CMPE 189-01: Mobile Robots **Due: 09/15/19**

Assignment #1: ROS (100 points)

Instructors: Prof. Wencen Wu *Semester:* Fall 2019

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- Submit all deliverables listed in boxes throughout this document in a single PDF file on Canvas by the due date.
- Programming deliverables must be submitted as zip files with your name and name of the deliverable in the name of the file.
- This assignment should be completed individually. Do not share code with others.
- All sources of material must be cited. The University Academic Code of Conduct will be strictly enforced.

1 Resources

1. ROS Wiki: `http://wiki.ros.org/`
2. ROS Robot Programming (Free e-book):
   `http://wiki.ros.org/Books/ROS_Robot_Programming_English`
3. Programming Robots with ROS (Book):
   `http://shop.oreilly.com/product/0636920024736.do`

2 Overview

This assignment is designed to familiarize students with Robot Operating System (ROS) framework and some of the related tools. All instructions are recommended to be performed on Ubuntu 16.04/18.04 LTS operating systems. We recommend you install a version of Ubuntu on your laptops besides your main OS. Instead you may install Ubuntu on a VM using VMWare or virtualbox. There are plenty of instructions for installing Ubuntu online.

Note: The instructions in this assignment are not meant to be comprehensive. You are expected to run into issues that will require debugging. You will need to search online and look through other resources to figure out these issues.

3 Installation

We will be using ROS Kinetic Kame version for the purpose of this assignment. Please note that ROS Kinetic only supports Ubuntu 16.04; if you have Ubuntu 18.04 installed, please follow instructions for downloading ROS Melodic Morenia instead.

- ROS Kinetic Kame: `http://wiki.ros.org/kinetic/Installation/Ubuntu`
- ROS Medodic Morenia: `http://wiki.ros.org/melodic/Installation/Ubuntu`

ROS Wiki lists many tutorials to get started with ROS here: `http://wiki.ros.org/ROS/Tutorials`. Go through the following tutorials to get accustommed to common ROS systems. You will need to complete these tutorials to complete deliverables for this section.

- Configuring your ROS environment:
  `http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment`
- Navigating ROS Filesystem: `http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem`

- Read Chapter 2: Preliminaries (pages 9-29) of "Programming Robots with ROS (Book)" to understand the basic terminology of ROS.

Now that you have installed ROS and understand some of the basic terminologies, complete the following deliverables.

Deliverable 1: Run `roscore` in a terminal. This should start the ROS master node. Submit a screenshot of the terminal.

(5 points)

Deliverable 2: In a couple of sentences each, answer the following questions and provide corresponding screenshots:

1. **What is Catkin Workspace? Where is it located on your system? Run `catkin_make` command in your workspace and submit the screenshot of the output.**
2. **Provide screenshot of `ROS_PACKAGE_PATH` environment variable.**
3. **Using `rospack` command, find `rospy` package. Where is it located? Show the screenshot.**

(15 points)

Please note that you will need to have `roscore` running for completing majority of questions in this assignment. If you see an error saying "Failed to contact master at ...", it is most likely because you closed the terminal running `roscore`.

4 ROS Packages

ROS uses packages to organize its programs. You can think of a package as all the files that a specific ROS program contains; all its C++ files, python files, configuration files, compilation files, launch files, and parameters files. In this section, you will create your first ROS package.

4.1 Structure of ROS packages

- `launch` folder: Contains launch files for launching one or many ROS nodes inside the package.
- `src` folder: Contains the source files (.cpp or .py files).
- `CMakeLists.txt`: List of CMake rules for compilation.
- `package.xml`: Package information and dependencies.

4.2 Creating a package

To create ROS packages, you need to work inside your Catkin workspace.

- Use the `roscd` command to go to your workspace as follows:

```
roscd
cd ..
pwd
```

  If your workspace was named `catkin_ws`, the output would be similar to

```
/home/user/catkin_ws
```

- Now, go to the `src` folder. You will be placing your package here.

```
cd src
```

- We will now use the `catkin_create_pkg` command to create our package. This command is used as follows:

```
catkin_create_pkg <package name> <dependencies>
```

Use this command to create a package with your name with the format of "lastname_firstname_pkg" and add `rospy` as a dependency. For example, if your name were "John Smith", your package name would be `smith_john_pkg`.

Deliverable 3: Use `rospack list | grep lastname_firstname_pkg` to make sure your package is correctly created andrecognized by ROS. Show screenshot of the terminal.

(5 points)

- Go to the `src` directory of your newly created package by using `roscd lastname_firstname_pkg/src` command.
- Create a python file named `first_node.py` and insert the following contents into this file.

```
#! /usr/bin/env python
import rospy
rospy.init_node('mobile_robots')
rate = rospy.Rate(5)
count = 0
while not rospy.is_shutdown():
    if count >= 10:
        break
    print "mobile robots are awesome!"
    rate.sleep()
    count += 1
```

This file imports the `rospy` package and creates a ROS node named "mobile_robots". It then prints the following statement: "Mobile robots are awesome!" 10 times at the rate of 5 Hz. For more details on `rospy`, visit `http://docs.ros.org/api/rospy/html/`.

- Now, go back up to your package directory and create a `launch` folder. Inside this folder, create a launch file named `first_launch.launch`. Copy the following contents into this file (replace package name with your package name.)

```
<launch>
  <node name="mobile_robots"
        pkg="lastname_firstname_pkg"
        type="first_node.py"
        output="screen">
  </node>
</launch>
```

This file creates a `node` tag, which specifies the details of the node created by `first_node.py` file. Launch files can launch multiple nodes from the same package, as well as call launch files from other packages.

- Now, you're ready to launch your first ROS node from your first ROS package. First, open a new terminal and run `roscore`, this initiates the ROS master node. Then, in a different terminal run the following command.

```
roslaunch lastname_firstname_pkg first_launch.launch
```

Did it work as expected? If not, you may need to rebuild your catkin workspace. To do this, go back to your workspace using `roscd; cd ..` and then rebuild using `catkin_make`. Then try the previous command again.

If you get the error message "can't locate ... node in package ...", thats because of the lack of permission for this file. We can give the file permission by typing:

```
chmod +x first_node.py
```

Deliverable 4: Run the previous command and submit the screenshot showing the output of launching the ROS node you created.

(5 points)

Congratulations! You've successfully built your first ROS package.
You can learn more about ROS packages here: `http://wiki.ros.org/Packages`.

4.3 ROS Nodes

In the previous section, you built your first ROS node. ROS provides a helpful command: `rosnode` that allows you to gain information about **running** nodes. Explore the following commands:

- `rosnode list`
  What does this command do? Can you see the `mobile_robots` node created in the previous section? You cannot! This is because the node is killed when the python program ends.

Deliverable 5: Modify the python code in the previous section so that the `mobile_robots` node runs in a loop forever. Then, run `rosnode list` in a different terminal and locate the node. Submit screenshot of the terminal with this node located.

(5 points)

- `rosnode info /autonomous_driving`
  This command provides an easy way to see a summary what this node does.

You can learn more about ROS nodes here: `http://wiki.ros.org/Nodes`.

5 ROS Environment Variables

ROS uses various Linux environment variables to locate its components and work correctly. You can see these variables by running the following command:

```
export | grep ROS
```

The most important variables here are `ROS_PACKAGE_PATH` and `ROS_MASTER_URI`.

Deliverable 6: Submit a screenshot of the output of running the command above. Search online and answer the following questions:

1. **What is the `ROS_MASTER_URI`? Where is it pointing to on your system?**
2. **What are your `ROS_DISTRO` and `ROS_VERSION` variables?**

(5 points)

6 Topics and Messages

One of the ways ROS nodes communicate with each other is by using topics and messages. A topic is like a pipe that can carry a specific type of message through it. ROS nodes can either publish or subscribe to these topics to deliver or receive information from other nodes in the system. In this section, you will create two more nodes in your ROS package that will use topics and messages.

6.1 Publisher

- Navigate back to your package using the `roscd <your package name>` command and create a new file named `my_name_publisher.py` in the `src` directory.
- Copy the following contents into the file.

```
#! /usr/bin/env python

import rospy
from std_msgs.msg import String

rospy.init_node('my_name_node')
pub = rospy.Publisher('/my_name_topic', String, queue_size=1)
name = String()
rate = rospy.Rate(1)
count = 1

while not rospy.is_shutdown():
  name.data = "John Smith-" + str(count)
  pub.publish(name)
  count += 1
  rate.sleep()
```

  This code initializes a node named `my_name_node` and creates a `rospy.Publisher` object that publishes on topic `/my_name_topic` messages of type `String` from `std_msgs` package. **Don't forget to change "John Smith" to your name!** Can you tell what the published data would look like?

- Now create a launch file for launching `my_name_node`. You can use the launch file we created in 4 as reference. This launch file will be very similar. Call this launch file `second_launch.launch` and place it in the `launch` directory of your package.
- You can now rebuild your package using `catkin_make`. However, this time, we will use a new argument that will **only** build the package you specify. Remember to go back to your catkin workspace to run this command.

```
catkin_make --only-pkg-with-deps <package_name>
```

- Now launch your `second_launch.launch` file using `roslaunch`. Do you remember how to use this command?
- Once your `my_name_node` is running, it will begin publishing on the specified topic. How can we see the published data though? There is a handy command called `rostopic` that lets us print the published data in the terminal. First, let us make sure that the node is working correctly. In a **new** terminal window, type `rostopic list | grep /my_name_topic`. You should see your topic there, if not, check and make sure there aren't any errors in launching the node.

  You can now see the published data by typing `rostopic echo /my_name_topic`. This command should print messages as they are published by the node.

Deliverable 7: Submit the screenshot of the terminal after running the `rostopic echo /my_name_topic` command.

(5 points)

6.2 Subscriber

In this section, we will create a node that subscribes to the `my_name_topic` created in the previous section.

- Create a new python file in the `src` directory of your package and call it `my_name_subscriber.py`. Place the following contents into it.

```
#! /usr/bin/env python

import rospy
from std_msgs.msg import String

def callback(msg):
  print "I received: " + msg.data

rospy.init_node('my_name_subscriber_node')
sub = rospy.Subscriber('/my_name_topic', String, callback)
rospy.spin()
```

  This code creates a new node `my_name_subscriber_node` which subscribes to `my_name_topic` and accepts `String` type messages. Once it receives a message, it calls the `callback` function that prints the contents of the message.

- Now create a launch file for launching this node and name it `third_launch.launch`. Place this launch file in the `launch` directory of your package.
- Rebuild your package and launch both - publisher and subscriber - in two separate terminals. Does your subscriber print what you expect the publisher to be publishing? Is it similar to the `rostopic echo /my_name_topic` command we used earlier?

Deliverable 8: Submit the screenshot of the terminal after launching the subscriber node. It should print messages being published by the publisher node.

(10 points)

6.3 RQT Graph

ROS provides a nice visual way of showing which nodes are running and how they are communicating with each other. This tool is called RQT Graph. With both the publisher and subscriber from the previous section running, open a new terminal and enter `rqt_graph`. A new window should open that will show your two nodes and the topic they are communicating on.

Deliverable 9: Submit screenshot of the RQT graph with both nodes running.

(5 points)

Now kill the subscriber node by pressing CTRL+C in the terminal. Refresh RQT graph by hitting the refresh button in the top left corner.

Deliverable 10: Submit screenshot of the RQT graph with only the publisher running.

(5 points)

6.4 Messages

So far you have used the `String` message provided in the `std_msgs` package. However, ROS provides many types of primitive and more complex messages natively. You can find messages defined in `std_msgs` package here: `http://wiki.ros.org/std_msgs`.

You can also create your own custom messages, however the process is a bit involved. To learn how to do that, follow the steps here `http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv#Creating_a_msg`. You

can ignore the `srv` sections. This link also shows how to use the `rosmsg show` command. Read the section and complete the following deliverable.

Deliverable 11: `Twist` is a message type that is used very commonly in defining a robot's velocity in free-space. Use `rosmsg show` command to find how it is defined. Submit a screenshot of the terminal showing the definition of `Twist` message type. It is defined in the `geometry_msgs` package.

(5 points)

7 ROSbags

ROSbags are a handy tool that allows you to record all data passed through topics and replay them later to recreate the same conditions through a single file. Let's record the data being published by the publisher on the `my_name_topic`. Launch both - publisher and subscriber - you created in 6 and also run RQT Graph.

To record ROSbags, type `rosbag record -O <bag name> <topic1> <topic2> ... <topicN>`. In this case, type:

```
rosbag record -O bag1.bag my_name_topic
```

Refresh RQT graph. What changes do you see on the RQT Graph?

Deliverable 12: Explain the changes hyou see on the RQT Graph. Submit a screenshot of the RQT Graph.

(5 points)

Record for at least 2 minutes. Then, stop recording, and also kill the publisher. **Do not kill the subscriber.** You will notice that the subscriber stops printing as nothing new is being published to the topic anymore. Now, replay the rosbag you recorded using the following command:

```
rosbag play bag1.bag
```

Does the subscriber print the recorded messages? Where are these messages coming from? Refresh the RQT Graph.

Deliverable 13: Explain the changes you see on the RQT Graph. Submit a screenshot of the RQT Graph.

(5 points)

8 Visualization

Visualizing recorded data is extremely important for debugging in the field of robotics and self-driving vehicles. In this section, we will visualize data recorded from the LiDAR sensor of an actual autonomous car by folks at Autoware.

First, download the ROSbag and extract it from the following link. Note that the download size is around 3.1 GB, and extracted size around 8.4 GB.

```
http://db3.ertl.jp/autoware/sample_data/sample_moriyama_150324.tar.gz
```

Deliverable 14: What are all the topics and messages contained in the provided ROSbag. You'll have to search online for the command that provides you with this information. Provide a screenshot of the terminal window listing all topics and messages. How long was this ROSbag collected for?

(10 points)

8.1 RViz

RViz is a visualization tool built into ROS which provides a way to visualize many different message types. For a quick overview of RViz, what this video by TheConstruct: `https://youtu.be/yLwr5Zhr_t8?t=72`.

To visualize the recorded LiDAR pointclouds in the provided ROSbag, follow these steps:

- Play ROSbag in a loop using `-l` argument after the command `rosbag play`.
- In a different terminal, run `rosrun rviz rviz`. This should open up a new window with RViz interface.
- In the RViz interfance, under "Global Options", change value of "Fixed Frame" to "velodyne".
- Click "Add" and select a "PointCloud2" message type. Under "PointCloud2", enter the correct topic that pointclouds are being published from. You should have seen this topic while completing the previous deliverable. Do you see the pointclouds contained in the ROSbag?

Deliverable 15: Submit a screenshot of the pointcloud visualization in RViz. You may need to change the Size parameter to make the points more visible. Play around with other settings in RViz and customize the view for yourself. Now that you have seen LiDAR pointclouds, what do you think they can be useful for? Answer in 50-100 words.

(10 points)