F1TENTH Autonomous Racing **(Due Date:)**

Lab 1: Introduction to ROS

Instructor: INSTRUCTOR *Name:* STUDENT NAME, *StudentID:* ID

This lab and all related course material on F1TENTH Autonomous Racing has been developed by the Safe Autonomous Systems Lab at the University of Pennsylvania (Dr. Rahul Mangharam). It is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. You may download, use, and modify the material, but must give attribution appropriately. Best practices can be found here.

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission. All sources of material must be cited. The University Academic Code of Conduct will be strictly enforced.

1 Learning outcomes

The following fundamentals should be understood by the students upon completion of this lab:

- Understanding the directory structure and framework of ROS
- Understanding how publishers and subscribers are implemented
- Understanding how custom messages are implemented
- Understanding CMakeLists.txt and package.xml files
- Understanding dependencies
- Working with launch files
- Working with RViz
- Working with Bag files

2 Overview

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 various sub-systems.

We highly recommend that you go through this tutorial in both Python and C++ to help yourself to learn all aspects of ROS. The written questions are titled with their corresponding languates. The questions titled with **Python** are for Python, **C++** are for C++, and **Python & C++** for both Python and C++. If you choose to finish this lab in one language, answer the questions for that language and the questions for both languages.

3 Workspaces and Packages

Use the following two tutorials to setup a workspace and a test package.
Create a Workspace
Create a Package
Name your workspace as:

```
<student_name_ws>
```

And the Package as:

```
<student_name_roslab>
```

3.1 Written Questions

1. (**Python & C++**) What is a CMakeList? Is it related to a make file used for compiling C++ objects? If yes then what is the difference between the two?
2. (**Python & C++**) Are you using CMakeList.txt for Python in ROS? Is there a executable object being created for Python?
3. (**Python & C++**) In which directory would you run `catkin_make`?
4. (**Python & C++**) The following commands were used in the tutorial

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

   Why do we need to source `setup.bash`? What does it do? Why do we have to different `setup.bash` files here and what is there difference?

4 Publishers and Subscribers

We will now implement a publisher and a subscriber. The corresponding tutorials are here:
Writing publishers and subscribers in C++ ROS
Writing publishers and subscribers in Python ROS

4.1 Subscribing to Data Using a Simple LIDAR Processing Node

We will subscribe to the data published by the synthetic LIDAR in the simulator. Run the following commands in different terminals. First launch the simulator:

```
$ roslaunch f110_simulator simulator.launch
```

In a different terminal, list all the topics active right now:

```
$ 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 to take a look at what's being published on this topic:

```
$ 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. The data contains distance measurements at fixed angle increments.

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

- In ROS, all data communicated on topics have to conform to a certain pre-defined message type. For the subscriber you're implementing that listens to the `/scan` topic, you'll need to designate the message type in your callback function. If you're not sure, you can check the message type of the messages being communicated on a topic (`/scan` in this case) by the following command:

  ```
  $ rostopic info /scan
  ```

- The message type of LIDAR scans is usually:

  `sensor_msgs::LaserScan` in C++ and `sensor_msgs.LaserScan` in Python.

  You can also check the detail information on the message type (e.g. what are the fields in the message structure and what are their primitive types) by using the following commands:

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

  Go through the message type documentation of LaserScan to figure out what fields you'll need and what primitive types they are so that you can extract the data you're looking for.

  You can go through ROS's documentation on what ROS primitive data types correspond to the languages' primitive and data type.

  Note: when working with LaserScan messages, **always** filter out NaNs and infs in the data. You can use `std::isinf` and `std::isnan` in C++ and `np.isinf` and `np.isnan` in Python(with NumPy).

  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)

4.2 Publishing to a New Topic Using a Simple Lidar Node

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). Publish them over two separate topics,

```
\closest_point
\farthest_point
```

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

4.3 Written Questions

1. (**C++**)What is a nodehandle object? Can we have more than one nodehandle objects in a single node?
2. (**Python**) Is there a nodehandle object in python? What is the significance of `rospy.init_node()`
3. (**C++**)What is `ros::spinOnce()`? How is it different from `ros::Spin()`?
4. (**C++**)What is `ros::rate()`?
5. (**Python**) How do you control callbacks in python for the subscribers? Do you need `spin()` or `spinonce()` in python?

5 Implementing Custom Messages

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 (**Pay attention to the cmake list and the XML file. Also, make sure to include the header file of hte 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
```

5.1 Written Questions

1. (**C++**)Why did you include the header file of the message file instead of the message file itself?
2. (**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.

6 Recording and Publishing Bag Files

Here we will work with bagfiles. Follow this tutorial to record a a bag file using the given commands.
Robsag Tutorial

6.1 Written Questions

1. (**Python & C++**)Where does the bag file get saved? How can you change where it is saved?
2. (**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?

7 Using Launch Files to Launch Multiple Nodes

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
Name your launchfile `student_name_lab1.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

8 Good Programming Practices

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.

1. **Python and C++**
   - 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. It will take a day to set up but will help you in the entire class. The debuggers are mentioned in the sections below.
2. **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
3. **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

9 Deliverables and Submission

Note: You are not supposed to use any separate packages or dependencies for this lab. Submit the following as `studentname_lab1.zip` (replace studentname with your name):

1. Pdf with written answers. Use the `lab1_solutions_template.tex` template in the latex folder.
2. A ROS Package by the name of `student_name_lab1`
3. the ROS Package should have the following files
   (a) `lidar_processing.cpp` OR `lidar_processing.py`
   (b) `scan_range.msg`
   (c) `student_name_roslab.launch`
   (d) Any other helper function files that you use.
   (e) A `README` with any other dependencies your submission requires (you should not need any).

10 Grading

10.1 Rubric

Topics	**Points**
Compilation	15
Written answers	15
Bag file Pprovided	5
Publisher subscriber implemented	20
custom messages implemented	20
Launch file and rviz configuration	20
Programming practices	5
Total	100