Worksheet 3 – ROS Bags

The focus of this work is to show how to record exchanged data on the ROS network, so that it can be played back in order to recreate system's behaviours. This is certainly a crucial functionality for debugging and identifying errors and faults on the system in a controlled and selective way by allowing the user to replicate the systems interactions and responses. This tool can also assist the programmer on the development of their systems as it can solve some logistic problems by minimizing the need to collect experimental data every time the user needs to test the created program, and run the recorded data instead and analysing the response of the created program.

Once upon a time…

In order the properly understand the value of this mechanism available in the ROS framework, let's imagine the following scenario:

To test a newly implemented autonomous navigation system, which involved so many hours of your time, you take your mobile robot for a field test. As a precaution you start the recording process of the exchanged data during the experiment. During the test a fault occurs with an unknown source, causing the robot to misplan a trajectory which was then executed. Let's assume that there was no security subsystems running (there should be always a security system running) to halt the robot when in danger of collision, or let's assume that this subsystem has also suffered from an error. As a result, since you weren't expecting this behaviour and were not able to reach the emergency button in time in your control centre, the robot has crashed into a tree. The only thing left to do now is to declare the experiment as finished and return to the workstation with a damaged/malfunctioning robot. Back in the workstation you collect the recorded data from the robot's hard-drive to be played in the system's backup, which was carefully stored in your own computer. While the robot is being repaired, you play the recorded data and carefully analyse the systems responses and behaviours. You later find what caused the error and were even able to find other potential errors on your system that were hidden and would need a field test to be discovered, being that test replaced by playing the recorded data. At the end, you will most likely be thanking the existence of this tool on the ROS framework.

How it works

The recorded data is stored in its own structure, denominated ROS Bags, which allow a selective reproduction of any type of message exchanged by the system with the same rate as they were first published. This way the time intervals between each message are kept as they were originally created and in particular, even if there was no data to be recorded for a while (for example, when you start the recording process) that time interval will be stored as well in the bag.

Creation of a ROS Bag

Let's start by learning how to record the exchanged data by creating a ROS Bag. To perform this task lets start by executing two processes from the "turtlesim" package, already installed on your ROS framework: the "turtle_teleop_key" node, and the "turtlesim_node" node. The last one simulates a turtle that will move accordingly to the messages received on the topic **/turtle1/cmd_vel**. This messages are of **geometry_msgs/Twist** type and are generated by the first node, "turtle_teleop_key", which listens for any keyboard events. Our objective is to record the messages published and play the generated bag. You'll see the turtle moving as if you were still pressing the keys on the keyboard in the same sequence as you did before.

Open three terminal windows and execute both nodes (run each command on a separate window):

```
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_teleop_key
```

After starting these three processes, a window with the simulator TurtleSim will open (Figure 1) and in the terminal that is running the "turtle_teleop_key" node you'll see the following text:

```
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
```

With that terminal selected, by pressing the arrow keys the turtle will move. Check what available topics are on the ROS network with the use of the **rostopic list** command. These are the topics that will be recorded on the bag we will create. You can do the same with to the services by using **rosservice list** to also know what services will be recorded. To start recording messages and services published in through the ROS framework you need to run the command **rosbag record -a**. The **-a** parameter stands for **all** and will indicate the intention to record every available topics in the network. You can also record only a sub-group of messages and services.

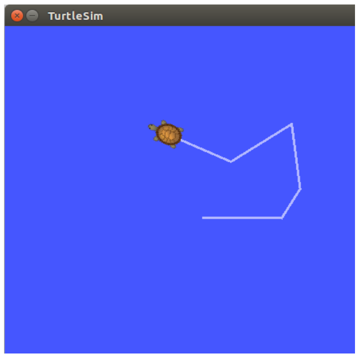

Figure 1 - TurtleSim Simulator

Restart the process “turtlesim_node” to get a clean start and open a new terminal window which will be used to record and play the bag. In the new window run the following commands:

```
$ roscd
$ cd ..
$ mkdir bags
$ cd /bags
$ rosbag record -a
```

Select the terminal where the node “turtle_teleop_key” is running and press the arrow keys on your keyboard. When you are done recording terminate the rosbag record process by selecting it’s respective terminal and pressing CTRL+C.

After recording a bag it is possible to check it’s contents and play them using the commands rosbag info <bag_name> and rosbag play <bag_name>, respectively. Keep in mind that <bag_name> should be the relative path of the bag you want to play or check, i.e. the full path from the actual folder where the terminal is executing the commands. Check the bag content using the respective command. You should see something similar to the following:

```
bag: <bag_name: year-month-day-hour-minutes-seconds>.bag
version: 1.2
start_time: 1259967777871383000
end_time: 1259967797238692999
length: 19367309999
topics:
  - name: /rosout
    count: 2
    datatype: roslib/Log
    md5sum: acffd30cd6b6de30f120938c17c593fb
  - name: /turtle1/color_sensor
    count: 1122
    datatype: turtlesim/Color
    md5sum: 353891e354491c51aabe32df673fb446
  - name: /turtle1/command_velocity
    count: 23
    datatype: turtlesim/Velocity
    md5sum: 9d5c2dcd348ac8f76ce2a4307bd63a13
  - name: /turtle1/pose
    count: 1121
    datatype: turtlesim/Pose
    md5sum: 863b248d5016ca62ea2e895ae5265cf9
```

We can see that beside the data of the messages themselves, the bag includes meta-data composed by the bag name, the instant when the bag started and stopped recording and the indication of how many messages were recorded.

Now let’s play the bag where we recorded the messages published by the “turtle_teleop_key”. Start by closing the terminal where this node was running and restart the “turtlesim_node” process to it’s initial state. Once again select the terminal where you executed the “rosbag record” and run:

```
$ rosbag play <bag_name>
```

As you can see the turtle is moving in the same path as it did before since the “turtle_teleop_key” is publishing the same information as if it was receiving the keystrokes it received before.

You can repeat this process even without running the “turtlesim_node” and recording a bag with only the “turtle_teleop_key” and when you play it with the simulation running, the turtle will move accordingly. As mentioned before, this is useful when programing your own system, as you can debug it more thoroughly after a field trial. You can even add new functionalities and test them on your system, as long as the required sensorial data were recorded.

Final Exercise

The resulting ROS bag from this worksheet must be zipped into a file named as “T3_<NUMBER1>_<NUMBER2>_<NUMBER3>.zip”, where NUMBER1 is your student’s number (i.e., T3_44500_46320_45321.zip). This file must be submitted at the Moodle platform until October 6th, 23:55.