Assignment 0: Getting Started

The goal of this assignment is to help you become acquainted with the ins and outs of the Racecar platform. In doing so, you will need to employ basic concepts from Python, Numpy and ROS. Before starting this assignment, please review these three topics as necessary.
This assignment is to be completed individually.

Recall that the following command can be used to launch the Racecar and teleoperation nodes:
roslaunch racecar teleop.launch

1 Getting Started

Do the following in order to setup your ROS environment:

1. Put the following command in your ~/.bashrc if it is not already there:
   source /opt/ros/kinetic/setup.bash
   Make sure to source your ~/.bashrc after adding this command.
2. Create a ROS workspace on your lab/personal machine (i.e. not on the robot). If using a lab machine, keep your workspace in the `/home` directory.
3. Place the base drivers and skeleton package at an appropriate place in your workspace (hint: build a catkin workspace, and populate it according to here; if you discover missing packages when compiling, they can be installed according to ROS installation)
4. Compile the workspace
5. Source your workspace (or add appropriate command(s) to your ~/.bashrc and source it)
6. Set environment variables, such as ROS_MASTER_URI and ROS_IP (or edit and source ~/.bashrc).

2 Warm-up [8 pts]

Use ROS commands and tools to answer the following questions:

1. At what rate is IMU data published?
2. Which message type is used to publish laser scans?
3. Across all packages in your ROS distribution, which packages define a Pose message? Which fields are common across all Pose messages? (Hint: rosmsg)
4. In RVIZ, add the necessary topics to visualize the following:
   - Display the RobotModel with an alpha value of 0.5. Display the *base_link*, *laser*, *camera_rgb_frame*, and *base_imu_link* TFs (no other TFs should be displayed). Submit an image of the visualization.
   - Display RGB image, depth image, and laser data. Submit an approximately 30 second video of your screen as you teleoperate the car.
5. Create a diagram showing the interconnections between TF frames of the car (Hint: tf).
6. Where is the Hokuyo Laser located with respect to the *base_link* (according to their transform(s) specified in TF)?
7. Which topic does the *joy_teleop* node use to send controls to the motor controller (a.k.a VESC)?

3 Controlling the Racecar [10 pts]

In this section, you will use ROS utilities to record control inputs as you drive the robot around. You will then write a program to playback these inputs so that the robot can follow a pre-recorded path.

1. Use the `rosbag` command to record data published to the topic you specified in Problem 2.7 as you teleoperate the robot to follow a Figure-8 path. The output of `rosbag` can be specified with the `-o` argument; larger bag files should be saved locally, then copied to your `/home` for storage.
2. Implement a Python script in **BagFollower.py** that:
   - Extracts the data from the recorded bag file
   - Re-publishes the extracted data such that the robot follows a Figure-8 path without being tele-operated. Publish the data to the */vesc/high_level/ackermann_cmd_mux/input/nav_0* topic.
3. Implement a launch file in **BagFollower.launch** that:
   - Passes the bag file's path to your script
   - Launches your script
4. Test your script. Assuming that nothing is currently running on the robot, the following commands should cause your robot to follow a Figure-8 path:

   roslaunch racecar teleop.launch
   # Wait for robot to finish launching
   # Hold down the RIGHT bumper on the Logitech controller (it is labeled 'RB')
   roslaunch lab0 BagFollower.launch
5. Submit a video of your robot performing a Figure-8
6. Extra Credit: Add functionality to your script that allows the robot to follow the recorded path backwards (i.e. the robot drives in reverse). Add a parameter to your launch file that specifies whether the robot follows the path forwards or backwards. Submit a video.

4 Image Filtering [10 pts]

When working with image data in Computer Vision, Robotics, and/or Machine Learning systems, it is common to pre-process the images. Oftentimes, this pre-processing can involve the application of filters to the images. A filter is applied by sliding it along each direction of the image, where a sum of products is calculated at each position. For example, given the 5x5 image and 3x3 filter shown below:

$$\begin{bmatrix} 3 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 3 & 1 \\ 3 & 1 & 2 & 2 & 3 \\ 2 & 0 & 0 & 2 & 2 \\ 2 & 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

The filter is applied to the image by overlaying it onto the top left corner of the image, as indicated by the shading of the blue diagram below (note that the values of the filter have been placed in the lower right corner of each box). Corresponding elements of the filter and image are multiplied, and all of these products are summed. The result is placed into the first element of the output image, as illustrated by the shading in the green diagram below. The filter is then slid one position to the right, and the process repeats until the filter has visited all possible positions. Note that the output image has a smaller size than the input image. One can pad the border of the input image in order to have an output image with the same dimensions as the input image, but that is not required for this assignment. Complete the following to implement image filtering:

1. Implement a Python script in **ApplyFilter.py** that:
   - Uses Numpy to load a filter from a CSV file
   - Subscribes to a sensor_msgs/Image topic published by the camera

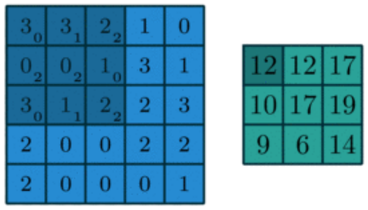

Figure 1: Left: The filter is first applied to the top left corner of the image. Right: The resulting output image. Figure is from Theano.

- Utilizes a callback function that applies the filter to the incoming image.Your callback should implement two different methods for filter application (but only one should used at a time, depending on a value specified in your launch file). First it should be able to apply it using nested for loops. Second, it should be able to apply it using Scipy's *convolve* function
- Publishes the output of the filter. Make sure your personal computer doesn't have a firewall that interferes with network publication.

2. Implement a launch file in **ApplyFilter.launch** that:
   - Specifies the path to the CSV filter file, the camera topic to be subscribed, and which of the two filter application methods should be used
   - Launches your Python script
3. Apply each of the following filters to RGB camera data:

$$\begin{bmatrix}-1 & 0 & 1\end{bmatrix}, \begin{bmatrix}\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9}\end{bmatrix}$$

   For each filter and each filter application method, at what rate is the filtered image published?
4. While using the faster filter application method, submit a video displaying both the original camera data and the filtered data as you drive the car around. You may use RVIZ for the visualizer.

5 Assignment Submission

Submit a single ZIP file containing a PDF or .txt file with answers to the questions, bag files, diagrams/images/videos, and code (2 python files, 2 launch files) to the Canvas dropbox.