**Федеральное государственное автономное образовательное учреждение высшего образования**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ» (РУДН)**

Основное учебное подразделение ФФМиЕН

Направление/специальность 02.03.02 «Фундаментальная информатика и информаци-

онные технологии»

**НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА**

**на тему**

**«Моделирование и управление робототехническими системами в ROS »**

Ломакина София Васильевна

Курс, группа 4 курс, группа НФИбд 02-19

Место прохождения практики «Отдел информационно-технического обеспечения естественно-научных факультетов УИТО и СТС РУДН»

Сроки прохождения с «18» апреля 2022 г. по «19» июня 2022 г.

Руководители практики:

от РУДН Фомин М.Б., к.ф.-м.н, доцент, доцент

кафедры информационных технологий

от организации (предприятия)

Киселёв Г. А., к.т.н., ст. преподаватель

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2022 г.

**Содержание**

[Введение 3](#)

[Robot Operating System 5](#)

[Проекты на базе ROS 8](#)

[Уровень файловой системы 10](#)

[Уровень вычислений 12](#)

[Выбор ПО для моделирования 16](#)

[Симуляция модели реального мобильного робота 17](#)

[Узел laser\_data 25](#)

[Узел avoid\_obstacle 27](#)

[Заключение 31](#)

[Список использованных источников и литературы 33](#)

[Приложение А 34](#)

[Приложение Б 35](#)

# Введение

В настоящее время робототехника превратилась в развитую отрасль промышленности: тысячи роботов работают на различных предприятиях мира, подводные манипуляторы стали непременной принадлежностью подводных исследовательских и спасательных аппаратов, изучение космоса опирается на широкое использование роботов с различным уровнем интеллекта. Особенное внимание уделяется автоматизации тяжелых, вредных, утомительных и монотонных работ в различных отраслях с помощью роботов-манипуляторов.

Дадим определение понятию «робот». Робот (от чеш. Robota - «подневольный труд») - автоматическое устройство, предназначенное для осуществления различного рода механических операций, действующее по заложенной заранее программе. Робот обычно получает информацию о состоянии окружающего пространства посредством датчиков (технических аналогов органов чувств живых организмов). Робот может самостоятельно осуществлять производственные и иные операции, частично или полностью заменяя труд человека. Робот может иметь связь с оператором, получая от него команды (ручное управление), действовать автономно, в соответствии с заложенной программой (автоматическое управление).

Перейдем к понятию «операционная система», сокр. ОС (англ. operating system, OS). Операционная система – комплекс взаимосвязанных программ (набор инструкций), предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем. Без операционной системы компьютер – это просто набор микросхем. На базе операционной системы работают все программы, которые мы используем. Именно ОС позволяет пользователю использовать возможности «железа» (hardware). К операционным системам, пользующимся на сегодняшний день популярностью у пользователей и разработчиков программного обеспечения, можно отнести MacOS, Linux, Windows, iOS, Android и другие. Сегодня у специалистов в области робототехники возникают примерно те же трудности, что и 30 лет назад у разработчиков ЭВМ. Из-за отсутствия общих стандартов и платформ создателям роботов приходится начинать разработку каждого нового творения практически с нуля. Для решения данной проблемы по всему миру ведется разработка различных операционных систем и фреймворков для роботов.

Таким образом, актуальность выбранной темы обусловлена значимостью развития подобных платформ в современной робототехнике.

Целью настоящей работы является изучение структуры и особенностей функционирования фреймворка для разработки и управления робототехнических систем Robot Operating System.

Задачи, решаемые в работе:

1. Описание фреймворка ROS и его базовых понятий.
2. Исследование принципа работы Robot Operating System на примерах простых программ, управляющих поведением виртуальной идеализированной модели робота.
3. Разработка собственной модели виртуального робототехнического устройства и написание программы управления им в среде ROS.

# 

# Robot Operating System

ROS (Robot Operating System) – это набор инструментов, библиотек и соглашений, направленных на упрощение задачи создания сложного и надежного поведения робота на различных роботизированных платформах. Подобно ОС, ROS состоит из двух частей: непосредственно ядра фреймворка ros и ros-pkg, набора поддерживаемых сообществом разработчиков пакетов, которые реализуют различные функции робототехники: позиционирование, планирование, восприятие, моделирование и другие.

Преимуществом ROS является аппаратная абстракция, позволяющая разработчику проектировать алгоритмы управления роботами таким образом, чтобы они могли быть перенесены на любое оборудование, обеспечивающее выполнение созданных алгоритмов. Для работы с реальным оборудованием ROS содержит драйверы устройств различных производителей.

Основная разработка ROS ведется на базе UNIX-подобных операционных систем: Linux Ubuntu, Mac OS X.

Перечислим особенности ROS:

1. Доступность. ROS выпускается в соответствии с условиями BSD-лицензии и c открытым исходным кодом, и код в нем является результатом совместных усилий сообщества разработчиков со всего мира. ROS бесплатен для использования, как в исследовательских, так и в коммерческих целях.
2. Поиск и исправление ошибок. Одной из сильных сторон ROS является то, что определенная задача, например, управление колесным мобильным роботом, может быть разделена на ряд более простых задач. Задачи могут включать в себя восприятие окружающей среды с помощью камеры или лазерного дальномера, составление карты, планирование маршрута (навигация), мониторинг уровня заряда батареи робота и управление двигателями, приводящими в движение колеса робота. Каждое из этих действий может быть запрограммировано в виде узла или серии узлов для выполнения конкретных задач. О том, что такое узлы, будет рассказано ниже. Эта концепция, используемая в ROS, полезна для поиска ошибок, потому что программы, которые разделены на минимальные функции (изолированы), могут быть отлажены по отдельности.
3. Гибкость. Одна из основных целей ROS – облегчить связь между модулями ROS, называемыми узлами. Эти узлы представляют собой исполняемый код, и код может полностью находиться на одном компьютере, или узлы могут быть распределены между компьютерами или между компьютерами и роботами. Преимущество этой распределенной структуры состоит в том, что каждый узел может управлять одним свойством системы. К примеру, аппаратно-программная платформа Arduino, публикующий сообщения ноутбук, подписывающийся на них, и Android-телефон, приводящий в движение робота. Это делает ROS действительно гибкой и адаптируемой к потребностям пользователя.
4. Поддержка различных языков программирования. ROS содержит библиотеки для поддержки различных языков программирования. Наиболее популярны из них Python (rospy), C ++ (roscpp).
5. Поддержка различных инструментов разработки. ROS предоставляет инструменты отладки и тестирования (например, пакет rostest), двумерной (2D) визуализации rqt визуализации ROS с графическим интерфейсом в виде плагинов для анализа передаваемых сообщений, отображения временных данных, показаний датчиков и т.д.), инструмент трехмерной визуализации RViz (ROS Visualization) и 3D-симулятор Gazebo для создания симуляций работы робототехнических систем в окружении виртуальных объектов. На рисунке 1 показан пример рабочей сцены в RViz.

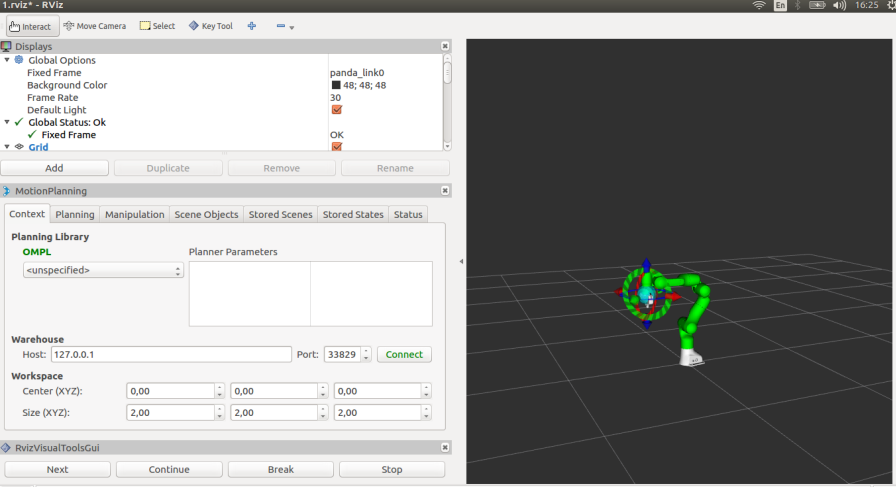


Рисунок 1 – Пример сцены RViz

1. Активное сообщество. Сообщество пользователей ROS является наиболее активным, чем какие-либо другие сообщества робототехнических платформ, потому что ROS разрабатывается не только Open Robotics, но и академическими исследователями, разработчиками и любителями со всего мира. Тем самым, когда вы сталкиваетесь с проблемой, вам проще найти ее решение, посетив раздел официального сайта ROS «Q & A (вопрос-ответ)», а также различные форумы в интернете.

# 

# Проекты на базе ROS

ROS на сегодняшний день находит применение в робототехнических проектах многих исследовательских групп и компаний. Сегодня ROS объединяет более ста роботов, от автономных автомобилей до БПЛА и человекоподобных роботов с использованием множества датчиков, реализованных с помощью ROS либо обладающих поддержкой ROS, что позволяет робототехникам со всего мира быстрее и дешевле создавать интересные проекты, прокладывая путь к значительным прорывам. Опишем некоторые проекты.

*Мобильный робот TurtleBot3.*

TurtleBot3 - это небольшой, доступный, программируемый мобильный робот на базе ROS для использования в образовательных организациях, различных исследованиях и для любителей робототехники.

Технологии, используемые в TurtleBot3: SLAM, навигация и манипулирование. Аппаратное обеспечение и ПО TurtleBot3 имеют открытый исходный код. 3D-модели деталей TurtleBot3 можно загрузить с официального репозитория разработчиков либо создать свои собственные и распечатать их на 3D-принтере.

На рисунке 2 представлены модели TurtleBot3: Burger, Waffle, Waffle Pi.

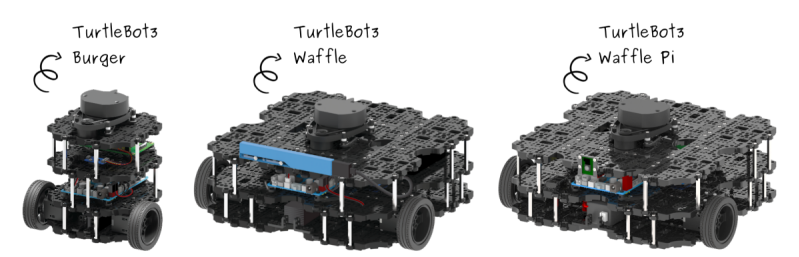


Рисунок 2 – Модели TurtleBot3

*Антропоморфная рука Shadow Hand.*

Размер, вес и гибкость 5-пальцевой руки Shadow Hand (рисунок 3) близки к человеческой и могут быть использованы для ловких манипуляций и жестов. Рука полностью интегрирована с ROS. Управление рукой, включая алгоритмы контроля положения, может быть изменено пользователем (прошивка доступна для модификации).



Рисунок 3 - Антропоморфная рука Shadow Hand

# 

# Уровень файловой системы

Программы ROS организуются с помощью пакетов, каждый из которых содержит все необходимые файлы - cpp, py, параметры, конфигурации и т.д. Несколько пакетов, имеющих общую цель, могут быть объединены в метапакет (стек), а иерархия файловой системы ROS показывает связь между пакетами и метапакетами. Система ROS устанавливается в корневой каталог Linux и использует иерархию каталогов, определенную стандартом иерархии файловых систем (FHS) в Linux.

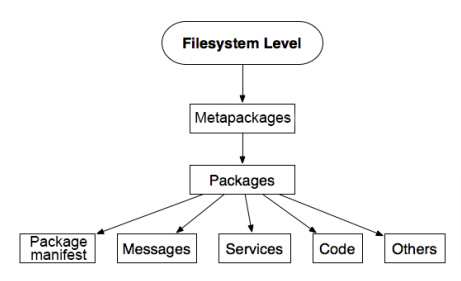


Рисунок 4 - файловая система

Мастерская (рабочее пространство - workspace) - это каталог, содержащий пакеты с исходными файлами, которые можно скомпилировать, используя мастерскую. Это удобно, когда нужно скомпилировать несколько пакетов одновременно и централизовать все разработки. Вы можете создать свою мастерскую и назвать ее как угодно, где угодно. В ROS по умолчанию мастерской является директория catkin\_ws, где вы можете вносить изменения, компилировать и устанавливать пакеты catkin. Путь: /home/user/catkin\_ws. Для перехода в директорию рабочего пространства в терминале Linux нужно ввести команду: «cd ~/catkin\_ws».

Каталог, содержащий несколько папок, предоставляет различные пространства с разными функциями. Например, исходное пространство (src) предназначено для хранения своих пакетов, проектов и клонированных пакетов с одним из важных файлов - CMakeLists.txt, который используется для сборки пакета с помощью инструмента CMake. Пространство сборки (build) содержит промежуточные файлы для пакетов и проектов, а пространство разработки (devel) используется для хранения скомпилированных программ и их тестирования без установки или для экспортирования пакета. Для перемещения между пакетами и их файлами ROS обеспечивает нам полезную утилиту rosbash, которая является подмножеством bash Linux и имеет подобные команды. Интерпретатор оболочки bash Linux используется для выполнения навигации по файловой системе, запуска программ и общения с устройствами.

# 

# Уровень вычислений

Данный уровень представляет собой граф вычислений - сеть процессов ROS, которые совместно обрабатывают данные.

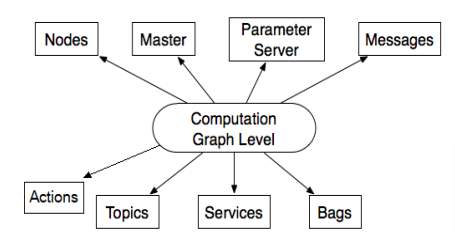


Рисунок 5 - составляющие графа вычислений

Master (мастер) в системе ROS присваивает названия узлам и сохраняет их в системе; отслеживает издателей и подписчиков, связанных с определенными темами. Он играет ключевую роль для соединения узлов ROS между собой, похоже на сервер DNS. Запуск мастера осуществляется через команду "roscore", которая также запускает сервер параметров ROS и систему логирования rosout.

Nodes (узлы) в ROS - это исполняемые файлы, позволяющие программам взаимодействовать с другими процессами. Использование узлов в системе ROS имеет много преимуществ: увеличивает отказоустойчивость, изолирует код и функциональные возможности, делает систему более удобной. Узлы также могут обмениваться сообщениями, состоящими из данных, команд или другой информации, необходимой для решения конкретных задач.

Parameter server – это сервер, который хранит набор значений, используемых узлами во время выполнения программы, например, радиус колеса робота. С помощью команды "rosparam" можно изменять настройки узлов, хранящихся на сервере, например, настраивать максимальную и минимальную скорость робота.

Messages – это сообщения различных типов, которые могут содержать разнообразные данные (текст, изображения, положение робота и другие), а также иметь структуры, такие как массив сообщений. Общее количество типов сообщений составляет более 200.

Существует 3 метода обмена сообщениями (рисунок 6):

1. Тема (topic) обеспечивает однонаправленную связь (передачу / прием сообщений)
2. Сервис (service) обеспечивает двунаправленную связь (запрос (request) / ответ (response) на сообщение)
3. Действие (action) обеспечивает двунаправленную связь (цель сообщения (goal) / результат (result) / обратную связь (feedback)).

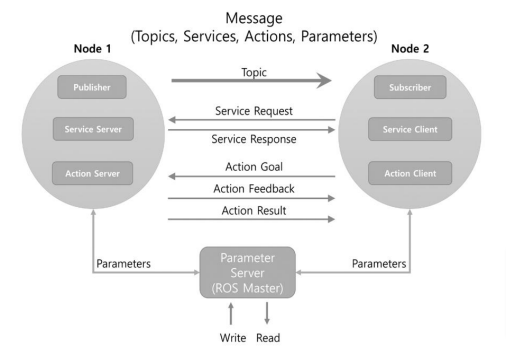


Рисунок 6 - методы обмена сообщениями между узлами

Рассмотрим эти методы подробнее.

1. Topic (тема, топик).

В системе ROS используется шина, называемая темой, для передачи данных между узлами, которые делятся на издателей и подписчиков. Первым шагом узел-издатель регистрирует свою тему в мастере, а затем начинает отправлять сообщения через нее. Узлы-подписчики получают информацию о теме узла-издателя от мастера и затем напрямую соединяются с ним, чтобы получать сообщения. На рисунках 13 и 14 показано, как узлы соединяются в системе ROS, используя мастер. Тема может использоваться для передачи потока сообщений от датчиков, таких как данные о местоположении робота или расстояние до препятствий. Узлы могут не только публиковать сообщения в теме, но и подписываться на нее, чтобы получать информацию.

1. Service (Сервис).

ROS предназначен для обеспечения коммуникации между узлами через модель сервис. Эта модель включает в себя сервер, который отвечает на запросы, и клиента, который отправляет запросы, и работает путем обмена двумя сообщениями: запросом и ответом. В отличие от темы, сервис представляет собой однократный обмен сообщениями, и после завершения обмена связь между узлами прерывается.

1. Action (Действие).

Для задач, требующих продолжительного выполнения и обратной связи с клиентом, обычно используется модель коммуникации в режиме Действие (Action). В отличие от Сервиса, этот подход в основном используется для управления сложными задачами робота, такими как перемещение к указанной точке или запуск лазерного сканирования.

1. Rosbag и bag-файлы.

Важно отметить rosbag как функциональный инструмент, позволяющий сохранять данные, передаваемые через топики ROS, на специальные bag-файлы. Rosbag также способен воспроизводить bag-файлы с данными, которые будут публиковаться в те же топики, что и изначально. Это полезно для имитации данных от сенсоров, доступ к которым отсутствует.

# 

# Выбор ПО для моделирования

В работе использовался дистрибутив OC Linux Ubuntu 20.04, установленный на виртуальной машине Oracle VM VirtualBox и дистрибутив ROS: Noetic, который установлен в соответствии с инструкцией на официальном сайте.

Для программирования был выбран язык Python, по причине его простоты, развитой системы API и библиотеки для ROS.

Для написания кода был выбран редактор VS Code, созданный Microsoft, так как он обладает большими возможностями, включая отладку программ, подсветку синтаксиса, интеллектуальное завершение кода и встроенный Git, что значительно упрощает процесс разработки.

# 

# Симуляция модели реального мобильного робота

Одно из главных преимуществ использования робототехнической платформы ROS является то, что для написания собственных алгоритмов управления не потребуется приобретение дорогостоящего оборудования или ПО, а отладка и демонстрация работы алгоритмов может проводится в 3D-симуляторе Gazebo.

Что такое Gazebo?

Gazebo - это трехмерный динамический симулятор с высокой производительностью и открытым исходным кодом, способный точно и эффективно симулировать роботов в сложных условиях благодаря использованию нескольких физических движков, богатой библиотеке моделей роботов, датчиков и сред.

Gazebo 3D имеет ряд преимуществ по сравнению с другими робототехническими симуляторами. Во-первых, он бесплатный и имеет открытый код. Во-вторых, он очень популярен среди мирового робототехнического сообщества и является официальным симулятором соревнований DARPA. В-третьих, Gazebo отлично интегрируется с программной платформой ROS (Robot Operating System), а значит разработанную программу управления виртуальным роботом в Gazebo и ROS будет относительно несложно перенести на реального робота.

Приведем пример системы управления движением реального автономного мобильного робота, состоящей из трех вычислительных узлов (рисунок 7).

Первый узел (wheel odometer) – одометр, который использует данные о движении приводов, для оценки перемещения. Данный узел публикует данные о положении робота (x: 0.1, y: -1, z: 0) в тему /odom.

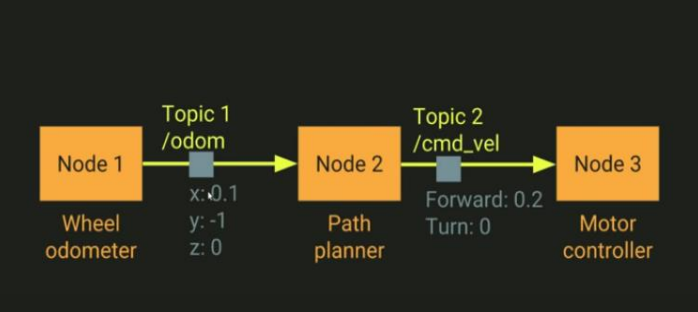


Рисунок 7 - Граф узлов мобильного робота (упрощенная схема)

Второй узел (Path planner), отвечающий за планирование движения от источника (получает сообщения из темы /odom) к месту назначения путём вычисления необходимых данных (параметров) и их передачи в виде сообщений в тему cmd\_vel, в свою очередь третий узел Motor controller принимает эти сообщения типа geometry\_msgs/Twist, которые преобразуются в команды управления вращением двигателей, и робот осуществляет перемещение в заданном направлении.

Подобную систему управления можно создать самостоятельно или же использовать готовые решения, которые предлагает ROS, например навигационный стек ROS, состоящий из 3 основных пакетов, предназначенных для решения навигационных задач высокого уровня:

* gmapping строит карту помещения на основе алгоритма SLAM, данных лазерного дальномера и одометрии;
* amcl определяет местоположение робота уже на существующей карте, то есть, алгоритм узла amcl (Adaptive Monte Carlo Localization) работает с уже готовой картой, например, созданной с помощью пакета gmapping;
* move\_base позволяет перемещать робота в любую точку на карте с учетом препятствий. Можно перемещать робота по карте, управляя им с клавиатуры, пакет move\_base также позволяет осуществлять его автономное перемещение. Достаточно задать точку на карте, и робот самостоятельно проложит оптимальный маршрут до цели.

Использование одних одометров является относительно простым способом определения позиции робота, однако этот метод не отличается высокой точностью. Использование GPS не всегда представляется возможным. Тогда разумным решением будет использование карты для определения положения робота, но откуда мы можем ее получить, если окружающее робота пространство неизвестно?

На практике, возможность загрузить рабочую карту окружающего пространства имеется не всегда, поэтому встает задача: научить робота строить карту местности и одновременно определять положение в этой местности и траекторию движения. Такая задача может вставать в различных областях, где применяются роботы: от роботов-пылесосов до автономных марсоходов или роботов-исследователей морского дна. Область знаний, описывающая методы решения данной задачи, получила название SLAM (одновременная локализация и построение карты). На данный момент существует довольно большое количество реализаций и подходов, опирающихся на различные программные и аппаратные возможности платформ. Таким образом, алгоритмы определения положения и ориентации роботов в пространстве постоянно совершенствуются.

Контроль за работой робота осуществляется при помощи системы управления. Она включает в себя огромное количество датчиков, которые помогают роботу взаимодействовать с внешним миром, в частности мобильный робот Turtlebot3 Burger оснащен лазерным дальномером 360 ° LiDAR (лидар) с “углом обзора” равным 360 градусов, 9-осевым гиростабилизатором (IMU) и высокоточным энкодером, что позволяет использовать его для решения задач автономной навигации, локализации и тестирования алгоритмов SLAM.

Возьмём симуляцию модели мобильного робота Turtlebot3 и создадим несколько узлов для его управления.

Для того, чтобы начать использовать симуляцию модели Turtlebot3 нужно загрузить необходимые пакеты с официального репозитория его производителя - Robotis на Github. Инструкция по его установке приведена в Приложении А.

На рисунке 8 представлено изображение мобильного робота TurtleBot3 и названия его компонентов справа.



Рисунок 8 - Turtlebot3 Burger

Запуск симуляции.

Запустим симуляцию мобильного робота из установленного пакета turtlebot3\_gazebo с помощью следующей команды:

roslaunch turtlebot3\_gazebo turtlebot3\_world.launch.

Мы можем изменить сцену в которой находится робот или создать свою с помощью встроенных инструментов Gazebo. Как видно на рисунке 9, cимуляция запустилась.

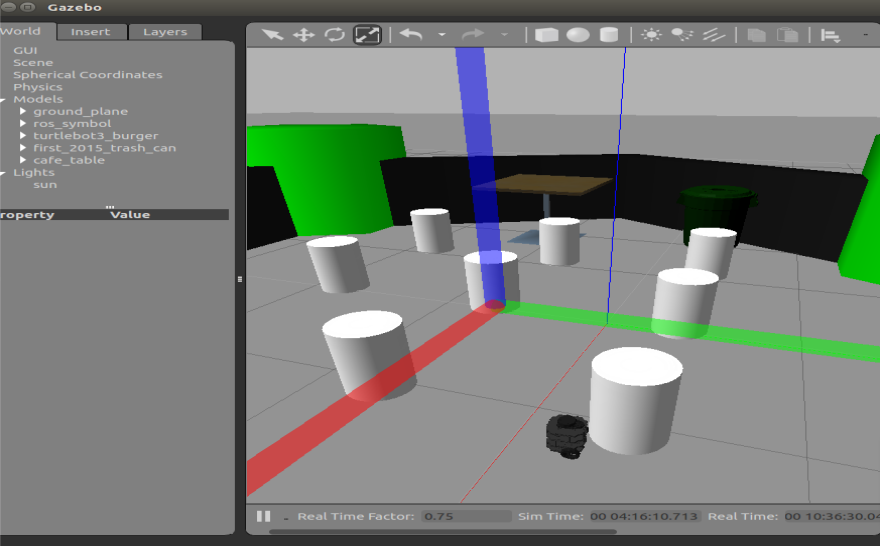


Рисунок 9 – Сцена World в среде Gazebo

Определение глубины (расстояния до объектов) является фундаментальной задачей в контексте её применения в робототехнике и технологии AV (Autonomous Vehicle). Датчики глубины, например RGB-D камеры, такие как Kinect Xbox 360 как правило, более точные из-за ограниченного расстояния в помещении и хороших условий освещения. Для выполнения задач на открытом воздухе, обычно выбирается LiDAR (лидар) изза его высокой точности и большого радиуса действия. Модель Turtlebot3 Burger имеет на своём борту лидар. На рисунке 10 показано как он выглядит.



Рисунок 10 – LiDAR 360°

Характеристики LiDAR 360°:

* Диапазон измерений (радиус действия): 120 - 3500 мм;
* Угол сканирования (обзора): 360°;
* Угловое разрешение: 1°.

Мобильный робот Turtlebot3 Burger периодически считывает данные с лидара, который сканирует окружение, посылая несколько сотен лучей, отстоящих друг от друга на равные углы так, чтобы покрывать весь угол обзора. Таким образом, в каждый период робот получает набор значений, на каком расстоянии от него находятся некоторые препятствия внутри угла обзора. По приходящим “сканам” робот может строить карту и определять свое положение на ней, используя разные алгоритмы SLAM. На рисунке 11 приведена иллюстрация процесса сканирования окружения лидаром.

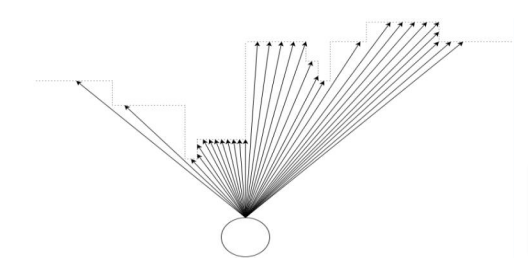


Рисунок 11 – LiDAR 360° сканирует окружение

Визуализируя то, что робот «видит» и делает, позволяет упростить отладку и разработку программ робота. Для этого ROS предлагает такой инструмент как RViz. RViz – это инструмент трехмерной визуализации для ROS. Rviz отображает данные со стереокамер, лидаров и других датчиков робота в режиме реального времени в виде облака точек или изображений глубины (depth image). Если реальный робот связывается с рабочей станцией, на которой запущен rviz, rviz отобразит текущую конфигурацию робота на модели виртуального робота. Покажем это на примере робота Turtlebot3 Burger. Запустим RViz с помощью команды:

roslaunch turtlebot3\_gazebo turtlebot3\_gazebo\_rviz.launch

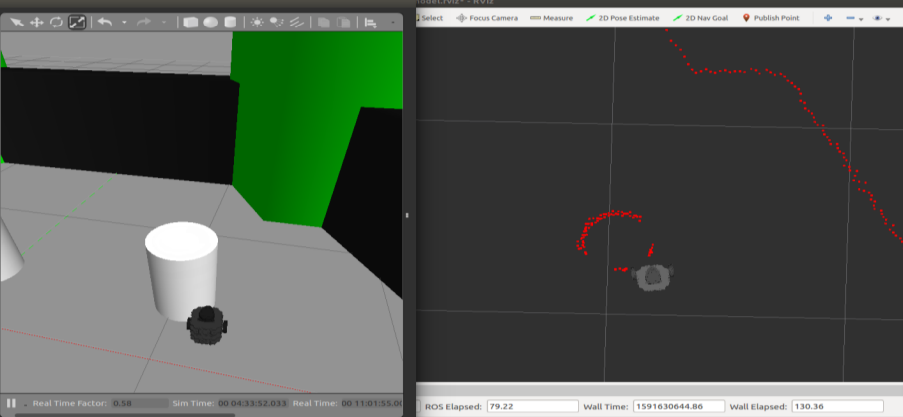


Рисунок 12 – Симуляция робота в Gazebo (окно слева) и облако точек в RViz

Как видно на рисунке 12, RViz отображает робота и объекты находящиеся его в поле зрения выделены красными точками.

# 

# Узел laser\_data

Создадим программу, которая будет выводить на экран расстояние до объектов относительно робота.

Используемая единица измерения расстояния – метр [м].

Загрузим симуляцию робота TurtleBot3 и создадим сцену, где вокруг него установлены 4 стены и зеленый бак. Полученная сцена показана на рисунке 13.

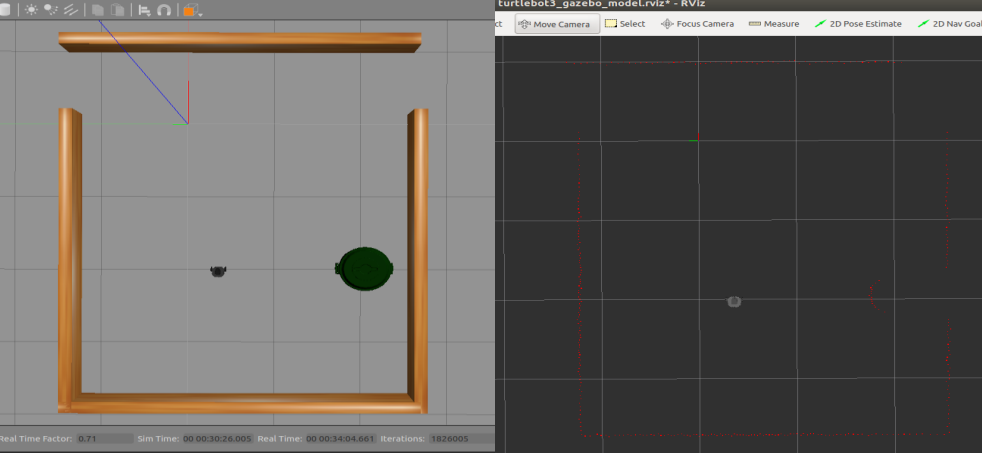


Рисунок 13 – Созданная вручную сцена в Gazebo и RViz

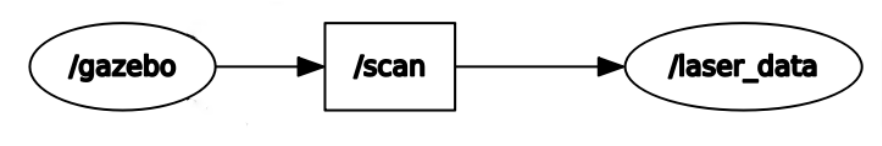


Рисунок 14 – Структура программы для получения данных с лидара виртуальной модели робота в Gazebo

Чтобы узнать расстояние до объектов находящихся вокруг робота создадим узел, который будет подписан на тему “scan” и извлекать из нее сообщения типа sensor\_msgs/LaserScan. Эти сообщения будут публиковаться мобильным роботом запущенным в Gazebo, на борту которого установлен лидар. Лидар сканирует окружение против часовой стрелки. Структура программы приведена на рисунке 14.

rosrun move\_turtlebot3 laser\_turtlebot3.py

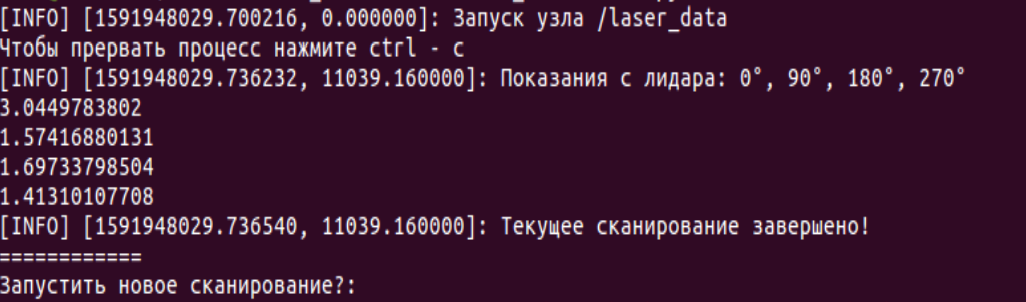


Рисунок 15 – Демонстрация работы программы

Как видно на рисунке 15, программа успешно запустилась и выводит необходимые нам данные на экран: расстояние до объекта напротив равно 3.0449 м, до объекта слева – 1.5741 м, до объекта сзади – 1.6973 м, и до объекта справа (зеленый бак) – 1.4131 м. Данные были получены путём среза элементов списка float32 ranges, шаг равен 90.

Следует отметить что, данный лидар имеет разную точность измерения при разных условиях, так например, в диапазоне (120 – 499 мм) погрешность измерения может составлять порядка 15 мм, и в диапазоне (500 – 3500 мм) – 25- 30 мм. Код программы приведен в приложении Б.

# 

# Узел avoid\_obstacle

Создадим узел avoid\_obstacle, который будет управлять движением мобильного робота Turtlebot3, используя данные с датчика расстояния.

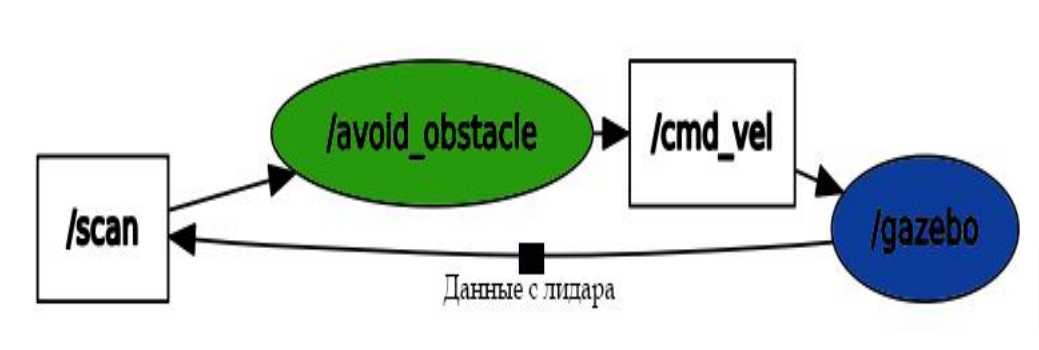


Рисунок 16 – Структура программы управления

Поясним то, что изображено на рисунке 16.

Виртуальная модель мобильного робота Turtlebot3 запущенная в симуляторе Gazebo подписана на тему /scan и публикует в нее данные с датчиков, установленных на борту робота, в том числе, данные (сообщения) с лидара типа sensor\_msgs /LaserScan.

Узел avoid\_obstacle подписан на тему /scan и следовательно, может извлекать данные из неё. Этими данными в данном случае являются данные с лидара, в которых хранится информация о расстоянии до объектов относительно робота. На основе этих данных программа решает какое значение установить для скорости перемещения робота. Далее эти значения публикуются в тему /cmd\_vel в виде сообщений типа geometry\_msgs/Twist, которые преобразуются в команды управления вращением приводов, и виртуальный робот Turtlebot3 Burger осуществляет перемещение в заданном направлении. Алгоритм программы получается довольно простым и может быть представлен в виде блок-схемы (рисунок 18). Код программы приведен на рисунке 19.

rosrun move\_turtlebot3 avoid\_obstacle.py

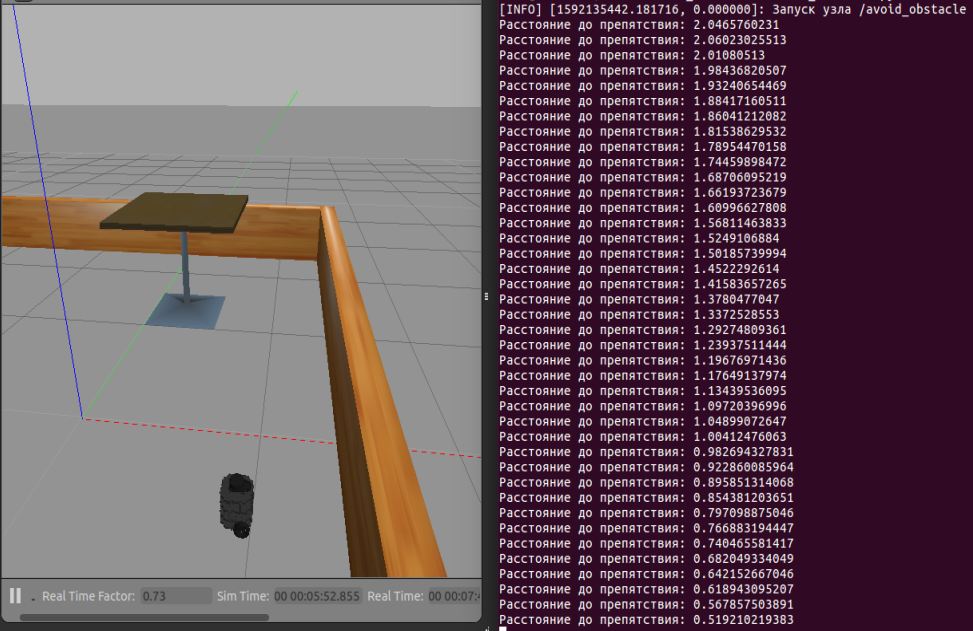


Рисунок 17 – Демонстрация программы avoid\_obstacle

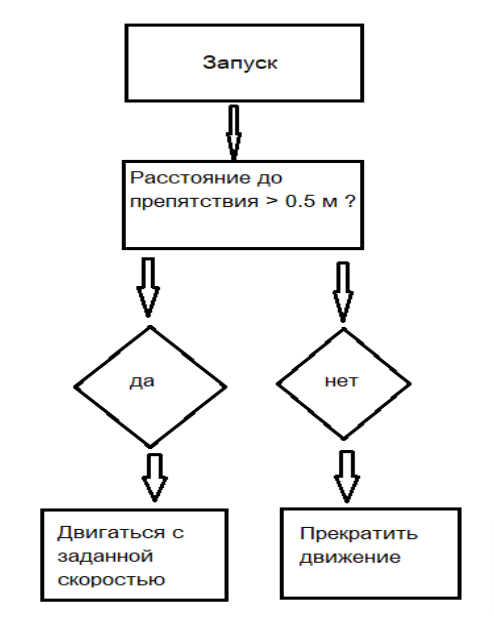


Рисунок 18 – Блок-схема программы avoid\_obstacle

Код программы может быть представлен в следующем виде:

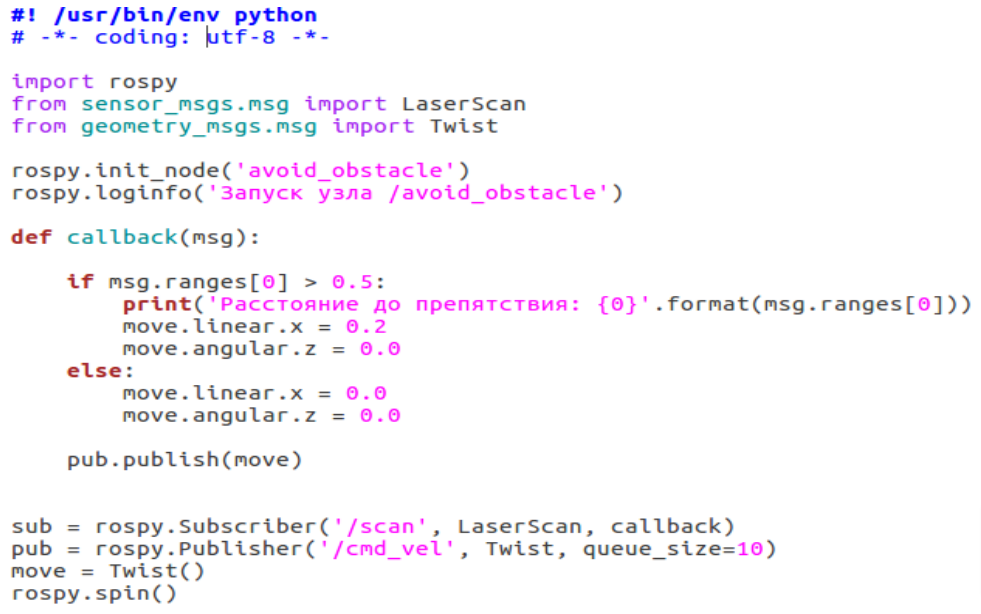


Рисунок 19 – Код программы avoid\_obstacle

# 

# Заключение

В ходе выполнения научно-исследовательской работы был проведён обзор и анализ робототехнического фреймворка ROS. Были изучены файловая система Linux, язык программирования Python, принцип взаимодействия узлов в системе ROS, способы отправки команд через терминал, а также инструменты визуализации RViz и 3D-симуляции Gazebo.

С помощью команд rosbash были изучены основные детали и принципы взаимодействия узлов в системе ROS для согласованного управления виртуальными роботами turtlesim и TurtleBot3.

Были рассмотрены этапы создания собственных программ в системе ROS. Описаны ключевые моменты использования платформы, её возможностей и инструментов.

Разработана программа для перемещения виртуального мобильного робота в заданную точку на плоскости. Предложены реализации программ для использования мобильного робота TurtleBot3. Далее эти программы могут быть усовершенствованы или использованы в качестве основы для решения задач навигации мобильного робота.

В результате изучения и анализа данного робототехнического фреймворка были сделаны следующие выводы:

ROS – это платформа значительно упрощающая процесс моделирования робототехнических систем, но вместе с тем достаточно сложная по своей архитектуре и организации. Таким образом, эта платформа подойдёт для решения серьёзных задач робототехники и их моделирования.

Развитая библиотека rospy позволяет значительно ускорить процесс разработки программ для роботов путём импорта необходимых классов и объектов.

В дальнейшем методический материал приведенный в данной работе может быть полезен всем начинающим знакомство с ROS.

# 

# Список использованных источников и литературы

1. Marwan Qaid Mohammed, L.C. Kwek, Shing Chyi Chua. Review of Deep Reinforcement Learning-Based Object Grasping: Techniques, Open Challenges, and Recommendations //
2. Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 552 с. //
3. Лаура Г., Лун К.В. Глубокое обучение с подкреплением: теория и практика на языке Python. — СПб.: Питер, 2022. — 416 с.: ил. — (Серия «Библиотека программиста»). //
4. «ROS Robot Programming» YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim: ROBOTIS Co., 2017 - 487 c
5. ROS википедия [электронный ресурс] — <http://wiki.ros.org/> //
6. Томас Бройнль. Встраиваемые робототехнические системы: проектирование и применение мобильных роботов со встроенными системами управления / Томас Бройнль: Институт компьютерных исследований, 2012. - 520 c. //
7. O’Kane, Jason. A Gentle Introduction to ROS / O’Kane, Jason: CreateSpace, 2013 – 166 c. //

# 

# Приложение А

*Установка пакета turtlebot3*

Перейдём в директорию, в которую будет загружен данный пакет:

cd ~/catkin\_ws/src

Загрузим пакет turtlebot3 с официального репозитория с помощью команды git:

git clone -b kinetic-devel https://github.com/ros/urdf.git

git clone https://github.com/ROBOTIS-GIT/turtlebot3

Введём следующую команды для сборки пакета в рабочем пространстве catkin:

cd ~/catkin\_ws$ catkin\_make

Перед тем как запустить симулятор gazebo, нужно будет добавить специальную строку в .bashrc.

.bashrc - это сценарий оболочки bash, который находится в домашнем каталоге пользователя. Он используется для сохранения и загрузки настроек терминала и переменных среды.

Для этого введем следующую команду в терминале:

gedit ~/.bashrc

и добавим следующую строку в конец этого файла:

export TURTLEBOT3\_MODEL=burger

Сохраним конфигурацию файла bashrc в текущем терминале:

source ~/.bashrc

# 

# Приложение Б

Laser\_data

