Занятие № 2: Детальное представление о среде

По завершении занятия студент должен:

- уметь строить иерархию взаимодействия нод и топиков у готовой программы
- уметь писать ноды, обменивающиеся собственными типами сообщений

Содержание

- демонстрация работы rqt_graph
- подробное описание CMakeLists.txt и package.xml

Представление связи node-topic в виде графа

Во всех учебниках по ROS знакомство с понятиями Node, Topic, Message начинают с примера, который называется “turtle_sim”. Работа с этим пакетом очень наглядна и помогает разобраться в том, как происходит взаимодействие между нодами.

В первую очередь необходимо помнить, что ROS - это Robotics Operation System, её основное предназначение в упрощении работы с роботами. Робот - это совокупность узлов, каждый из которых принимает какие-то данные, обрабатывает их и передаёт другому узлу. В ROS нода - это абстракция, которую можно сравнить с таким узлом робота. Физически нода - это поток, выполняющийся в системе и обменивающийся сообщениями с другими нодами.

Запуск нод в ROS осуществляется при помощи команды rosrun. Синтаксис употребления этой команды выглядит так:

```
rosrun <пакет> <нода> [<имя параметра>:=<значение>]
```

Причём в графе <нода> указывается не произвольное имя новой ноды, а класс-тип нод, экземпляр которого нужно запустить. Непосредственно имя ноды устанавливается через параметры.

Прежде, чем запускать эту команду, необходимо запустить ядро ROS. Поскольку ноды не существуют сами по себе, а взаимодействуют друг с другом, им необходимо окружение. В случае с настоящим роботом роль этого окружения играет операционная система этого робота. В нашем же случае в первую очередь необходимо вызвать команду

```
roscore
```

Эта команда будет выполняться в терминале, из которого она была вызвана. Поэтому для дальнейшей работы необходимо открыть ещё одно окно терминала.

Как только запущено окружение, можно посмотреть какие ноды запустились по умолчанию. Это можно сделать, выполнив команду

```
rosnode list
```

И на данном этапе в консоли появится следующий вывод:

```
/rosout
```

Это означает, что по умолчанию вместе с roscore запускается одна нода с именем rosout. Причём знак слеш “/” означает, что эта нода находится в глобальном поле имён и используется без префикса-имени пакета. Нода /rosout отвечает за вывод сообщений на экран. Безусловно в компьютерной симуляции роль /rosout может выполнить std::cout, но в реальных условиях у робота не будет iostream.h или stdio.h, поэтому в ROS для вывода сообщений создана настраиваемая нода /rosout. По любой ноде можно узнать, какие сообщения она посылает и на какие подписана. Для того, чтобы узнать информацию о ноде наберите

```
rosnode info <имя ноды>
```

То есть в данном случае

```
rosnode info /rosout
```

В ответ будет получено

```
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]
Subscriptions:
 * /rosout [unknown type]
Services:
 * /rosout/set_logger_level
 * /rosout/get_loggers
```

Разберёмся по порядку, что представляет из себя информация, которая была получена в ответе. Итак, первая строка представляет имя ноды, информацию о которой запрашивалась. Далее идёт список топиков, в которых эта нода публикует сообщения. Топик - абстракция однонаправленной коммуникации. /rosout посылает сообщения в топик /rosout_agg типа rosgraph_msgs/Log. Сперва это может показаться странным, что rosout - нода, предназначенная только для вывода сообщений на экран, ещё публикует какую-то информацию ещё куда-то. Но если прикинуть ситуацию, в которой появляется необходимость сохранять всё, что пишется на экран ещё и в файл, становится понятно, зачем это было сделано. Ведь, допустим, если есть несколько нод, которые что-то пишут на экран, то для осуществления логирования этой всей информации на экран, необходимо вручную для каждой ноды указывать дополнительный топик, куда они будут клонировать информацию. Для того, чтобы избежать ручной работы, было принято решение создать дополнительный топик, куда будут передаваться все сообщения, полученный нодой /rosout.

Далее в списке информации о ноде следует список нод, на которых она подписана. В данном случае /rosout подписан на /rosout. Пусть вас не смущают одинаковые имена, ведь в одном случае речь идёт о ноде, а во втором - о топике. Тут сразу возникает следующий вопрос: если /rosout подписан на /rosout и получает всю информацию через этот топик, то зачем нужен топик /rosout_agg, который был рассмотрен выше? Ответ прост: в топик /rosout_agg информация поступает уже отформатированная с указанием, кто, когда и откуда присылал сообщение для вывода на экран. Следующий вопрос: почему тип сообщения, на которые подписан /rosout помечен как “unknown type”? Дело в том, что топик фактически создаётся тогда, когда создаётся нода-publisher в этот топик. Нода-subscriber не создаёт топик, а лишь упоминает о нём, но не представляет какого типа сообщения в этот топик будут поступать. Этот тип будет конкретизирован как только будет создана хотя бы одна нода оправляющая сообщения в этот топик.

Графически описанные топики и ноды представлены на рисунке ниже.

Рис. 3.1. Связь ноды /rosout и её топиков.

При отображении топиков и нод принято соглашение, что эллипсами отображаются ноды, а прямоугольниками - топики. Стрелочками показывается поток сообщений из топика к ноде или наоборот. Таким образом, на рисунке 3.1 показано, что нода /rosout подписана на сообщения из топика /rosout и публикует свои сообщения в топик /rosout_agg.

И последнее о чём следует сказать про информацию о ноде - это список сервисов, которые эта нода представляет. Сервис - это аналог топика, только в отличие от последнего, сообщение сервиса двунаправлено и ожидает ответа на запрос. Сервис можно рассматривать как команду с некоторым откликом, которую одна нода просит выполнить у другой ноды. Подробнее о сервисах и их использовании будет рассказано позднее. Теперь вернёмся к turtle_sim и в новом окне запустим команду

```
rosrun turtlesim turtlesim_node
```

Что означает, что будет запущена нода типа turtlesim_node из пакета turtlesim. Имя ноды в данном случае не указывалось и оно присвоится автоматически. Теперь, если в новом терминале ввести команду

```
rosnode list
```

в ответ будет получено следующее:

```
/rosout
/turtlesim
```

Таким образом создалась ещё одна нода с именем turtlesim и определённая в глобальном поле имён, о чём говорит предшествующий символ слеш “/”. Если в новом терминале вновь будет выполнена команда

```
rosrun turtlesim turtlesim_node
```

То создастся новая нода с тем же именем /turtlesim. Это вызовет аварийную остановку уже созданной ноды:

```
[ WARN] [1471851936.261860979]: Shutdown request received.
[ WARN] [1471851936.261924457]: Reason given for shutdown: [new node
registered with same name]
```

Для того, чтобы указать имя ноды, при её создании необходимо присвоить значение переменной __name (с двумя символами нижнего подчёркивания). Таким образом, чтобы создать ноду с именем “turtle_node” необходимо выполнить команду

```
rosrun turtlesim turtlesim_node __name:=turtle_node
```

При запуске ноды turtlesim_node появится окно синего цвета с черепашкой в центре. Причём от запуска к запуску внешний вид черепашки может меняться. Её вид выбирается произвольно. Теперь, если запросить информацию о новой ноде

```
rosnode info /turtle_node
```

то будет получен следующий ответ:

```
Node [/turtle_node]
Publications:
 * /turtle1/color_sensor [turtlesim/Color]
 * /rosout [rosgraph_msgs/Log]
 * /turtle1/pose [turtlesim/Pose]
Subscriptions:
 * /turtle1/cmd_vel [unknown type]
Services:
 * /turtle1/teleport_absolute
 * /turtle_node/set_logger_level
 * /turtle_node/get_loggers
 * /reset
 * /spawn
 * /clear
 * /turtle1/set_pen
 * /turtle1/teleport_relative
 * /kill
```

Здесь видно, что turtlesim_node представляет большое количество сервисов. Но нам сейчас более интересными являются топики, на которые эта нода подписывается и в какие публикуется. Подписывается она на топик /turtle1/cmd_vel. По сути это - топик команд перемещения в него будут посылаться команды перемещения черепашки. Поместить сообщение в топик можно с клавиатуры командой:

```
rostopic pub <имя топика> <тип сообщения> <сообщение>
```

В /turtle1/cmd_vel посылаются сообщения типа geometry_msgs/Twist. В этом можно убедиться, прочитав документацию по turtlesim-у.

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist “[2, 0, 0]” “[0, 0, 1]”
```

Это передвинет черепашку на двойку в местных координатах вперёд и повернёт на один радиан против часовой стрелки. Остальные числа в квадратных скобках могут быть произвольными. Нода /turtle_node игнорирует их и использует только первое число, как величина пути вперёд и последнее, как величина поворота в радианах. Чтобы не запоминать формат сообщения, можно после типа сообщения дважды нажать клавишу tab. Это выпишет имена полей и рядом начальные значения, которыми они инициализируются. Далее их можно изменить:

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear:
 x: 0.0
 y: 0.0
```

```
 z: 0.0
angular:
 x: 0.0
 y: 0.0
 z: 0.0"
```

Данный формат говорит о том, что сообещние типа geometry_msgs/Twist представляет собой две структуры названные “linear” и “angular”, каждая из которых состоит из трёх вещественных переменных. Как уже было указано ранее, /turtle_node использует только координаты linear.x и angular.z. Для прерывания исполнения команды используйте ctrl+c.

Теперь, если мы посмотрим на граф нод и топиков, он будет выглядеть, как на рисунке ниже.

Рис. 3.2. Две ноды /turtle_node и /rosout и топики между ними.

Теперь создадим ноду-publisher-а в топик /turtle1/cmd_vel. Например ноду из пакета turtle_sim называемую turtle_teleop_key. Запустим её в новом терминале так как, как вы успели заметить, каждая нода - это отдельный процесс: $ rosrun turtlesim turtle_teleop_key Теперь, если оставить активным терминал, в котором была запущена эта нода, и нажимать клавиши ←↑↓→ это приведёт к движению черепашки в окне turtle_node. Теперь, связь нод и топиков будет выглядеть так, как показано на рисунке ниже.

Рис. 3.3. Ноды /teleop_turtle, /turtle_node и /rosout и топики между ними.

Наконец, чтобы отобразить связи существующих нод и топиков можно запустить ещё графовую ноду rqt_graph из пакета rqt_graph командой $ rosrun rqt_graph rqt_graph Эта нода собирает всю информацию о запущенных нодах и созданных топиках и визуализирует их связи. При первичном запуске появиться окно вида, представленного на рисунке ниже.

Рис. 3.4. Иллюстрация работы ноды типа rqt_graph.

Если убрать галочки Namespaces (которая отображает пространство имён нод и тпоиков), Dead skins, Leaf topics и Debug, то можно увидеть все существующие ноды и топики. В этом случае также будет показана нода созданного графового построителя.

Рис. 3.5. Иллюстрация работы ноды типа rqt_graph с полным представлением.

На рисунке 3.4 можно видеть набор запущенных важных нод и топиков между ними. В этом случае не указывается ни нода /rosout, ни нода графа /rqt_gui_py_node, ни их топики, а также не указываются “висячие” топики - в которых никто не пишет и из которых никто не читает. На рисунке 3.5 показаны все запущенные ноды и топики. Этот граф иллюстрирует всё состояние робота, но является нагруженным для выделения основных компонентов.

- Лабораторная работа № 2 : Обмен сообщениями своего типа