ROS lab

Houssam Abbas
`houssam.abbas@oregonstate.edu`

March 28, 2020

1 Team logistics

Decide on a weekly meeting time for your team. You can meet via Skype, or Slack, or Google hangouts, or Whereby, etc. Post your weekly meeting time to Canvas and report it in your submission for this assignment.

2 ROS Tutorials

Install ROS on both the Jetson and your laptop if they're not already installed (follow the instructions in the Reference Manual).
Every student should do the following: Follow *all* 20 Beginner ROS tutorials at `http://wiki.ros.org/ROS/Tutorials`. Everything that follows in this class depends fundamentally on your ability to work in ROS comfortably and know where to find help and resources online.
Use C++ throughout the tutorials.
Evaluation:
From this, every student will commit their entire catkin workspace studentLastName_ws and the smaller of the ROS bags they created. Students in one team can commit these as different directories within the same repo. We will download your workspace, start up roscore, then

- run the listener and talker nodes, implemented in C++, and verify that indeed the communication is happening.
- examine the new message struct you created in Tutorial 10.
- play the smaller ROS bag (created in "Recording a subset of the data")

3 Extracting data from a ROS bag

Every student should do the following: Take the smaller of the ROS bags that you created (and which you turned in from Part 1). Create a second bag from it which contains only topic /turtle1/cmd_vel (you will need to figure out how to do this.) Call it studentLastName_velonly.bag and commit it in your workspace.

4 Create an averaging node for turtlesim

Every student should do the following: Using the talker / listener nodes from the tutorials as an example, create a *averaging* node that listens to the x-speed (speed along the x-axis) published by the turtlesim_node, and computes a sliding window average speed.
Note that unlike future labs, this lab does not provide skeleton code. Your will create the entire package and its nodes from scratch, following the examples in the tutorials.

- Your node will be implemented in C++.
- Node will be called studentLastName_average (studentLastName = your last name)
- The node is part of a new package called `studentLastName_runtime_monitoring`.
- The window size for average computation is 10 samples.
- It will subscribe to the messages published on topic /turtle1/cmd_vel
- Choose an appropriate data type for the messages that this node publishes. Each message is simply the computed average speed. These messages will be published to a new topic called 'average_velocity'.
- It will publish at a rate of 5 Hz.
- Have the callback in the node print some informative message to screen with every message it processes, e.g. using ROS_INFO.

Hints:

- Note that you can't pass extra arguments to the subscriber's callback: by design, the callback is always called with just the latest message. So how will you keep track of the last 10 messages?
- Remember to #include the header files for message types that you are using
- Make sure that your callback's signature specifies the right message type.

Evaluation:
We will add your package `studentLastName_runtime_monitoring` to our `catkin_ws` as part of the `beginner_tutorialspackage`, then run

```
$ catkin_make
$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun turtlesim turtle_telekey_op
$ rosrun studentLastName_runtime_monitoring studentLastName_average
$ rosbag record /turtle1/cmd_vel /average_velocity
```

We then drive the turtle around, close everything, and examine the messages in the bag. We expect to see the messages published from your node, and we will also do some code inspection.

5 Create an averaging node for ESC messages

This is a team exercise, so one submission per team.
You will now re-use your averaging node, but this time you want it to listen to a topic published by the ESC on-board the car.
At this point, you might not have learned anything about this ESC yet, or even run the car! Even if you ran it in manual mode, we haven't yet examined the various ROS nodes running on-board and how they relate to each other.
On the other hand, someone else has done that work. They ran the car, and recorded a bag of data from the messages published by the ESC. In this exercise, we demonstrate how replaying bag data is a very useful way of providing data without actually running the system, thus allowing you to test your code on pre-recorded data before deploying on the simulator or the car. Get the bag file, `motor_servo_bag_2018-06-20-13-40-13.bag`, which contains VESC messages, from the shared google drive `bags-for-students`.
The package for VESC can be found here: `https://github.com/mit-racecar/vesc`. It has dependencies on ackermann_msgs and serial. You can get the latter online and add it to your workspace. You can sudo apt−get the latter.
You will need to answer the following to develop your node (don't submit separate answers to these):
1. The authors of the VESC node have decided to define messages in their own package (unlike in the tutorials where the new message was created as part of package `beginner_tutorials`.) By inspecting the contents of this repo, can you tell what is the name of the package for the messages?
2. There are two types of messages in this package. What are they? Is there a relation between them?
3. By examining the code in `vesc_driver/src/vesc_driver.cpp`, can you tell what is the topic to which the VESC messages, of type VescStateStamped, are published (i.e., advertized)?
4. Create a new monitoring node and add it to package teamname_runtime_monitoring you created in the previous question.

- Your node will be implemented in C++.

- Node will be called teamname_average_vesc (teamname = your team's name). You can use one of the averaging nodes you created above as a starting point.
- The window size for average computation is 10 samples.
- It will subscribe to the VESC messages on their topic.
- It will publish messages of type float64, where each message is simply the computed average speed. These messages will be published to topic average_velocity.
- It will publish at a rate of 5 Hz.
- Have the callback in the node print some informative message to screen with every message it processes, e.g. using ROS_INFO.

To test your code, run (in separate terminals, as explained in the tutorials)

```
$ roscore
$ rosrun teamname_runtime_monitoring teamname_average_vesc
$ rosbag record -O teamname_vesc.bag /average_velocity
$ rosbag play motor_servo_bag_2018-06-20-13-40-13.bag
```

You will submit this new node as part of package teamname_runtime_monitoring which you created earlier.

Evaluation:

We will evaluate your code by executing the above sequence of commands, and by inspecting the code.