ESE-680 - Autonomous Racing **(Due: 09/04/19)**

ROS Lab

Instructor: Rahul Mangharam *Name:* Hongtao Zhang, *PennID:* hzha

Goals and Learning outcomes The goal of this lab assignment is to get you familiar with the various paradigms and uses of ROS and how it can be used to build robust robotic systems.
ROS is a meta-operating system which simplifies inter-process communication between elements of a robot's perception planning and control systems.
The following are the basic fundamentals of ROS that you should be done with by the end of this lab.

- Understanding the directory structure and framework of ROS
- Understanding how publishers and subscribers are implemented
- Implementing custom messages
- Understanding Cmake lists and package.XML files
- Understanding dependencies
- Working with launch files
- Working with Rviz
- Working with Bag files

We would highly recommend that you are able to understand this tutorial in both Python and C++. The Autonomous/Robotics industry is heavily C++ oriented, especially outside of machine learning and data science applications. This class will be a good opportunity to get hands-on with C++ implementation in robotic systems. In any case, the lab may be submitted in C++ or Python. At least one lab in the course will require the use of C++; in general both options will be available. All written questions must be answered irrespective of the language you choose for your implementation. The questions titled with **Python** are for python and **C++** is for C++. If the question titles **Python & C++** then the same question applies for both Python and C++

NOTE: There is a section of good programming practices while writing code at the very end of this lab. It would be good to go through that once before you start this lab and keep it mind while you implement the code of this lab and also in later labs.

Problem Lab Assignment (points)

Complete the programming exercises and answer the questions with every section.

1. **Setting up ROS Workspace and a new package. (10)**
   Use the following two tutorials to setup a workspace and a test package in your vehicle.
   Create a Workspace
   Create a Package

   The tutorials have both for python and C++. Pick one. Name your workspace as:

   ```
   <student_name_ws>
   ```

   And the Package as :

   ```
   <student_name_roslab>
   ```

   Answer the following questions:

   (a) (**C++ & Python**) What is a Cmakelist.txt ? Is it related to a make file used for compiling C++ objects? If yes then what is the difference between the two?

   The file CMakeLists.txt is the input to the CMake build system for building packages. Any CMake-compliant package contains one or more CmakeList.txt file describing how to build the code and where it should be installed to.
   A make file in C++ contains instructions for how to automatically build the program. The CMakeList.txt is the script by which the CMake instruction can generate make files. According to these make files, the make instruction can produce executable files. And CMake can be cross-platform.

   (b) (**Python & C++**) Are you using CMakelist.txt for python in ROS? Is there a executable object being created for python?

   Yes, there is a CMakelist.txt for Python in ROS. There is also an executable object being created for Python, it is necessary to use the command chmod +x your_node.py to get an executable that rosrun can use.

   (c) (**Python & C++**) Where would you run the catkin_make?In which directory?

   The catkin_make should be called in the top level of the work space in the terminal.

   (d) The following command was used in the tutorial

   ```
   $ source /opt/ros/kinetic(melodic)/setup.bash
   $ source devel/setup.bash
   $ echo $ROS_PACKAGE_PATH
   ```

   (**Python & C++**)What is the significance of sourcing the setup files?

   Sourcing the setup.bash file in the devel folder will make sure the local workspace is properly overlayed by the setup script, which is adding environment variables to local path to allow ROS to function or make sure ROS_PACKAGE_PATH environment variable includes the current directory.

2. **Implementing a publisher and subscriber (35)**
We will now implement a publisher and a subscriber. Use the following references if you are new to ROS:

Wrtiting publishers and subscribers in C++ ROS

Wrtiting publishers and subscribers in python ROS

2.1 Simple Lidar Processing Node: Subscribing to data (15)
We will subscribe to the data published by the lidar in the simulated vehicle in the Sim that has been introduced. Run the following commands in different terminals.

```
$ roslaunch racecar_simulator simulator.launch
```

In a different terminal

```
$ rostopic list
```

You will now see a complete list of topics being published by the Simulator, one of which will be /scan. Run the following commands

```
$ rostopic echo /scan
```

This command prints out the data which is being published over the /scan topic in the terminal. The scan topic contains the measurements made by the 2d lidar scanner around the vehicles. The data contains 1080 distance measurements at fixed angle increments.

Your task is to create a new node which subscribes to the /scan topic.

- You will have to take care of the data(message) type of the /scan topic. It should be included in your call back function. You can check this by the command.

```
$ rostopic info /scan
```

- The message type should be :

```
sensor_msg::LaserScan
```

  You can also check the information of the message by using the following commands:

```
$ rosmsg show sensor_msgs/LaserScan
```

  Go through the data type documentation and see how you can work acquire the data(hint: you should be using std::arrays and/or std::vectors here and be taking care of inf values using std::isinf and NaN values using std::isnan)

  Suggestion: If you are not familiar with using gdb (c++ debugger) or pdb(python debugger) you can print out messages using:

```
ROS_INFO_STREAM()
```

- Be sure to include the header file of the message file in your script. (more on what this header file is in a later section of this lab)

2.2 Simple Lidar Processing Node: Publishing to a new topic (15)

Now we will process the data we have received from the lidar and publish it over some topic.
Find out the maximum value in the lidar data (the farthest point which is the range in meters) and the minimum value(the closest point which is the range in meters). Publishin them over two separate topics,

```
\closest_point
\farthest_point
```

Keep the data type (message type) for both the topics as Float64.

(a) (**C++**)What is a nodehandle object? Can we have more than one nodehandle objects in a single node ?

A NodeHandle object is object representing the ROS node.
Yes. Usually there is one or two nodehandles in one node.

(b) (**Python**) Is there a nodehandle object in python? What is the significance of rospy.init_node()

Python does not have a nodehandle object as in C++. Issues related to resolving names about the node name in Python can use rospy.resolve_name(), but a string instead of an object will be returned. rospy.init_node() is executed in a rospy program to initialize the ROS node for the process. Only one node is premitted in a rospy progress, so the rospy.init_node() can only be called once.

(c) (**C++**)What is ros::spinOnce()?How is it different from ros::Spin()

When users are subscribing messages, services or actions, they must call spin to process event. While ros::spinOnce() handles the issues and returns immediately, ros::spin() blocks until ros invokes a shutdown. So ros::spin() provides the users with more control if needed.

(d) (**C++**)What is ros::rate() ?

ros::rate() is controlling the desired frequency. There is a parameter defining rate to run at in Hz.

(e) (**Python**) How do you control callbacks in python for the subscribers ? Do you need spin() or spinonce() in python?

In Python, rospy.rate() is utilized to control the frequency for calling the callback function. Spin() is keeping Python from existing till the node ends, and there is no spinonce() in python.

3. **Implementing Custom Messages (20)**
Now we will implement a Custom message in the package you have developed above. The following tutorial explains how to implement and use Custom messages. Creating custom ROS message files (**take care of the cmake list and the XML file. Also, take care of including the header file of the message file in your script**)
You need to implement a custom message which includes both maximum and minimum values of the scan topic and publishes them over a topic:

```
Msg File name: scan_range.msg
Topic name : /scan_range
```

Questions:

(a) (**C++**)Why did you include the header file of the message file instead of the message file itself?

The header file of the message file is paired with message file, with the header file providing forward declarations for the message file. And these forward declarations are necessary when dealing with the message elements.

(b) (**Python & C++**)In the documentation of the LaserScan message there was also a data type called Header header. What is that? Can you also include it in your message file? What information does it provide? Include Header in your message file too.

Header is a special ROS type which provides a general mechanism for setting frame IDs for libraries like tf.
Yes, the Header header is usually in the first line of a custom msg file, which will be resolved as std_msgs/Header.
The header contains a timestamp and coordinate frame information that are commonly used in ROS.

4. **Recording and publishing bag files(15)**
Here we will work with bagfiles. Follow this tutorial to record a a bag file using the given commands.
Robsag Tutorial

Questions:

(a) (**Python & C++**)Where does the bag file get saved? How can you change where it is saved?

The rosbag file is saved in the directory where the command "rosbag record -a" is run. If the command "rosbag record -a" is run in different directory, the bag file will be saved in that directory.

(b) (**Python & C++**)Where will the bag file be saved if you were launching the recording of bagfile record through a launch file. How can you change where it is saved?

When using .launch file to run rosbag, the bag file will be saved in /.ros. And by renaming the name of the bag file by -o(prefix)/-O(whole name), changing the path the bag file is saved can be achived.

5. **Using Launch files to launch multiple nodes (15)**
Implement a launch file which starts the node you have developed above along with Rviz. If you are not familiar with RViz or launch files, the Rviz tutorial of RosWiki will be helpful:
RViz Tutorial
Roslaunch Tutorial
Launch file name : student_name_roslab.launch

Set the parameters of Rviz to display your lidar scan instead of manually doing it through the Rviz GUI. Change rviz configuration file. You will have to first change the configurations in the Rviz GUI, save them and then launch them using the launch file.
Here are a couple of good answers on ROS wiki for saving and launching Rviz Configruation files:
Launching Rviz Config file
Saving Rviz config file

6. **Good Programming practices (5)**
This class is heavily implementation oriented and it is our hope that this class will help you reach a better level of programming robotic systems which are robust and safety critical.

(a) **Common**

- The skeletons will be in the format of class objects. Keep them as that. If you need to implement a new functionality which can be kept separate put it in a separate function definition inside the skeleton class or inside a different class whose object you can call in the skeleton class.
- Keep things private as much as you can. Global variables are strongly discouraged. Additionally, public class variables are also to be used only when absolutely necessary. Most of the labs you can keep everything private apart from initialization function.
- Use debuggers. Will take a day to set up but will help you in the entire class. The debuggers are mentioned in the sections below:

(b) **Python**

- ROS uses Python2 and not Python3. Make sure that your system-wide default python is python2 and not python3.
- Use PDB. Easy to use and amazing to work with. PDB Tutorial
- Use spaces instead of tab. Spaces are universally the same in all machines and text editors. If you are used to using Tabs, then take care that you are consistent in the entire script.
- Vectorize your code. Numpy is extremely helpful and easy to use for vectorizing loops. Nested for loops will slow down your code, try to avoid them.
- This is a good reference for Python-ROS coding style: Python-ROS style guide

(c) **C++**

- Use GDB and/or Valgrind. You will have to define the dependencies in your cmake lists and some flags. GDB is good for segmentation faults and Valgrind is good for Memory leaks. Debugging with ROS Tutorial
- C++ 11 has functionalities which are helpful in writing better code.You should be looking at things like uniform initialization, **auto** key word and iterating with **auto** in loops.
- Use maps and un-ordered maps whenever you need key value pair implementations. Use sets when you want to make sure that there are unique values in the series. Vectors are good too when you just want good old arrays. All the aforementioned containers are good for searching as they don't require going through the entire data to search. Linked lists will not be helpful too much in most cases. Exceptions maybe there.
- This is a good reference for C++ - ROS coding style: C++ -ROS style guide

Note: You are not supposed to use any separate packages or dependencies for this lab.

Deliverable:

1. Pdf with answers filled in. (the source LaTex files are provided)
2. A ROS Package by the name of : student_name_roslab
3. the ROS Package should have the following files
   (a) lidar_processing.cpp
   (b) (or) lidar_processing.py
   (c) scan_range.msg
   (d) student_name_roslab.launch
   (e) Any other helper function files that you use.
   (f) A README with any other dependencies your submission requires (you should not need any).