2: Запущенный плагин rqt_plot, отображающий изменение координат робота-черепашки, выполняющей круговые вращения.

2. ***rqt_ plot*** предоставляет плагин ROS, визуализирующий числовые значения в 2D-графике, используя различные шаблоны рисования графиков. Например мы можем отобразить координаты робота-черепашки в виде графика (рисунок 5.2), набрав команду:

rqt_plot /turtle1/pose/x /turtle1/pose/y

3. ***rqt_ image_ view*** позволяет отображать изображение или видео с нескольких топиков-источников. Например, в первом окне можно отобразить цветное видео с камеры (например, с топика */image_ RGB*, а во втором — черно-белое, после его обработки в ноде (например, с топика */image_ Grey*.

4. ***rqt_ reconfigure*** предоставляет способ просмотра и редактирования параметров нод. Когда вы используете, к примеру, ноды планирования пути или локализации для мобильных роботов, в них имеется большое количество параметров, влияющих на результаты алгоритмов. В зависимости от поставленной задачи, любой из подобных алгоритмов будет нуждаться в дополнительной настройке, что

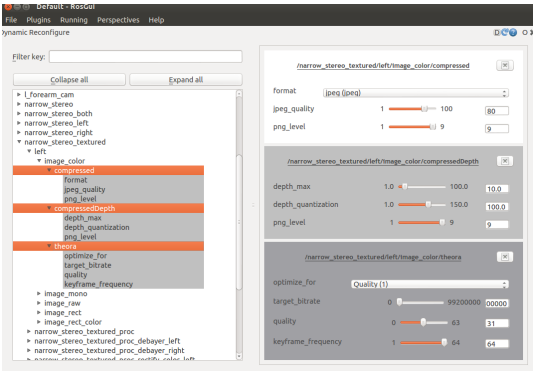

Рис. 5.3: Запущенный плагин rqt_reconfigure, в котором можно настраивать параметры нод.

и можно сделать, регулируя параметры с помощью *rqt_reconfigure*. Пример окна представлен на рисунке 5.3.

Существует еще много готовых плагинов, упрощающих разработку собственных нод ROS и исследования существующих. Плагин ***rqt_launch*** упрощает работу с .launch файлами, анализируя ноды в нем и позволяя запускать и останавливать их работу по отдельности. Плагин ***rqt_topic*** отображает информацию о топиках, кто на них подписан и кто в них публикует. ***rqt_msg*** может визуально представить структуру передаваемых сообщений и в каких топиках они передаются. Это еще не весь список плагинов, полезных при исследовании и создании сложных, многопроцессорных алгоритмов. Плагины rqt поддерживаются во всех выходящих дистрибутивах ROS и для разработчиков открыт шаблон по созданию собственных плагинов для ROS.

5.2 Симулятор RViz

Наряду с инструментами командной строки, ROS имеет программы с графическим интерфейсом для визуализации данных с датчиков. ***RViz*** — это программа визуализации, которая используется в ROS. В ней используется 3D-среда, которая позволяет пользователям видеть мир с точки зрения робота. Используя 3D, мы можем визуализировать данные с топиков: это может быть изображение с камеры, 3D облако точек, данные с лазерного дальномера робота, двух- и трехмерные карты, получаемые в результате картографирования. Также можно получить информацию о положении суставов (джоинтов — англ. joints) робота, о системах координат каждого сочленения робота. Запустить RViz (если уже запущен

ROS-Мастер) можно командой:

rosrun rviz rviz

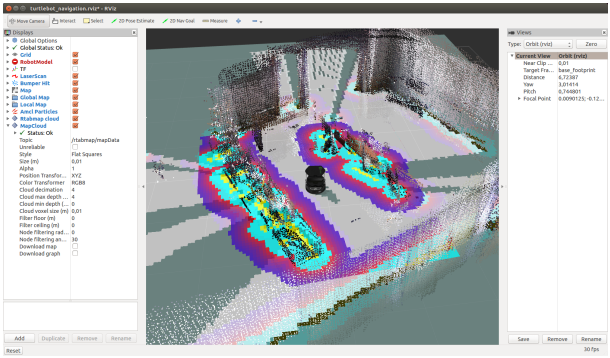

Рис. 5.4: Пример запущенного симулятора RViz. Отображены 2D-карта, получаемое 3D-облако точек и внешний вид робота.

Пример запущенного симулятора RViz можно увидеть на рисунке 5.4. В главном окне 3D-визуализации отображаются данные с датчиков, различные системы координат (мировые, базы роботов и их сочленений) и другие виды информации. Слева находится панель "Displays". В ней находится информация о том, что выбрано для отображения и с какого топика идет чтение данных. Например, на рисунке 5.4 видно, что выбрано отображение облака точек (MapCloud) и чтение данных идет с топика (/rtabmap/mapData). Внизу панели "Displays"находятся кнопки редактирования списка данных. Кнопка "Add"для добавления новых источников данных, "Remove— для удаления из списка и т.д. Сверху находится панель инструментов. В ней мы можем изменять положение камеры, отдалять и приближать изображение, а также устанавливать цели для робота, если среди запущенных нод имеются планировщики пути.

Итак, симулятор RViz необходим, чтобы отображать различные данные с сенсоров робота и другие данные. Но как создать виртуальную среду для тестирования роботов. Окружение робота можно сформировать с помощью симулятора Gazebo.

5.3 Симулятор Gazebo

Gazebo 3D, также как и ROS, разрабатывается некоммерческой организацией OSRF (Open Source Robotics Foundation). Он бесплатный и имеет открытый код. Кроме того, он очень популярен среди мирового робототехнического сообщества и является официальным симулято-

ром соревнований DARPA. Gazebo отлично интегрируется с программной платформой ROS (Robot Operating System), а значит разработанную вами программу управления виртуальным роботом в Gazebo и ROS будет относительно несложно перенести на реального робота.

Gazebo позволяет моделировать динамику и кинематику механизмов роботов (включая моменты взаимодействия с внешней средой) и формировать физически правдоподобные показания виртуальных датчиков.

Симулятор Gazebo имеет свой собственный редактор, позволяющий без программирования создавать трехмерные сцены и включающий огромную библиотеку моделей. Программа также предоставляет следующие возможности:

1. Использование популярных общеизвестных моделей роботов, таких как: iRobot Create, PR2, TurtleBot, Pioneer 2 DX, Segway RMP, Pioneer 2 AT. Помимо заранее созданных разработчиками моделей есть возможность самостоятельного проектирования необходимых устройств (сенсоров и роботов), загрузки их в мир и дальнейшей симуляции. Однако устройства, модели которых уже есть в программе, эмулируются с гораздо более высокой точностью.

2. Поддержка и симулирование работы множества различных сенсоров, в том числе сонара, лазерного дальномера, датчиков семейства IMU, моно- и стереокамер, кинект-сенсоров, прибора для чтения RFID-меток и других.

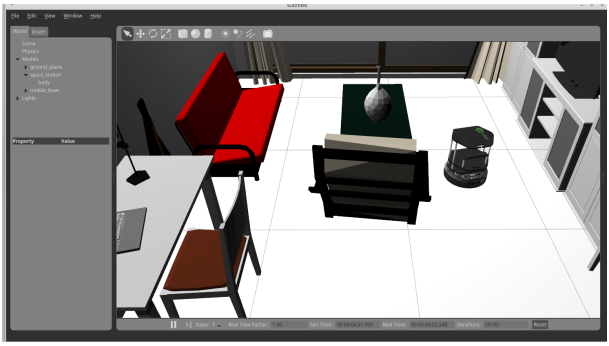

Рис. 5.5: Окно Gazebo. Создана среда, имитирующая жилую комнату, в ней находится робот Turtlebot.

Симулятор состоит из сервера (***gzserver***), который занимается просчетом физики, столкновений симуляцией, сенсоров. В качестве физического движка там используется в том числе библиотека Bullet3, которая

может ускорять вычисления с помощью OpenCL на GPU. К серверу могут подсоединятся клиенты — ***gzclient*** — один из них. Именно он занимается отображением графического интерфейса. Для сенсоров и моделей можно писать свои плагины — это позволяет управлять самой моделью и симулировать входные данные от датчиков. Сервер и клиент Gazebo запускаются последовательно после ввода в терминале команды:

gazebo

После запуска этой команды вы увидите пустую 3D-среду, куда можно добавлять стандартные объекты из верхней панели 5.5. Однако чаще всего вызов среды Gazebo с размещенном в какой либо среде робота происходит в .launch файлах. Пример созданной среды и робота в ней можно увидеть на рисунке 5.5.

Создаваемое окружение описывается в формате SDF (Simulation Description Format) и обычно имеет расширение .world. В нем могут описываться все аспекты окружающего мира: освещение, ветер, гравитация, магнитное поле, и т.д. Кроме того, в нем же описаны добавленные в среду статичные (дома, мебель) и динамические (мобильные роботы) объекты и их первоначальное положение. Пример такого файла на рисунке 5.6.

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>

    <model name="my_mesh">
      <pose>10 0 0 0 0 0</pose>
      <static>true</static>
      <link name="body">
        <visual name="parallel_wall">
            <geometry>
                <mesh><uri>file://maze_parallel_wall.dae</uri></mesh>
            </geometry>
        </visual>
      </link>
    </model>
  </world>
</sdf>
```

Рис. 5.6: Пример .world файла с добавленным освещением, поверхностью и 3D-моделью лабиринта.

Заключение

В данном пособии описано что такое Робототехническая Операционная Система, объяснены основные понятия ROS. Обозначены отличия фреймворков ROS1 и ROS2. Были приведены примеры использования ROS, перечислено какие средства командной строки и программы с графическим интерфейсом можно использовать для анализа и создания собственных ROS-систем. Кроме того, проведен обзор симуляторов, которые как отображают данные с сенсоров (RViz) так и создают физическую симуляцию для генерации этих данных (Gazebo)

В следующем пособии будут разобраны примеры программирования собственных алгоритмов для ROS и создание собственных роботов.

Все предложения и замечания просьба высылать по адресу *lavrenov@it.kfu.ru*.

Литература

[1] *Quigley M.* Programming Robots with ROS: a practical introduction to the Robot Operating System / Quigley M., Gerkey B., Smart W. D. — "O'Reilly Media, Inc. 2015. — 448 p.

[2] *Koubâa A.* Robot Operating System (ROS) / Quigley M. – Verlag : Springer, 2017. — 728 p.

[3] *Martinez A.* Learning ROS for robotics programming/ Martinez A., Fernández E. — Packt Publishing Ltd, 2013. — 332 p.

[4] ROS Tutorials. — URL:http://wiki.ros.org/ROS/Tutorials (дата обращения 09.01.2024)

[5] ROS2 vs ROS1.Установка ROS2 на Ubuntu 18.04 — URL:https://habr.com/ru/articles/492058 (дата обращения 09.01.2024)

[6] eProsima Fast DDS Documentation — URL: https://fast-dds.docs.eprosima.com/en/latest (дата обращения 09.01.2024)