Оглавление

Введение

Робототехническое сообщество за последние годы добилось заметных успехов. Стали доступными надежные и недорогие аппаратные средства, начиная с наземных мобильных роботов и заканчивая квадрокоптерами и человекоподобными роботами. Еще более показательным является тот факт, что развитие программного обеспечения (ПО) позволило значительно повысить уровень автономности робототехнических систем (РТС).

Написание робототехнического ПО является сложной задачей. Основными причинами являются:

– *Сложность повторного использования кода*

Это связано с увеличением количества различных по структуре и функционированию робототехнических систем, обладающих совершенно различным ПО.

– *Сложность и большой объем кода ПО*

Код ПО должен содержать обширный набор подпрограмм, начиная с программного обеспечения на уровне драйверов и заканчивая сложной многокомпонентной системой управления.

– *Высокие требования к архитектуре программного обеспечения*

Очевидно, что требуемый для создания ПО современных РТС диапазон знаний выходит далеко за рамки возможностей любого отдельного разработчика, поэтому архитектура робототехнического ПО должна быть выстроена таким образом, чтобы интеграция отдельных частей кода была максимально простой.

Для решения этих проблем исследователи в области робототехники ранее создали большое разнообразие программных платформ (ПП) – фреймворков. Существующие ПП управляют сложным ПО и способствуют увеличению скорости моделирования и проведения экспериментальных исследований. Одной из наиболее совершенных и наиболее широко применяемых робототехнических ПП является «Robot Operating System» (ROS).

В пособии содержится руководство к выполнению лабораторной работы, в рамках которой рассматриваются основы работы с ПП ROS, раскрывается суть элементов архитектуры ROS, происходит знакомство с основными инструментами командной строки, предоставляемыми ROS, и развиваются навыки работы с ними.

Учебное пособие предназначено для студентов по направлению подготовки 15.03.06 «Мехатроника и робототехника», а также представляет интерес для научных сотрудников, инженеров, занимающихся созданием и применением робототехнических средств и интересующихся проблемами робототехники.

Список условных сокращений

ПО – программное обеспечение,

ПП – программная платформа,

РТС – робототехническая система,

IDE (Integrated Development Environment) – интегрированная среда разработки,

ROS (Robot Operating System) – робототехническая операционная система.

1. Краткие сведения о ROS

Robot Operating System – ПП, содержащая набор библиотек, протоколов обмена и инструментов, созданных для облегчения разработки ПО РТС.

Работа над ROS началась в 2007 году в Лаборатории искусственного интеллекта Стэнфордского университета для проекта «STAIR». В 2008 году разработка была продолжена в научно-исследовательском институте робототехники Willow Garage совместно с более чем двадцатью институтами.

Можно выделить две основные стороны ROS. Во-первых, ROS обеспечивает стандартные службы операционной системы (аппаратную абстракцию, низкоуровневый контроль устройств, реализацию часто используемых функций, передачу сообщений между процессами). Во-вторых, ROS предоставляет возможность управления набором поддерживаемых пользователями пакетов и библиотек, организованным в так называемый стек. Такие пакеты и библиотеки реализуют различные функции РТС – планирование движений, обработку информации датчиков и др.

Для понимания, чем же является ROS, важно отметить, чем ROS не является.

– ROS не является языком программирования

ПО на базе ROS обычно написано на языках C ++ и/или Python. Также существуют клиентские библиотеки для поддержки Java, Lisp и нескольких других языков.

– ROS не является только библиотекой

Хотя ROS действительно включает клиентские библиотеки, она также включает в себя (помимо всего прочего), центральный сервер (ros-мастер), набор инструментов командной строки, набор графических инструментов и систему сборки.

– ROS не является интегрированной средой разработки

ROS не предписывает какой-либо конкретной среды разработки. Рассматриваемая ПП может быть использована с наиболее популярными интегрированными средами разработки (IDE) в соответствии с личными предпочтениями разработчика. Можно пользоваться просто текстовым редактором и командной строкой, без какой-либо IDE.

Конечно, ROS не единственная ПП, которая предлагает перечисленные возможности. Но широкая поддержка робототехнического сообщества по всему миру обеспечивает постоянное расширение и совершенствование ROS, что делает эту платформу уникальной.

1.1. Основные элементы архитектуры ROS

ROS основана на архитектуре графов. Обработка данных происходит в узлах, которые могут получать и передавать сообщения между собой. Таким образом, основными элементами архитектуры ROS являются узлы, сообщения и механизмы передачи сообщений – темы, сервисы и действия.

Узлы (nodes) представляют собой процессы, в которых выполняются вычисления. ПО на базе ROS разрабатывается в виде модульной системы, состоящей, как правило, из множества узлов. В этом контексте термин «узел» является взаимозаменяемым с «программным модулем». Термин «узел» возникает из визуализации системы: при одновременной работе многих узлов систему удобно представить в виде графа, а процессы – в виде узлов этого графа.

Клиентские библиотеки ROS позволяют реализовывать узлы на различных языках программирования, а именно на Python (клиентская библиотека rospy) и C++ (клиентская библиотека rocpp). В рамках лабораторных работ будет использоваться язык C++.

Узлы могут обмениваться между собой информацией посредством передачи *сообщений* (messages). Сообщение является строго прописанной структурой типов данных. Поддерживаются стандартные примитивные типы данных и их массивы. Сообщения могут состоять из других сообщений, а также массивов других сообщений.

Узлы отправляют и получают сообщения путем публикации или подписки на *темы* (topics), а так же с помощью предоставления или использования *сервисов* (services) или *действий* (actions).

В первом случае связь между узлами происходит путем опубликования и считывания сообщения из темы. Узел, публикующий сообщения, называется *издатель* (publisher), а узел, принимающий сообщения, – *абонент* (subscriber). Тема определяется именем и типом передаваемого сообщения.

Одновременно могут существовать несколько издателей и абонентов по одной теме. Один узел может публиковать и/или подписываться на несколько тем. Как правило, издатели и абоненты не знают о существовании друг друга.

Сервисы являются ***вторым способом*** связи узлов между собой. Узел, предоставляющий сервис, называется *сервером* (server), а узел, использующий его, – *клиентом* (client). Сервисы позволяют клиентам послать *запрос* (request) сервису и получить от него *ответ* (response). Сообщение, формируемое из запроса и ответа, называют *сервисным запросом*. Сервер определяется именем и типом сервисного запроса (парой строго типизированных сообщений – один для запроса и второй для ответа).

Третьим способом связи узлов между собой являются *действия*. Узел, предоставляющий действия, называется *сервером действия* (action server), а узел, использующий его, – *клиентом действия* (action client). Действия, по аналогии с сервисами, позволяют клиенту посылать *задачу* (goal) серверу действий и получить от него *результат* (result). Отличием действий от сервисов является наличие *обратной связи* (feedback), посредствам которой клиент действий получает информацию от сервера действий в процессе обработки в нем данных для формирования ответа. Сообщение, формируемое из задачи, результата и обратной связи называют *запросом действия*. Сервер действий определяется именем и типом запроса действий (тройкой строго типизированных сообщений – для задачи, результата и обратной связи).

В отличие от темы, только один узел может предоставлять сервис или действие с конкретным именем. Второе отличие сервисов и действий от тем заключается в том, что в тему сообщения публикуются непрерывно с заданной частотой, а в случае вызова сервиса или действия запрос и ответ посылаются один раз.

1.2. ROS-мастер и запуск разработанных программ

Как было отмечено ранее, разработанное на базе ROS робототехническое ПО представляет собой набор небольших независимых программ (узлов), работающих одновременно и общающихся между собой с помощью сообщений. Часть ROS, обеспечивающая взаимодействие узлов, называется ROS-мастер. Для его запуска в окне терминала выполняется команда:

$ roscore

Здесь и далее наличие символа «$» в начале строки означает, что команду выполняют в окне терминала. При этом сам символ «$» в команду не входит.

Примерное содержание окна терминала после выполнения команды roscore представлено на рисунке 1.

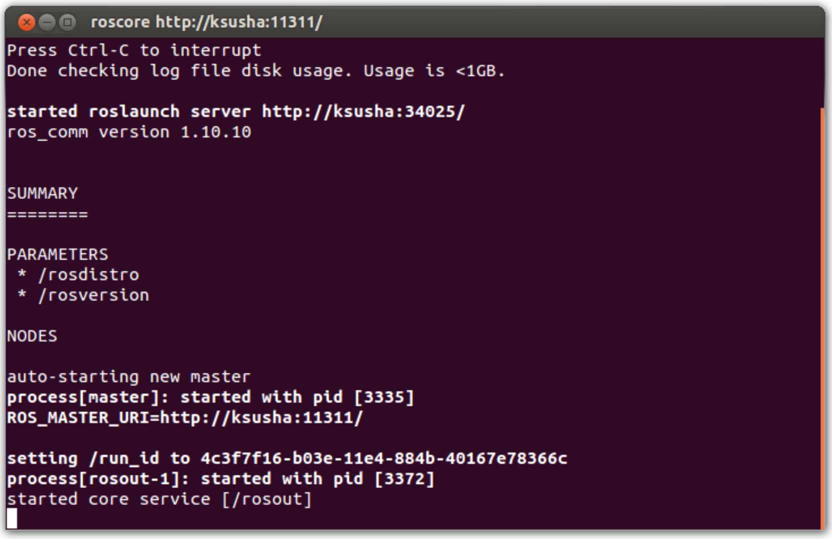

Рисунок 1 – Пример содержания окна терминала после выполнения команды roscore

ROS-мастер должен быть запущен до начала работы любых других узлов и продолжать работать на протяжении всего времени выполнения ROS-программ. Поэтому команда rescore выполняется в отдельном окне терминала, а для запуска узлов используются другие окна. Для остановки ROS-мастера в окне терминала, в котором он работает, выполняется команда Ctrl+C.

Для запуска узла используется команда rosrun. В качестве параметров rosrun принимает имя узла и имя пакета, содержащего данный узел. При этом указывать полный путь к пакету не нужно.

Например, чтобы начать выполнение узла с именем my_node, который находится в пакете с именем my_package, необходимо в окне терминала выполнить следующую команду:

$ rosrun my_package my_node

Для передачи значения value параметру с именем my_param, содержащемуся в узле с именем my_node, используется следующий синтаксис:

$ rosrun my_package my_node my_param:= value

1.3. Инструмент командной строки rostopic

Инструмент rostopic позволяет получать информацию о темах ROS и взаимодействоватьс ними. Ниже перечислены некоторые подкоманды rostopic:

$ rostopic bw /topic_name

– выводит пропускную способность темы с именем topic_name;

$ rostopic delay /topic_name

– возвращает задержку по времени темы с именем topic_name;

$ rostopic echo /topic_name

– выводит сообщения, опубликованные в теме с именем topic_name;

$ rostopic echo --offset /topic_name

– возвращает временную задержку сообщения от текущего времени в теме с именем topic_name;

$ rostopic echo -c /topic_name

– очищает окно терминала после каждой публикации сообщения из темы с именем topic_name;

$ rostopic echo /topic_name/field

– выводит поле field сообщений, опубликованных в теме с именем topic_name;

$ rostopic find msg-type

– находит темы, использующие тип сообщений msg-type;

$ rostopic hz /topic_name

– отображает частоту публикации сообщений в теме с именем topic_name;

$ rostopic info /topic_name

– выводит информацию о теме с именем topic_name;

$ rostopic list

– возвращает список доступных тем;

$ rostopic list /namespace

– возвращает список доступных тем в пространстве имен namespace;

$ rostopic list -p

– возвращает список узлов-издателей;

$ rostopic list -s

– возвращает список узлов-подписчиков;

$ rostopic pub /topic_name topic_type data

– один раз публикует данные data в тему с именем topic_name и типом сообщений topic_type;

$ rostopic pub -1 /topic_name topic_type data

– публикует данные data в тему с именем topic_name и типом сообщений topic_type на протяжении одной секунды;

$ rostopic pub –r rate /topic_name topic_type data

– непрерывно публикует данные data в тему с именем topic_name и типом сообщений topic_type с частотой rate Гц;

$ rostopic type /topic_name

– возвращает тип сообщений темы с именем topic_name.

В рамках выполнения лабораторной работы особенное внимание будет уделено командам rostopic echo и rostopic pub. Вторая команда, осуществляющая публикацию сообщений в тему непосредственно через командную строку, позволяет избежать необходимости создания отдельного узла-издателя для передачи сообщений узлу-абоненту.

1.4. Инструмент командной строки rosservice

Инструмент rosservice используется для подключения к клиентско-сервисной структуре программной системы ROS. Ниже перечислены некоторые подкоманды rosservice:

$ rosservice find service_type

– находит сервисы, использующие тип сообщений service_type;

$ rosservice list

– возвращает список доступных сервисов. Сервис является доступным, если его предоставляет запущенный узел. Узел стандартного вывода ROS – rosout, который создается вместе с запуском roscore, обладает сервисами rosout/get_loggers и rosout/set_logger_level. Таким образом, выполнение команды rosservice list без запуска какого либо узла, вернет результат:

$ rosservice list

/rosout/get_loggers

/rosout/set_logger_level

$ rosservice list /namespace

– возвращает список доступных сервисов в пространстве имен namespace;

$ rosservice list -n

– возвращает список узлов, предоставляющих сервисы;

$ rosservice info /service_name

– выводит информацию о сервисе с именем service_name;

$ rosservice node /service_name

– возвращает имя узла, предоставляющего сервис с именем service_name;

$ rosservice type /service_name

– возвращает типы и имена полей сервисного сообщения, используемого сервисом с именем service_name. Если rosservice type вернул std_srvs/Empty, то сервис является пустым (поля запроса и ответа какого сервиса не заполняются при его вызове);

$ rosservice args /service_name

– выводит поля запроса (аргументы) сервиса с именем service_name;

$ rosservice call /service_name service_args

– вызывает сервис с именем service_name с предоставлением значений полей запроса (аргументов) service_args и возвращает ответ. Если сервис с именем empty_service имеет пустой запрос, то его вызов происходит с помощью выполнения в окне терминала команды:

$ rosservice call empty_service

В рамках выполнения лабораторной работы особенное внимание будет уделено командам rosservice list, rosservice type и rosservice call.

1.5. Инструменты визуализации работы ROS rqt_graph

Инструмент rqt_graph создает динамический граф, описывающий связи узлов между собой. Для его запуска в новом окне терминала нужно выполнить:

$ rqt_graph

Результатом выполнения данной команды будет окно, представленное на рисунке 2. На нем изображены узлы и темы ROS в виде овалов и стрелок соответственно. В данном случае узлы node_0 и node_1 передают сообщения через тему с именем topic.

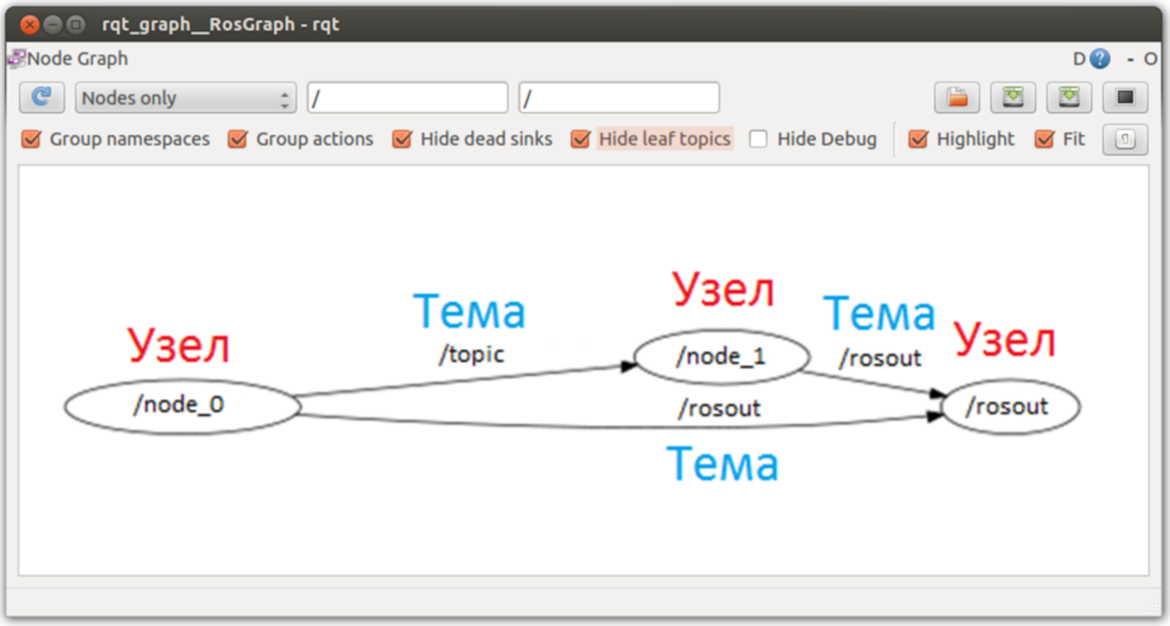

Рисунок 2 – Результат выполнения команды rqt_graph

Для ROS обязательным является наличие узла rosout – узла стандартного вывода ROS. Он создается вместе с выполнением roscore. При этом он является абонентом каждого созданного узла, принимая от них сообщения на тему rosout.

1.6. Инструменты визуализации работы ROS rqt_plot

Инструмент rqt_plot отображает график данных, опубликованных в выбранной теме. Для его запуска в новом окне терминала нужно выполнить:

$ rqt_plot

Результатом выполнения данной команды будет окно, представленное на рисунке 3. Интересующая тема выбирается в поле «Topic» и добавляется для отображения нажатием на кнопку «+».

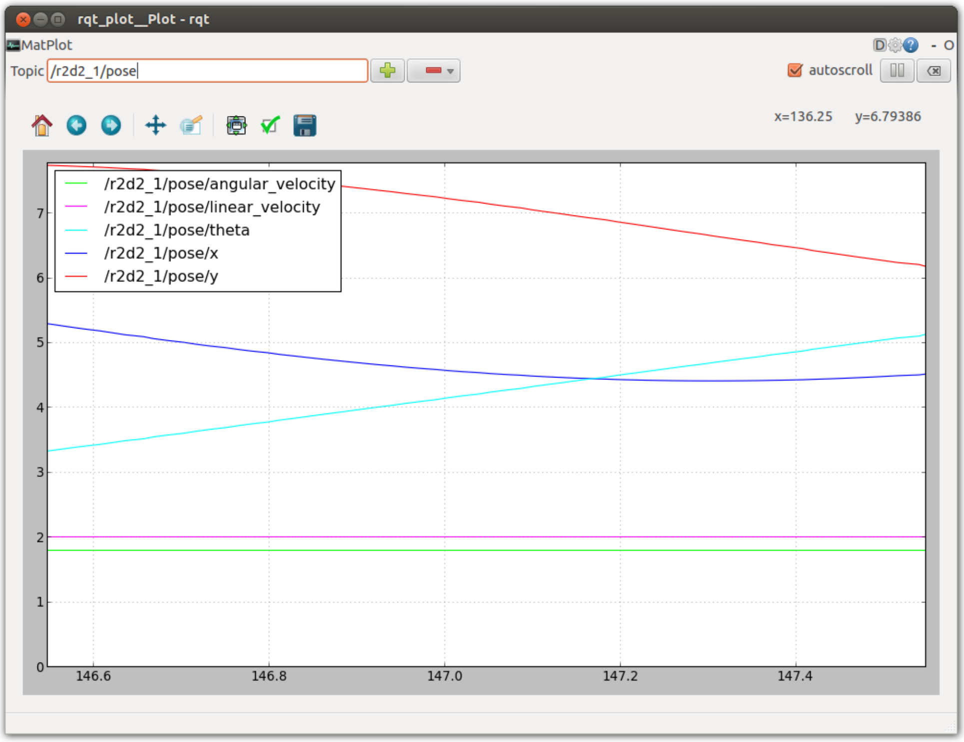

Рисунок 3 – Пример результата выполнения команды rqt_plot

Возможно одновременное отображение полей сообщений нескольких тем. На рисунке 3 изображены графики изменения данных из полей сообщений pose одной темы r2d2_1, которые обладают рядом полей: angular_velocity, linear_velocity, theta, x, y.

По сути, rqt_plot аналогична по принципу действия инструменту rostopic echo. Разница состоит в том, что rqt_plot представляет считываемую из сообщений информацию не в виде таблицы, как это делает rostopic echo, а в виде графика, что иногда удобнее для анализа протекающих в РТС процессов.

2. Последовательность действий при выполнении работы

Данная лабораторная работа знакомит с элементами архитектуры ПП ROS на примере пакета r2d2sim, реализующего движение робота R2-D2 по плоскости путем задания его перемещения и поворота. Пакет r2d2sim содержит два узла – визуализации (r2d2sim_node) и управления (r2d2_teleop_key).

Цель выполнения описанных ниже упражнений – добиться уверенного использования основных инструментов командной строки, предоставляемых ROS – roscore, rosrun, rostopic, rosservice, rqt_graph и rqt_plot.

Упражнение №1

1. Включите ПК, загрузите ОС Ubuntu.
2. Откройте новое окно терминала, кликнув левой клавишей мыши на значок на панели управления (слева) или используя комбинацию клавиш Ctrl+Alt+T.
3. В окне терминала выполните команду *$ roscore* (рисунок 1) и сверните окно (до окончания работы оно не понадобится).
4. Откройте второе окно терминала, кликнув правой клавишей мыши на значок на панели управления и выбрав в появившемся меню «Новый терминал» или используя комбинацию клавиш Ctrl+Alt+T.
5. Во втором окне терминала выполните команду *$ rqt_graph*.
6. Зафиксируйте имена узлов и тем, через которые общаются узлы, изображенные в окне инструмента rqt_graph, предварительно сняв галочку Hide Debug.
7. Во втором окне терминала откройте вторую вкладку. Для этого активируйте второе окно термина и используйте комбинацию клавиш Ctrl+Shift+T.

8. Во второй вкладке второго окна терминала запустите узел r2d2sim_node пакета r2d2sim, выполнив команду *$ rosrun r2d2sim r2d2sim_node* (рисунок 4).

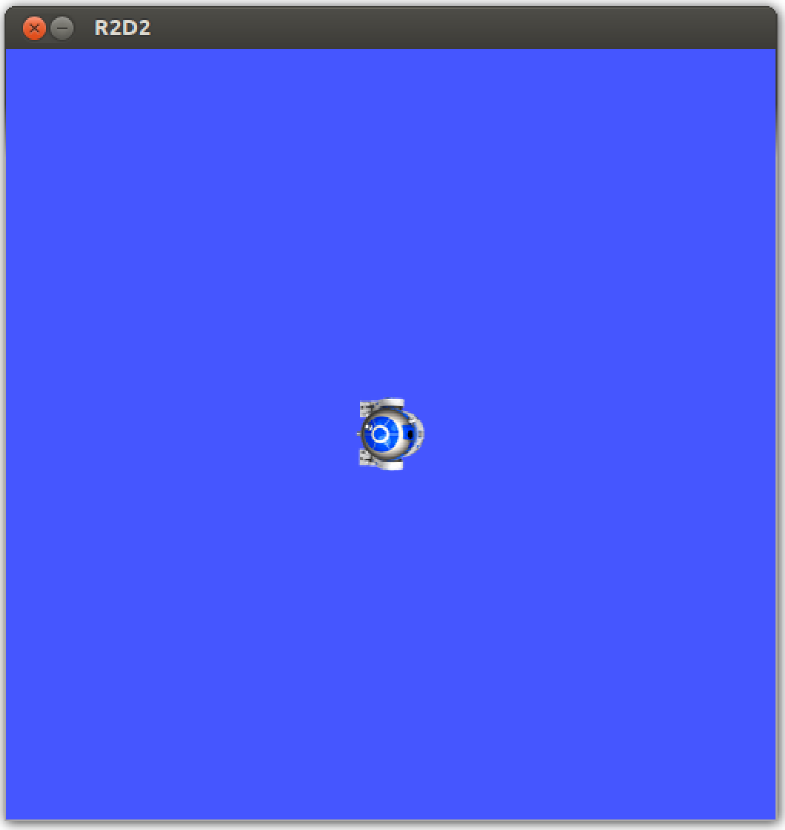

Рисунок 4 – Результат запуска узла r2d2sim_node из пакета r2d2sim

8. Зафиксируйте имена узлов и тем, через которые общаются узлы, изображенные в окне инструмента rqt_graph, предварительно сняв галочку Hide Debug.

9. Во втором окне терминала откройте третью вкладку, проделав действия, аналогичные действиям из пункта 7.

10. В третьей вкладке второго окна терминала запустите узел r2d2_teleop_key из того же пакета r2d2sim для управления перемещением робота R2-D2 с помощью стрелок клавиатуры. Для этого выполните команду *$ rosrun r2d2sim r2d2_teleop_key* (рисунок 5).

Если перемещение робота R2-D2 не происходит, сделайте активным вкладку окна терминала, в котором запущен узел r2d2_teleop_key, чтобы быть уверенными в том, что команды с клавиатуры поступают на вход узла.

Рисунок 5 – Результат управления роботом R2-D2 с клавиатуры

11. Зафиксируйте имена узлов и тем, через которые общаются узлы, изображенные в окне инструмента rqt_graph, предварительно сняв галочку Hide Debug.

12. Завершите работу всех узлов второго окна терминала, выполнив в каждой вкладке второго окна терминала команду Ctrl+C.

13. Закройте второе окно терминала.

Упражнение №2

1. Убедитесь, что в первом окне терминала по-прежнему работает узел roscore. Если это не так, выполните в этом окне команду Ctrl+C и повторите действия пункта 3 из упражнения №1.

2. Откройте второе окно терминала, кликнув правой клавишей мыши на значок на панели управления и выбрав в появившемся меню «Новый терминал» или используя комбинацию клавиш Ctrl+Alt+T.

3. Во втором окне запустите узел r2d2sim_node в пакете r2d2sim, выполнив команду *$ rosrun r2d2sim r2d2sim_node* (рисунок 4).

4. Ознакомьтесь со структурой команды rostopic pub, которая позволяет публиковать в терминале сообщения для узла r2d2sim_node. Пример команды: *$ rostopic pub -1 /r2d2_1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'*

Данная команда отправляет для узла r2d2sim сообщение, в котором задается движение с линейной скоростью в 2.0 и угловой скоростью 1.8. Рассмотрим каждый параметр отдельно: rostopic pub – команда опубликования сообщения в тему; -1 – опция (тире-один), позволяющая публиковать только одно сообщение и завершиться; /r2d2_1/cmd_vel – имя темы; geometry_msgs/Twist – тип сообщений темы; -- – опция (двойное тире), дающее указание парсеру терминала, что ни один из следующих аргументов не является опцией (необходима в случаях, когда аргументы могут использовать тире, например, отрицательные числа); '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]' – сообщение типа geometry_msgs/Twist, имеющие два элемента, представленных тройками чисел с плавающей точкой: линейная и угловая скорость. В рассматриваемом случае [2.0, 0.0, 0.0] становится значением линейной скорости вдоль оси x, а [0.0, 0.0, 1.8] — угловой вокруг оси z.

При выполнении данной команды робот будет двигаться в течение 1 секунды. Для задания непрерывного движения робота узлу r2d2sim_node требуется постоянный поток сообщений. Для создания такого потока необходимо использовать аргумент -r подкоманды rostopic pub, который задает частоту публикации сообщений.

5. Во втором окне терминала откройте вторую вкладку.

6. Задайте роботу непрерывное движение по окружности, для чего выполните команду *$ rostopic pub –r 1 /r2d2_1/cmd_vel geometry_msgs/Twist – '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'*.

7. Откройте новую вкладку второго окна и выполните команду *$ rqt_plot*.

8. Постройте график изменения положения робота на поле, выбрав для отображения инструмента rqt_plot поле pose сообщения из темы r2d2_1.

9. Завершите работу команд rqt_plot и rostopic pub.

10. Очистите поле и переместите робот R2-D2 в исходное положение. Для этого в свободной вкладке окна терминала найдите имя service_name необходимого сервиса командой *$ rosservice list,* определите его аргументы args для запроса, выполнив *$ rosservice type service_name*, и вызовите сервис командой *$ rosservice call service_name args.*

11. Закройте свободные вкладки окна терминала.

Упражнение №3

Требуется переместить робот R2-D2 в соответствии с номером варианта в точку поля из таблицы 1, используя команду rostopic pub. Информацию о текущем положении робота можно получить из поля pose сообщения темы r2d2_1.

№ варианта	1	2	3	4
Положение робота	[0.0, 0.0]	[2.5, 0.0]	[2.5, 2.5]	[2.5, 7.5]
№ варианта	5	6	7	8
Положение робота	[2.5, 11.0]	[7.5, 0.0]	[7.5, 2.5]	[7.5, 7.5]
№ варианта	9	10	11	12
Положение	[7.5, 11.0]	[0.0, 2.5]	[0.0, 7.5]	[0.0, 11.0]

робота				
№ варианта	13	14	15	16
Положение робота	[11.0, 0.0]	[11.0, 2.5]	[11.0, 7.5]	[11.0, 11.0]

Таблица 1 – Координаты итогового положения робота R2-D2

3. Контрольные вопросы

1. Что такое ROS?
2. Перечислите основные элементы архитектуры ПО на базе ROS.
3. Что такое узлы и сообщения?
4. Какие механизмы передачи сообщений между узлами существуют?
5. Чем передача сообщений через тему отличается от обмена информацией с помощью сервиса?
6. Чем сервис отличается от действия?
7. Перечислите инструменты командной строки, предоставляемые ROS.
8. Перечислите и опишите подкоманды команды rostopic.
9. Перечислите и опишите подкоманды команды rosservice.
10. Перечислите и опишите инструменты визуализации работы ROS.

Заключение

В пособии содержится руководство к выполнению лабораторной работы, в рамках которой рассматриваются основы работы с ПП ROS.

Раскрывается суть элементов архитектуры рассматриваемой ПП – узлов и сообщений. Описываются механизмы передачи сообщений между узлами – с помощью тем, сервисов и действий, приводится их сравнение между собой. Более подробно процесс создания узлов-издателей, узлов-подписчиков, узлов-серверов и узлов-клиентов будет рассмотрен в рамках последующих лабораторных работ.

В пособии приводится последовательность действий для создания и запуска программ, написанных на базе ROS.

Содержится описание основных инструментов командной строки, предоставляемых ROS – rostopic, rosservice, rqt_graph и rqt_plot, а последовательность действий, приведенная в пособии, направлена на формирование навыков уверенной работы с ними, которые будут основой для выполнения последующих лабораторных работ.

Список рекомендованной литературы

1. Lentin J. Mastering ROS for Robotics Programming / Packt Publishing, 2015.
2. Fernández E,. Crespo L.S., Mahtani A., Martinez A. Learning ROS for Robotics Program-ming - second edition / Packt Publishing, 2015.
3. Goebel P. ROS By Example / Lulu, 2013.
4. Goebel P. ROS By Example. Volume 2 / Lulu, 2014.
5. Jason M. O'Kane. A Gentle Introduction to ROS / CreateSpace Independent Publishing Plat-form, 2013.
6. Martinez A., Fernández E. Learning ROS for Robotics Programming / Packt Publishing, 2013.
7. Robot Operating System http://wiki.ros.org