LR_2_Prep

Создание пакета и сообщений

Создание пакета

Чтобы создать свой пакет, нам понадобится ранее созданное окружение catkin.

1. Создаем новый пакет при помощи команды `catkin_create_pkg` после чего мы можем указать зависимости (либо указать их потом)

Чтобы создать новый пакет

```
cd src
```

```
catkin_create_pkg my_super_robot_controller rospy std_msgs
```

```
cd ../
```

```
catkin build
```

> Всегда работаем в папке **src** и пакеты тоже создаем внутри него!

2. Внутри папки нашего пакета проверяем подпапки scripts, src и файлы **CMakeLists.txt и package.xml**

💡 Файл **CMakeLists.txt** является входом для CMake системы сборки пакетов. Любой совместимый с CMake пакет содержит один или больше CMakeLists.txt файлов, которые описывают как собрать код и куда установить. CMakeLists.txt файл, используемый catkin проектом является стандартным CMakeLists.txt файлом с некоторыми дополнительными ограничениями.

Создание паблишера и сабскрайбера в одном файле

Внутри нашего пакета заходим в папку *scripts* и создаем файл *super_node.py*

Пропишем внутри файла следующий программный код:

```
#!/usr/bin/env python3

import rospy

from turtlesim.msg import Pose
from geometry_msgs.msg import Twist

#from geometry_msgs.msg import Twist # geometry_msgs must be in package.xml also!!!
```

```
def pose_callback(pose):
    cmd = Twist()
    if pose.x > 9.0 or pose.x < 2.0:
        cmd.linear.x = 1.0
        cmd.angular.z = 1.4
    else:
        cmd.linear.x = 2.0
        cmd.angular.z = 0.0
    publisher.publish(cmd)
    #rospy.loginfo('('+ str(msg.x) + ", " + str(msg.y) + ")")

if __name__ == '__main__':
    rospy.init_node("turtle_super_node")
    rospy.loginfo("Node started...")
    # rate = rospy.Rate(2)
    subscriber = rospy.Subscriber("/turtle1/pose", Pose, callback=pose_callback)
    publisher = rospy.Publisher("/turtle1/cmd_vel", Twist, queue_size=10)
    rospy.spin()
```

делаем этот файл исполняемым при помощи `chmod +x my_node.py`

Запускаем наш исполняемый файл

```
rosrun my_robot_controller my_node.py
```

Создание пользовательских сообщений

Ключевые шаги:

1. Сначала создадим папку, в которой будут лежать типы сообщений:

`mkdir msg`

2. Создадим сам файл сообщения

`echo "int64 num" > msg/Num.msg`

В этом типе сообщении содержится всего лишь одна строка с одним числовым значением, конечно же можно его расширить отредактировав только что созданный файл, например на следующее содержание.

```
string first_name
string last_name
uint8 age
uint32 score
```

Кроме этого, нам нужно убедиться, что файлы msg преобразованы в исходный код для C++, Python и других языков. И чтобы этот было возможным, нужно выполнить следующие

шаги.

3. Откройте package.xml и убедитесь, что эти две строки находятся в нем и не закомментированы:

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

> 💡 Обратите внимание, что во время сборки нам нужно «message_generation», а во время выполнения нам нужно только «message_runtime».

4. Добавьте зависимость message_generation к вызову find_package, который уже существует в вашем CMakeLists.txt, чтобы вы могли генерировать сообщения. Вы можете сделать это просто добавив message_generation в список COMPONENTS, чтобы он выглядел следующим образом:

```
find_package(catkin REQUIRED COMPONENTS
   roscpp
   rospy
   std_msgs
   message_generation
)
```

5. Также убедитесь, что вы экспортировали зависимость message runtime.

```
catkin_package(
  ...
  CATKIN_DEPENDS message_runtime ...
  ...)
```

6. Найдите следующий блок кода:

```
# add_message_files(
#   FILES
#   Message1.msg
#   Message2.msg
# )
```

Раскомментируйте его, удалив символы #, а затем замените подставку в файлах Message*.msg своим файлом .msg, чтобы он выглядел следующим образом:

```
add_message_files(
  FILES
  Num.msg
)
```

Добавляя .msg файлы вручную, мы должны убедиться, что CMake знает, когда необходимо переконфигурировать проект после добавления других файлов .msg.

7. Теперь мы должны обеспечить вызов функции generate_messages() .

Необходимо найти следующие строки и тоже раскомментировать:

```
# generate_messages(
#   DEPENDENCIES
#   std_msgs
# )
```

8. Выполним catkin build чтобы наши изменения применились
9. После этого, необходимо убедиться, что ROS его видит с помощью команды `rosmsg show`.

Задание

1. Изменить только что созданный тип сообщений таким образом, чтобы он содержал только координаты x, y. (название на ваше усмотрение, но только не Num)
2. Необходимо написать программный код для узла, который будет получать Pose черпашки из пакета **turtlesim** и публиковать в топик **position2d** сообщения с координатами x,y
3. Дл наглядности работы в отдельном терминале запустить вывод из топика position2d при помощи `echo`.

Запуск нескольких узлов одновременно

1. Для этого предусмотрены файлы .launch
2. Создадим папку и соответствующий файл

`mkdir launch`

`touch launch/my_setup.launch`

`code launch/my_setup.launch`

3. Укажите содержимое файла

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
<!-- <node name="my_super_node" pkg="my_robot_controller" type="super_node.py" output="screen"> </node>-->
```

```
<node name="my_super_node" pkg="my_robot_controller" type="super_node.py"></node>
<!-- Start here the turtlesim!!! -->
</launch>
```

4. Запустите launch файл

   `roslaunch my_robot_controller my_setup.launch`

Задание

Отредактируйте `my_setup.launch` таким образом, чтобы одновременно с **my_robot_controller** запускался **turtlesim.**

Ссылки на ROS Wiki:

http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv