Лабораторная работа №1
«Знакомство со средой ROS и её файловой системой»

Цель:
изучить архитектуру и освоить основные принципы работы среды ROS, необходимые для проектирования и моделирования роботов.

Задачи:

- ознакомиться с концепцией среды ROS;
- узнать альтернативные версии среды и способы установки;
- изучить иерархию программных и логических модулей среды ROS;
- выполнить практические примеры по управлению программными модулями.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Введение

Robot Operating System (ROS) – набор специализированных библиотек и инструментов с открытым исходным кодом, предназначенных для проектирования, моделирования и управления робототехническими аппаратами различной сложности. Изначально ROS была предложена и разработана компанией Willow Garage и Лабораторией искусственного интеллекта Стэнфордского университета.

Несмотря на то, что ROS не является полноценной операционной системой, ROS полностью выполняет её основные функции – обеспечивает низкоуровневое управление роботизированным и промышленным оборудованием, реализует абстракцию аппаратных средств и выполнение других сервисов, что позволяет пользователю общаться с роботом через специальное API на функциональном уровне. С системной точки зрения ROS является надстройкой над обычной Unix-подобной операционной системой (такой, как Linux или Mac OS X) со всеми необходимыми функциями и библиотеками для программирования роботов.

ROS также предоставляет инструменты и библиотеки для получения, создания, написания и выполнения программного кода в виде распределенных систем. При этом поддерживаются языки программирования Python, C++ и Lisp, также существуют экспериментальные библиотеки на Java и Lua, что обеспечивает программисту удобную среду разработки программного обеспечения робота.

Таким образом, среда ROS позволяет унифицировать и автоматизировать процесс разработки системы управления роботами путём использования типовых:

- функциональной и структурной схем;
- протоколов обмена сообщениями;
- драйверов устройств ввода информации (джойстики, датчики и сенсоры);
- драйверов устройств вывода информации (двигатели колёс, манипуляторы);
- системой логирования данных;
- алгоритмами и методами навигации;
- системой технического зрения.

Так же, ROS уже содержит вспомогательные библиотеки и приложения для преобразования систем координат в различных кинематических и навигационных задачах, а также визуализации различных данных.

Версии и установка ROS

ROS поддерживает Debian, Ubuntu, Linux Mint, OS X, Fedora, Gentoo, openSUSE, Arch Linux и Windows. Причём, в последнем случае – через виртуализацию Linux с помощью виртуальных машин (VMware, VirtualBox) или виртуальную подсистему Linux для Windows 10 (Windows Subsystem for Linux – WSL).

Несмотря на то, что ROS за время своего развития имела более десятка версий, наиболее устойчивыми и популярными являются последние три, которые устанавливаются на разные версии Ubuntu (таблица 1).

Таблица 1 – Соответствие версий операционной системы Ubuntu и среды ROS

Версия Ubuntu	**Версия ROS**	
Ubuntu 15.04 & 16.04	Kinetic	
Ubuntu 18.04	Melodic	
Ubuntu 20.04	Noetic	

Различные способы установки операционной системы Ubuntu приведены в приложении А. Процесс установки ROS включает два основных этапа: 1) скачивание и установка требуемой версии ROS и соответствующих дополнительных пакетов; 2) инициализация рабочего пространства ROS.

В процессе установки с официального сайта (приложение Б) у пользователя есть возможность выбрать одну из трёх актуальных версий ROS путём активации соответствующей кнопки / Kinetic / Melodic / Noetic в верху страницы, а также с помощью соответствующей команды выбрать тип требуемой инсталляции:

- *sudo apt install ros-noetic-ros-base* – устанавливает ядро ROS, которое содержит необходимый минимум пакетов для функционирования и компиляции новых элементов и библиотеки для коммуникации между модулями. Как правило, ROS-Base устанавливается на компьютеры, непосредственно управляющие роботами и/или интегрируемые в их бортовую систему правления;
- *sudo apt install ros-noetic-desktop* – устанавливает ROS-Base и дополнительные средства визуализации сенсорных данных и внешнего вида самих роботов. Указанные средства являются незаменимыми помощниками в процессе отладки и диагностики систем управления роботами;

- *sudo apt install ros-noetic-desktop-full* – устанавливает ROS-Desktop, оснащённую 2D/3D симулятором, позволяющим производить не только имитационное моделирование процессов функционирования роботов в сложных 3-мерных сценах, учитывающее законы механики движущихся тел, но и моделировать показания различных датчиков, включая видео камеры, локаторы и лазерные дальномеры.

На втором этапе происходит инициализация (приложение В) рабочего пространства ROS, которое заключается не только в выделении на жёстком диске компьютера определённых рабочих папок, но и создании окружения для создания и контроля программного кода.

Иерархия программных и логических модулей среды ROS

Парадигма ROS основана на сетевой организации различных процессов, которые можно представить в виде графа. Граф вычислений представляет собой одноранговую (англ. peer-to-peer, P2P – рус. «равный к равному») сеть процессов ROS, которые совместно обрабатывают данные. Непосредственно обработка данных происходит в узлах графа, которые могут обмениваться сообщениями между собой. Основными элементами графа являются узлы (nodes), мастер (master), сервер параметров (parameter server), сообщения (messages), сервисы (services), темы (topics) и мешки (bags). Все они представляют данные графа разными способами.

В качестве примера рассмотрим организацию сетевой структуры ROS для управления мобильным роботом с видеокамерой от джойстика (рис. 1). Система состоит из двух частей: мобильного робота на гусеничном ходу, на корпусе которого установлена видеокамера, и джойстика, с помощью которого оператор способен не только управлять роботом движением, но и видеть перед ним преграды. И робот, и джойстик имеют на своём борту встраиваемые компьютеры Raspberry Pi, на которых установлен ROS. Причём, ROS на джойстике запущен в режиме мастера, а ROS мобильного робота через беспроводную связь Wi-Fi (общий роутер) подключён к нему. В итоге оба компьютера работают в одном информационном пространстве.

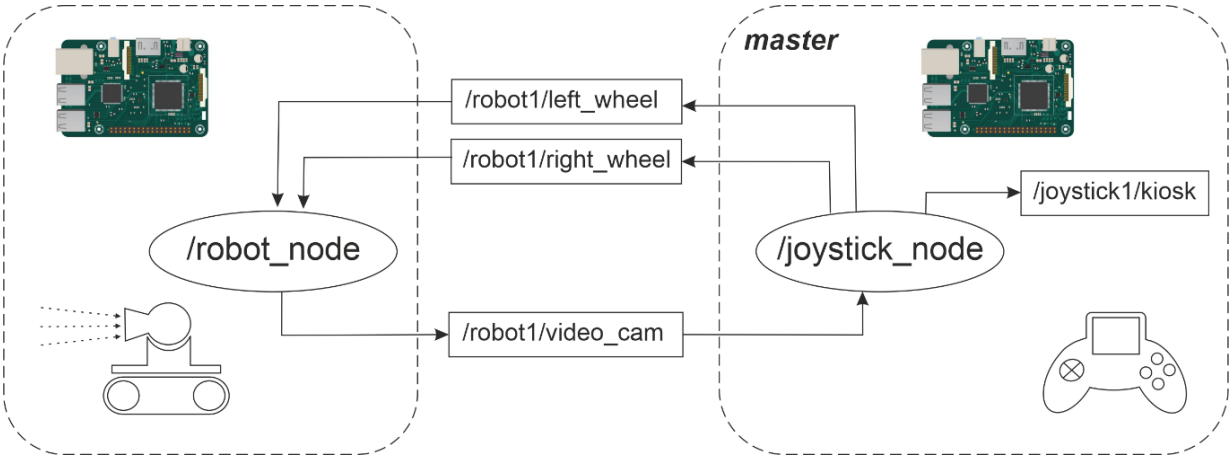


Рисунок 1 – Пример дистанционного управления мобильным роботом с помощью ROS

Узлы (nodes) – это исполнительные программные процессы, которые выполняют вычисления. ROS представляет собой модульную структуру, состоящую из множества узлов, где каждый выполняет одну небольшую функцию обработки данных. Например,

один узел принимает данные от дальномера, другой управляет двигателем, еще один выполняет планирование маршрута и т. п. Программа узла пишется с использованием библиотеки ROS, такой как roscpp или rospy.

Мастер (master) в ROS является назначаемой сетевой сущностью, которая обеспечивает регистрацию имён, направления взаимодействия и поиск всех частей графа вычислений. Без мастера узлы не смогут находить друг друга, обмениваться сообщениями и вызывать службы. Мастер ROS работает как служба имен в графе вычислений. Мастер хранит информацию о регистрации тем и сервисов для узлов ROS. Узлы общаются с мастером для передачи ему регистрационной информации. Также мастер сообщает узлам о других зарегистрированных (и вновь появившихся) узлах, и тогда узлы могут устанавливать соединения друг с другом. Узлы соединяются с другими узлами напрямую, мастер предоставляет только информацию о поиске, подобно DNS-серверу. Узлы, которые подписываются на тему, будут запрашивать соединения с узлами, публикующимися на эту тему. Протокол, по которому устанавливается соединение, называется TCPROS, он использует стандартные сокеты TCP/IP.

Сервер параметров (parameter server) является мультивариативным словарем общего пользования, доступный через сетевые программные интерфейсы API (англ. application programming interface, программный интерфейс приложения). Узлы используют этот сервер для хранения и получения параметров конфигурации системы во время работы. В настоящее время он является частью мастера.

Узлы могут общаться друг с другом с помощью сообщений (messages). Сообщение – это структура данных, включающая типизированные поля. Поддерживаются стандартные примитивные типы (integer, floating point, boolean и т. п.), а также массивы примитивных типов. Сообщения могут включать произвольно сложные структуры и массивы (подобно структурам языка C. Изучение структуры сообщений от сенсоров и других активных элементов системы управления конкретным роботом может многое сказать о их информационных потоках и, даже, назначении.

Сообщения в ROS организованы по темам (topics). Идея состоит в том, что узел, который намерен передавать информацию, будет публиковать сообщения по соответствующей теме или темам. А узел, который намерен получать информацию, должен быть подписан на интересующую тему или темы. Для одной темы может быть одновременно несколько узлов-источников (издателей) и узлов-получателей (подписчиков) информации. В целом издатели и подписчики не знают о существовании друг друга. Организацией взаимодействия между ними занимается мастер ROS. Логически можно представить тему как типизированную шину сообщений. Каждая шина имеет имя, и каждый узел может подключиться к шине для отправки или получения сообщений (если сообщение имеет правильный тип для этой темы). Недостатком работы с темами является то, что мастер не гарантирует передачу данных. Если по каким-либо причинам данные от издателя к подписчику не дошла, то в темах нет механизмов, чтобы это как-то зафиксировать и отреагировать. Для обеспечения передачи данных между ответственными узлами применяются сервисы.

Сервисы (services) являются более надёжным способом общения узлов друг с другом. Модель «издатель – подписчик» является гибкой коммуникационной средой, но в механизме публикаций сообщения нет понятия ответа и, соответственно, нет гарантии того, что это сообщение будет получено (и даже не гарантируется, что кто-либо подписывается на эти сообщения). Таким образом, модель «издатель – подписчик» не подходит для

взаимодействия между узлами в режиме «запрос-ответ», который реализуется через сервисы. Определены два типа сервисных сообщений: одно для запроса и одно для ответа. Узел, к которому обращаются, предоставляет сервис под определенным именем. Узел-клиент использует этот сервис для отправки сообщения запроса и ожидания ответа. Структура данных запроса и ответа хранится в файлах с расширением *.srv. Узел-клиент может выполнять постоянное подключение к сервису, что будет обеспечивать высокую производительность, однако это снизит надежность в случае изменений на узле, к которому выполнено подключение.

Мешки (bags) – это файлы, хранящие сообщения ROS с временными метками. Используются для отладки системы, когда необходимо собрать данные от датчиков, сохранить их и впоследствии воспроизвести многократно при отладке алгоритма. Например, мешки используются при разработке систем технического зрения для автономного вождения автомобилей. Если машину оснастить такими различным датчиками, как стереокамеры, лазерные лидары, инерциальные датчики и датчики поворота колёс, которые будут функционировать в системе управления на базе ROS, то после тестовых заездов разработчики могут воспроизвести всю картину движения через анализ записанных в мешки сенсорных данных.

Одной из основных особенностей хорошо спроектированной системы ROS является то, что узлы-потребители информации независимы от узлов, которые её производят. Эту архитектуру легко увидеть в модели коммуникации «публикации – подписки» через темы, которую использует ROS. Хороший абонентский узел должен работать постоянно, если нужные ему сообщения публикуются независимо от того, какой именно узел или узлы будут публиковать их. В случае, если требуется взаимодействие непосредственно между узлами, то реализуется модель «клиент – сервер» с использованием сервисов (рис. 2).

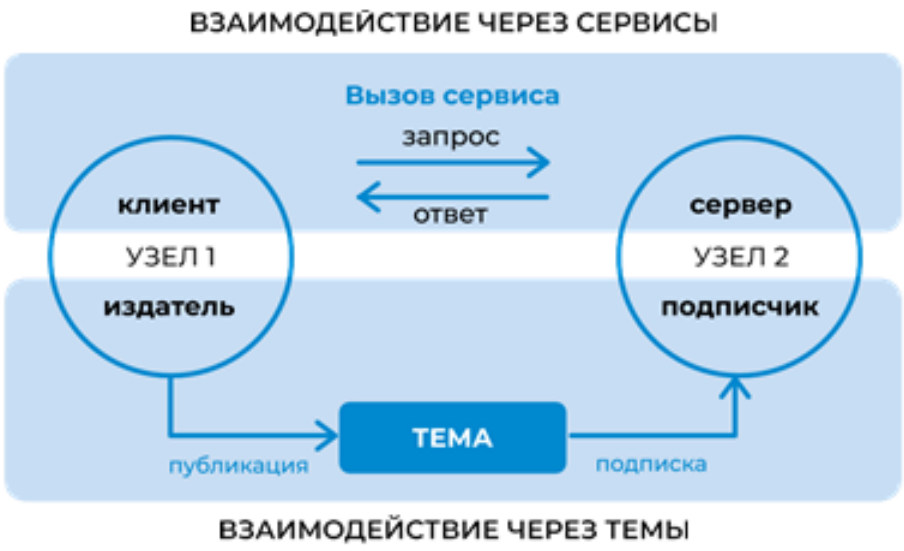


Рисунок 2 – Модель коммуникации узлов в среде ROS

Основные команды ROS

Команды ROS можно поделить по типам работы на команды с файловой системой и управляющими элементами ROS.

`roscd <packet_name>` – изменение директории с использованием имени пакета;

`rosls <packet_name>` – вывод содержимого директории пакета или списка пакетов;

Команды, работающие с управляющими элементами отображены в следующей таблице.

Таблица 2 –Команды работы с управляющими элементами

Команда	Действие	Примеры использования
roscore	запускает набор узлов и программ, которые требуются для работы ROS; при этом запускаются мастер ROS.	$ roscore
rosrun	Запускает исполняемый файл в произвольном пакете из любого места без необходимости указания полного пути до этого пакета	$ rosrun [package name] [executable name]
rosnode	выводит информацию об узлах ROS, которые запущены в системе.	$ rosnode <subcommand> [node name] Subcommand: list, info, kill, ping, cleanup
rostopic	позволяет получить информацию о темах в ROS	$ rostopic <subcommand> </topic name> Subcommands: list, echo, info, type, hz
rosservice	отображает информацию о сервисах и сообщениях, посланных в темы	$ rosservice <subcommand> [service name] Subcommands: args, call, find, info, list, type
rosparam	управляет данными на сервере параметров ROS	$ rosservice <subcommand> [service name] Subcommands: args, call, find, info, list, type
rosmsg	позволяет получить информации о типах сообщений	$ rosmsg <subcommand> [package name]/ [message type] Subcommands: show, type, and list

Описание встроенных графических приложений

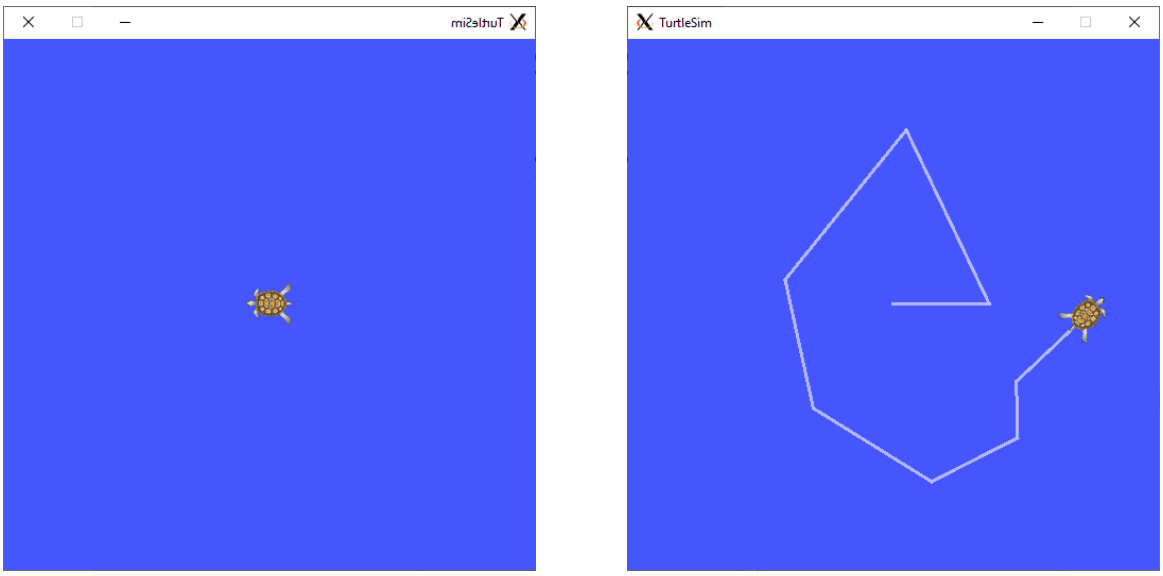
Для закрепления знаний и навыков, полученных в ходе данной лабораторной работы, воспользуемся встроенным в версии ROS-Desktop и ROS-Full пакетом TurtleSim, который предназначен для демонстрации парадигмы работы и основных походов к организации сетевого взаимодействия в среде ROS. Данный пакет выполняет, так называемую, функцию «Hello World!».

Чтобы запустить указанный симулятор требуется предварительно запустить ядро ROS командой roscore. После инициализации системы требуется открыть новое окно терминала и ввести команду

```
$ rosrun turtlesim turtlesim_node
```

В результате должно появиться отдельное графическое окно, в котором на голубом фоне изображена черепашка (рис. 3 *а*). При каждом новом запуске изображение самой

черепашки может меняться. Данное приложение основано по принципу тартл-графики, когда пользователь может задавать пошаговое управление движением и поворотом черепашки, а также подъёмом и опусканием «пера». Рабочее поле имеет форму квадрата с размером 11x11 см, отсчёт осей которого начинается с верхнего левого угла.



а) *б*)

Рисунок 3 – Примеры работы симулятора TurtleSim

Указанная команда запускает исполнительный узел turtlesim_node, включающий темы и сервисы, наименование и свойства которых можно узнать с помощью соответствующих команд rostopic и rosservice из таблицы 2. Например, с помощью команды

```
$ rostopic type /turtle1/cmd_vel
```

в терминале появится строка «Type: geometry_msgs/Twist». Это говорит о том, что тема /cmd_vel общается с помощью сообщений стандартного типа geometry_msgs/Twist, который сожержит в себе два вектора: три значения скорости линейного перемещения по осям *x*,*y* и *z* и три значения угловой скорости вращения вокруг осей *x*,*y* и *z*. Таким образом, задавая для узла-подписчика /cmd_vel значения линейных скоростей перемещений по осям *x* и *y*, а также вращение вокруг оси *z*, можно пошагово управлять движением черепашки. Так как симулятор TurtleSim является двумерным, то оставшиеся три переменные являются невостребованными. Рекомендуется по умолчанию в них прописывать нули, в противном случае они будут просто игнорироваться.

Задавать управляющие значения в уже запущенные темы и сервисы узла turtlesim_node можно с помощью клавиатуры. Для этого в новом терминале с помощью команды

```
$ rosrun turtlesim turtle_teleop_key
```

требуется запустить новый узел turtle_teleop_key, который при нажатии на клавиши «верх» и «вниз» заставляет черепашку совершать соответствующий шаг вперёд или назад, а при нажатии на клавиши «влево» и «вправо» – поворачивать в соответствующую сторону. В результате, черепашка начинает двигаться, оставляя после себя след. (рис. 3 *б*).

С помощью команды в новом окне терминала можно

```
$ rosrun rqt_graph rqt_graph
```

запуститься встроенное в ROS графическое rqt_grath, которое сгенерирует граф взаимодействия двух узлов через тему /cmd_vel.

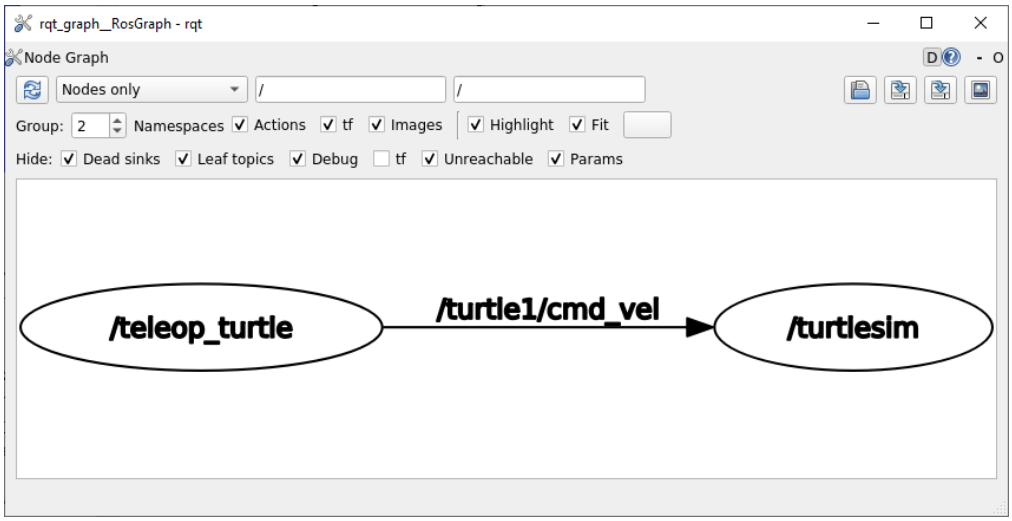


Рисунок 4 – Граф связей между узлами

Управлять черепашкой пошаговыми командами можно и с помощью командной строки. Например, команда

```
$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist '{linear: {x: 0.1, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

опубликует в тему /turtle1/cmd_vel значение двух векторов линейной [2.0, 0.0, 0.0] и угловой [0.0, 0.0, 1.8] скоростей перемещений ровно на один шаг, который выполняется 3 секунды. Если требуется обеспечить бесконечный поток указанных данных в тему, то требуется убрать флаг «-1». Вместо «-1» можно вставить ключ «-r RATE», где RATE – частота посылок в секунду, по умолчанию 10Гц. При необходимости можно остановить поток данных с помощью команды Ctrl+C.

Как уже говорилось выше, запущенный узел turtlesim_node имеет не только темы, но и сервисы. Например, сервис /turtle1/set_pen позволяет управлять цветом и шириной рисуемой черепашкой линией, а также подъёмом и опусканием самого «пера». Структуру данного сервиса можно просмотреть следующей командой

```
$ rosservice info /turtle1/set_pen
```

Чтобы активировать рассматриваемый сервис в терминале требуется ввести команду

```
$ rosservice call /turtle1/set_pen 255 0 0 5 1
```

где первые три цифры указывают на значение яркости красной, зелёной и синей составляющих цвета линии от 0 до 255, четвёртая представляет собой значение ширины линии, а последняя – опускание (0) и подъём (1) пера. Как уже указывалось выше, в отличие от тем, сервис работает по принципу регистра, т.е. после установки требуемых значений они сохраняются до тех пор, пока они не заменятся на другие.

ПРАКТИЧЕСКАЯ ЧАСТЬ

В качестве практического задания студентам требуется в графическом окне симулятора TurtleSim методом тартл-графики написать свои инициалы. Буквы должный быть написаны кириллицей, в верхнем регистре, трёх цветов и разной ширины.

Рекомендации:

- предварительно разбить начертания букв на составные прямые и нелинейные отрезки;
- так как каждый шаг выполняется 3 с, то для рисования линейных отрезков требуется подобрать соответствующие линейные скорости;
- для округленных элементов букв вручную подобрать соотношение линейной и угловой скоростей;
- для автоматизации вызова (активации) соответствующих тем и сервисов лучше создать исполняемый bash-скрипт (приложение Г).

Контрольные вопросы

1. Как выполняется взаимодействие между узлами в модели «издатель – подписчик»?
2. Чем отличается взаимодействие через тему от использования сервисов?
3. Для чего используется мастер ROS?
4. Какую команду необходимо обязательно выполнять в начале работы ROS?
5. С помощью какой команды можно вывести данные сообщений, публикуемых в теме?
6. С помощью какой команды можно вызвать сервис?
7. С помощью какой команды можно узнать тип сообщений?

Требования к отчету по работе

Отчет по выполнению лабораторной работы должен содержать:

1. титульный лист принятого в образовательной организации образца с названием лабораторной работы, фамилиями исполнителя и проверяющего работу;
2. цель и задания лабораторной работы;
3. результаты выполнения команд (листинги) по ходу выполнения работы и графы связей между узлами (скриншоты);
4. выводы по лабораторной работе.

Приложение А
Способы установки операционной системы Ubuntu

Три альтернативных способа установки операционной системы Ubuntu на ПК учащихся:

1. если на ПК учащихся установлена операционная система Ubuntu версии 15.10+, то этом случае необходимо на каждый ПК установить соответствующую версию ROS согласно таблице 1 и инструкции из приложения Б.

2. если на ПК учащихся установлена операционная система Microsoft Windows 7-8.1-10, то на каждом ПК следует установить программное обеспечение, реализующее технологию виртуализации VirtualBox. Создать и настроить виртуальную машину с операционной системой Ubuntu требуемой версии. После этого в виртуальной машине Ubuntu установить ROS соответствующей согласно инструкции из приложения Б.

3. если на ПК учащихся установлена операционная система Microsoft Windows 7-8.1-10 и нет возможности поставить виртуальную машину, то на каждом ПК требуется установить WSl 1 или 2 версии, установить консольное приложение Ubuntu (через магазин приложений), а также Windows XServer (например, VcXsrv) с предварительными настройками:

- при запуске XServer выключить native OpenGL
- для WSL1 требуется настроить дисплей

```
$ sudo strip --remove-section=.note.ABI-tag /usr/lib/x86_64-linux-gnu/libQt5Core.so.5
```

- в терминале Ubuntu ввести команды

```
$ export DISPLAY=:0.0
$ export LIBGL_ALWAYS_SOFTWARE=1
```

Приложение Б

Процесс установки среды ROS Noetic

Установите писок источников для скачивания

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

Установка ключей системы

```
$ sudo apt install curl # if you haven't already installed curl
$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

Проверка последних обновлений

```
$ sudo apt update -y
```

Инсталлирование полной версии ROS Noetic

```
$ sudo apt install ros-noetic-desktop-full
```

Инсталлирование специфических пакетов

```
$ sudo apt-get install ros-noetic-ros-control ros-noetic-ros-controllers
```

Добавление пространства имён установленной версии в Linux

```
$ source /opt/ros/noetic/setup.bash
```

Установка пакетов для возможности генерации собственных пакетов

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

Инициализация rosdep

```
$ sudo apt install python3-rosdep
$ sudo rosdep init
$ rosdep update
```

Приложение В

Инициализация рабочего пространства ROS

Переход в домашнюю папку

```
$ cd ~
```

Создание вложенной папки src, в которой будут храниться исходники будущих ROS-проектов. Имя папки catkin_ws не обязательное, но общепринятое и рекомендуемое

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

Переход в корневую папку будущих ROS-проектов

```
$ cd ~/catkin_ws
```

Инициализация рабочего пространства ROS

```
$ catkin_make
```

Настройка окружения

```
$ source ~/catkin_ws/devel/setup.bash
```

Последнюю команду требуется вводить при открытии каждого нового терминала.

Приложение Г

Процесс написания исполняемых bash-скриптов.

Bash-скрипты являются чрезвычайно мощным и полезным компонентом для разработки. С их помощью можно сократить короткие повторяющиеся задачи к однострочному вызову функции. А многие длинные команды могут быть объединены в один исполняемый код. Bash доступен практически во всех версиях Linux и не требует отдельной установки.

Скрипт или как его еще называют – сценарий, это последовательность команд, которые по очереди считывает и выполняет программа-интерпретатор, в нашем случае это программа командной строки bash. Скрипт – это обычный текстовый файл, в котором перечислены обычные команды, которые обычно вводятся вручную, а также указана программа, которая будет их выполнять. Загрузчик, который будет выполнять скрипт не умеет работать с переменными окружения, поэтому ему нужно передать точный путь к программе, которую нужно запустить. А дальше он уже передаст скрипт этой программе и начнется выполнение.

Список доступных оболочек можно проверить, введя следующую команду:

```
$ cat /etc/shells
```

В зависимости от дистрибутива Linux в командной строке появится список имеющихся в текущей ОС командных оболочек. Например, в Ubuntu 20.04, установленной в WSL1 полученный список будет иметь следующий вид:

```
/bin/sh
/bin/bash
/usr/bin/bash
```

Убедившись, что командная оболочка bash присутствует в вашей системе, следует создаст и перейти в директорию, где будет выполняться первая лабораторная работа:

```
$ mkdir -p ~/catkin_ws/src/Lab_01
$ cd ~/catkin_ws/src/Lab_01
```

В любом из текстовых редакторов требуется создать текстовый файл. Например, с помощью встроенного редактора Nanщ создадим текстовый файл с именем mylab01 и без расширения:

```
$ nano mylab01
```

В открывшемся окне в первой строчке после специальной комбинации двух символов “#!”следует указать, с помощью какой командной оболочки будет выполняться данный скрипт. Далее символ “#” может быть использован в скрипте для выделения комментариев.

```
#!/usr/bin/bash
```

Следующие строки должны содержать команды ROS, необходимые для выполнения поставленной в лабораторной работе задачи. Комбинацией клавиш Ctrl+X и Y в созданный документ будет закрыт с охранением введённого текста. Хтобы указанный скрипт сделать исполняемым файлом командой

```
$ chmod +x mylab01
```

требуется в файловой системе Linux изменить его права доступа.

После этого, с помощью следующей команды можно запустить указанный скрипт на выполнение:

```
$ ./mylab01
```