Exercise 2: Introduction to ROS

Development of Intelligent Systems

2019

In this exercise you will get further familiarized with ROS. We will explore services, writing nodes in Python, the usage of `roslaunch` and `rosbag` commands as well as some additional ROS commands. Download the code for Exercise 2 and extract the files in the `src` of your workspace. Build the package using the `catkin_make` command.

1 Services

Download (if you have not already) the materials for Exercise1. In the exercise you have examples of creating a ROS service and a ROS client as well as defining a custom service format. We define a custom service by specifying the structure of the request that the service will accept and the response that it will return. To see an example see the `exercise1/srv/Reverse.srv` file. When we define a custom service we need to instruct `catkin` to build the necessary header files. This is done inside `CMakeLists.txt` similarly to how we build message files. For actual examples of a service and a client node explore the files `exercise1/src/service_node.cpp` and `exercise1/src/client_node.cpp`. The following links are useful for a quick start with services:

- Simple service and client in C++
- Simple service and client in Python

2 Starting multiple nodes

In ROS we usually have multiple nodes running at the same time and it quickly becomes impractical to use the `roscore` and `rosrun` commands for starting each node separately. For these purposes we use the `roslaunch` tool. This tool starts multiple nodes, as they are configured in a `.launch` file. Additionally, it starts the ROS Master if it is not running, we can use it to set parameters on the Parameter Server and more. Run the following command in the terminal:

```
roslaunch exercise1 greeting.launch
```

Which nodes were just started? Now explore the `exerise1/launch/greeting.launch` file to see an example of how to write a a `launch` file and familiarize yourself with some additional functionalities like setting values of the parameter server and remapping topic names. You can find more detailed information on the usage of `roslaunch` on the following links:

- Command-line usage
- Launch files structure

3 Parameter Server

Sometimes we want to be able to store the values of certain parameters so that they are available to every node in ROS. These values can be one of very different things: calibration parameters, which camera is currently in use, some variables to describe the current state of the system and so on. ROS makes this functionality availavle though the Parameter Server. Inside the `exerise2` package, in the `pubvel` node you have an example of how to use the parameter server.

- Usage of the parameter server in C++
- Usage of the paramete server in Python

4 Python nodes

Navigate to the `exercise2` package you installed at the beginning of this tutorial. Inside this package you have an example of a Python node. Explore this node to see the usage of the `rospy` module. Python nodes (as all Python programs) do not need to be built. So you could just add a new program to your package without actually calling `catkin_make`! Of course, if you are using custom messages or services you will have to first build the package. See the following links for a quick introduction:

- Simple publisher and subscriber in Python
- A simple service and a client in Python

The given example only demonstrates the construction of simple executable Python node. If we are developing a more complex Python package then it is necessary to do better organization of our code. On the following link you can find how to organize your code so that your module is actually installed system wide using `catkin`:

- Installing Python and Catkin

5 Recording and replaying messages

ROS contains the `rosbag` package which enables the recording and playback of messages posted to certain topics. This can be extremely useful for debugging purposes and sometimes enables us to develop programs without having access to the real robot. There are numerous online resources on how to use this package:

- Command-line usage
- Code API
- Cookbook examples

Let us try to record and playback massages using these package. In one terminal start the `roscore`. In a second terminal start the `turtlesim_node` from the `turtlesim` package. In a third terminal start the `randomwalk.launch` file which starts the `pubvel` node by running:

```
roslaunch exercise2 randomwalk.launch
```

Let us now record the movement messages that are being published. Run the command:

```
rosbag record --duration=2m --output-name=randomwalk.bag /turtle1/cmd_vel /turtle1/pose
```

As you might have guessed it we will record the messages posted to the `/turtle1/cmd_vel` and the `/turtle1/pose` topics in the next 2 minutes and save them in a file called `randomwalk.bag`. There are a lot of options available for the recording and playback of `.bag` files so make sure you read the Command-line usage tutorial before you use it. Let us now explore the recorded file. Navigate to its location and run the following command:

```
rosbag info randomwalk.bag
```

You can see a summary of its contents with information like which topics are contained inside it, how many messages, what are their types and so on. Now let us play back the recorded messages. First kill the node that is currently publishing movement commands, navigate to the terminal that is running the `pubvel` node and press Ctrl+C. Next, navigate the the location of the recorded `.bag` file and run the command:

```
rosbag play randomwalk.bag
```

Now the turtle should start moving and the examining the topics we can see that the recorded messages are being published. There are many options for the playback of the `.bag` files, one of which is pausing the playback by pressing the space key on the keyboard.

6 Some additional ROS commands

- In a terminal run:

  ```
  rqt
  ```

  This starts the ROS inspection GUI. In the plugins menu you can find many tools for visualizing topics and nodes, publishing messages, sending service requests, working with bag files and many others.

- Use the `rosservice` commands to find out what are the available services from the `turtlesim_node` and use it to send a request to a certain function.

- Use `rosparam` to read the parameter that the same node sets. Change some values.

7 Homework

Your task is to create a service node which moves the simulated turtle from the `turtlesim` package. The request should contain a string and an integer field. The string should be one of "circle", "rectangle", "triangle" or "random" and the integer field is a duration. The node should then move the turtle in a circular trajectory, rectangular, triangular or random for the given duration in the integer field. After the given duration the turtle should stop moving. The response to the client should contain a string field with the last issued movement type.