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.

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 laptop/desktop machine (i.e. not on the robot).
3. If not already there, 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 the environment variables ROS_IP and ROS_MASTER_URI (or edit and source ~/.bashrc). ROS_IP should be set to the ip address of your desktop or laptop. ROS_MASTER_URI should be set to `http://<ip_address>:11311` where `<ip_address>` is the ip address of the ROS master. This will either be the ip of the robot if you are working with the robot, or the ip of your desktop/laptop if you are working in simulation.

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

2 Following the RACECAR [10 pts]

In this section, you will implement the teleoperated car (car A) being followed/lead by a clone (car B). The user will set an offset distance. If this distance is positive, your program should publish a pose for car B that is a fixed distance directly *ahead* of car A. If the distance is negative, the published pose should be a fixed distance directly *behind* car A.

2.1 In Simulation

The pose of car A changes as you teleoperate it. Your program should publish the pose of car B as specified above.

1. Implement a Python script in **CloneFollower.py** that:
   - Subscribes to the pose of car A.
   - Computes the pose of car B based on where car A is and the given offset distance.
   - Publishes the pose of car B.
2. Implement a launch file in **CloneFollower.launch** that:

- Passes an offset distance of 1.5 meters to your script
- Launches your script

3. Use your launch file to launch your node and verify that car B stays 1.5 meters directly ahead of car A.
4. Add bounds checking functionality to your script. Use the map to check if the pose computed for Car B is out of bounds. If it is, negate the offset distance (multiply it by negative one), recompute the pose of Car B, and publish the new pose. The result should be that if Car B goes out of bounds while it is *leading* Car A, Car B will start *following* Car A from a fixed distance. If Car B goes out of bounds while *following* Car A, it should start *leading* Car A from a fixed distance.
5. Add a parameter to your launch file that enables/disables bounds checking.
6. Test that your bounds checking functionality works

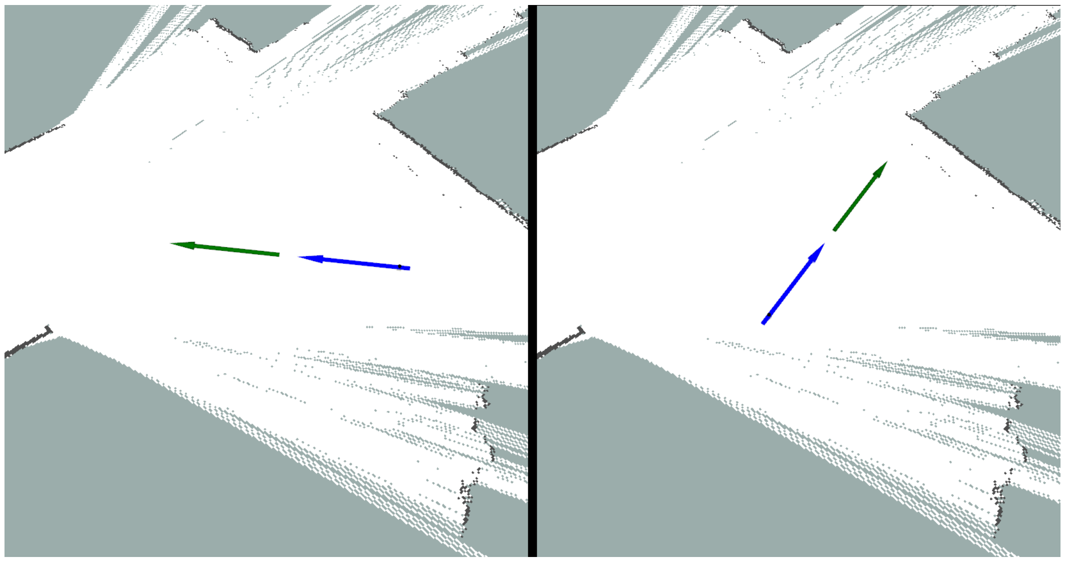
Figure 1: The pose of Car A in blue, the pose of Car B in green. As Car A is teleoperated, Car B stays 1.5 meters directly in front of Car A.

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.

3.1 In Simulation

1. Use the `rosbag` command to record data published to the */vesc/low_level/ackermann_cmd_mux/input/teleop* topic as you teleoperate the robot to follow a Figure-8 path. The output path of `rosbag` can be specified with the `-o` argument.
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 using the bag file. The robot should move in a Figure-8 pattern without being teleoperated.
5. 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. Test that the robot correctly does a Figure-8 backwards.

3.2 On Robot

1. Collect a bag of you teleoperating the real robot performing a Figure-8
2. Play the bag back so that the robot autonomously does a Figure-8. 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
```

3. Collect a bag of the robot travelling a path in the basement of the electrical engineering building, similar to the one shown below.
4. Again play this bag back so that the robot autonomously executes the commands recorded in the bag.

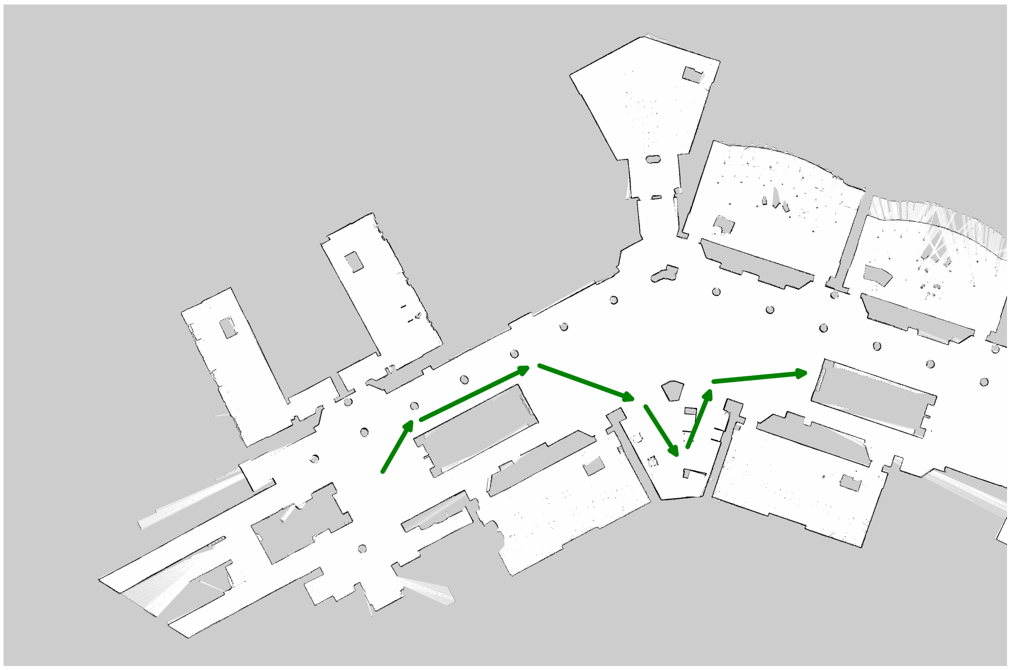

Figure 2: The path of the teleoperated robot is shown in green.

4 Assignment Submission

Submit the following:

1. Video demonstrating your CloneFollower following the teleoperated car and detecting when it goes out of bounds. (You can use Kazam to take a video of your screen.)
2. Video of your RACECAR performing a figure-8 in simulation
3. The two Python scripts and two launch files that you wrote.

In addition, if you have chosen to do the real robot track, submit the following:

1. Video of the real robot car performing a figure-8.
2. Video of the real robot at least attempting to follow the specified path in the EEB basement.
3. Answers to the following questions:
    - You should have collected two bags of the robot doing a figure-8, one in simulation, and one on the real robot. Play back both of these bags on the real robot. Explain any significant differences between the two performances.
    - Is the robot successfully able to navigate through Sieg Hall when playing back the corresponding bag? Note any deficiencies in performance. Explain why these deficiencies might occur.