ECE 4984 & 5984: (Advanced) Robot Motion Planning
Spring 2017
Homework 3
Due: Monday March 13th, 9PM

February 24, 2017

Instructions. This homework constitutes 5% of the course grade. You are allowed to work in teams of two (a team of one undergraduate and one graduate student is perfectly fine). **Both members of the team must submit a separate report written individually.** It is also okay to discuss the problems with other students in class. However, your submission must be your original work. This implies:

- If you discuss the problems with other students, you should write down their names in the pdf report.
- If you refer to any other source (textbook, websites, etc.), please cite them in the report at the relevant places.
- The answers in the pdf report must be written entirely by you from scratch. No verbatim copy-paste allowed without citations.
- Any software you submit must be written entirely by your team with no copy-pasting of significant portions of code from other sources.

Please follow the submission instructions posted on canvas exactly. You must submit your assignment online on canvas by the due date. Your submission must include one pdf report and one or more files containing your code. It is okay to scan your answers and create the pdf submission.

Finding a Partner

If you are already familiar with ROS, you are encouraged to partner with someone who is new to ROS. If you are looking for partners, please volunteer your contact information on this shared Google document: `https://docs.google.com/a/vt.edu/document/d/1rh9bSm3_r4btLNR60hGil3qgS6F27vTj3K9NdiPjLQA/edit?usp=sharing`

Problem Setup

In this assignment, you will write a simple controller for Pioneer 3DX, a differential drive robot[1], to follow a given path. You will implement two version of the controller: an open-loop one (relying on timing delays), and a closed-loop one (using feedback from the odometry).

The starter code is compatible with "Gazebo 2" and ROS Indigo. You are encouraged to use these versions to maintain compatibility throughout the course. If you prefer to use other versions of Gazebo and/or ROS, feel free to do so (but don't expect version related support from the instructors).

The starter code consists of configuration files to start the Gazebo simulator and source code for controlling the velocity of the simulated robot. We have provided the ROS source code in C++, Python, and

[1]See Section 13.1.2.2 from the "Planning Algorithms" textbook for a description of the Pioneer 3DX kinematic model.

MATLAB. You are free to use any one language you prefer. If you are using MATLAB, you would require the MATLAB Robotics Toolbox.[2]

Instructions to run the starter code.

1. Download the `HW3_Starter_Code` file and extract it to a convenient location inside the `src` directory within your catkin workspace.[3] We'll assume it is in the `src/hw3` directory from now on.
2. Install ROS controllers for the Pioneer robot by running the following commands in the terminal.
   ```
   sudo apt-get install ros-indigo-ros-control
   sudo apt-get install ros-indigo-ros-controllers
   ```
3. Run `catkin_make` inside your catkin workspace. The starter code should compile with no errors.
4. To run any ros node, you must run `roscore` first. Run this command in a separate terminal.
5. (C++, Python only) Open a new terminal. Source the setup file by running the following command
   ```
   source hw3/devel/setup.bash
   ```
6. Launch the gazebo world file
   `roslaunch p3dx_gazebo gazebo.launch`.
7. Now, execute the demo node: `rosrun ros_demo square` for C++ or `rosrun ros_demo square.py` for Python. If you are using MATLAB, then open the `square.m` file in MATLAB and execute the file. This node will make the pioneer move in a straight line continuously.
8. Observe the state of the robot by typing the following in separate terminals
   ```
   rostopic echo /odom
   rostopic echo /cmd_vel
   ```

The demo node in the starter code works by publishing velocity messages on the `cmd_vel` topic.

Problem 1: Open-Loop Square 40 points

Make the robot move in a 4×4 meters square. This will be with an open loop controller. You need to implement two subroutines: one to move in a straight line, and one to rotate in place. You need to publish the appropriate velocity values on the `cmd_vel` topic. This is a message of type `geometry_msgs/Twist`. You will need to publish on the linear component of `cmd_vel` and wait/sleep till the robot has moved four meters, stop the robot, and make the robot rotate 90 degrees by pushing on the angular component and repeat.

You should edit any of the three source files: `square.cpp`, `square.py`, or `square.m` to achieve the square motion.

Problem 2: Closed-Loop Control 60 points

You may notice that without feedback, the robot does not exactly move in a square. In the second version, you will improve the performance by implementing a closed-loop controller that reads in the current position of the robot. We can read the current position of the robot by subscribing to the `/odom` topic.

Implement a simple feedback controller to ensure the robot reaches the desired goal positions. A simple strategy is to use the two subroutines from Problem 1 (go-straight and rotate) to correct the robot's course if it steers away from the goal position.

The desired goal positions will be given in the form of a list provided in `input.txt`. This file contains a list of space separated x and y values. Your implementation must read in the file and get the robot to navigate to the goal positions one-by-one.

The implementation must be contained to any of the three files: `square.cpp`, `square.py`, or `square.m`. Note that you cannot use any existing navigation packages (e.g., move-base) for this problem.

[2] `https://www.mathworks.com/products/robotics.html`
[3] `http://wiki.ros.org/catkin/Tutorials/create_a_workspace`

Submission Instructions

Your submission must contain two items:

- A pdf report with a brief description of your implementation for both problems. Make sure you describe the controller. Specify which programming language you have used. Add one or more screenshots of Gazebo or rviz showing the robot's trajectory for both problems. You must also create a screencast video (e.g., using record-my-desktop) and upload this on Youtube. Add a link to the video in your report. Here's an example: `https://www.youtube.com/watch?v=i5GNDA_fkzE&feature=youtu.be`
- Your edited code as a zip file. This must include all the folders that are already present in the starter code plus any other files needed to compile your code.

The code you submit can be the same for both members of the team. However, you must each install and run the code on your own machine. Therefore, the screenshots and videos submitted in the report must be the ones you obtained on your individual machine.

Hints

- You may use a virtual machine which has ROS pre-installed for this assignment.
- Some relevant tutorials (see `http://www.ros.org/wiki/ROS/Tutorials`) are 5, 6, 11 and 12.
- The `geometry_msgs/Twist` type has six members: linear.x linear.y linear.z angular.x angular.y angular.z These are the requested velocities. linear.x sets the robot's translational speed, and angular.z sets the robots rotational speed.
- You may need a way to wait a specified time.
  For instance, See ros::Duration at `http://www.ros.org/wiki/roscpp/Overview/Time`
- A basic open-loop move can be executing by specifying a velocity and waiting enough time to travel the desired distance.

Helpful links

- Install ROS.
  `http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment`
- Install Gazebo.
  `http://gazebosim.org/tutorials?tut=install_ubuntu`
- Install Gazebo (Older versions).
  `http://gazebosim.org/tutorials?tut=install&ver=2.2`
- Gazebo-ROS version compatibility.
  `http://gazebosim.org/tutorials/?tut=ros_wrapper_versions`
- If you plan on using MATLAB, you would need to look into the MATLAB robotics toolbox.
  `https://www.mathworks.com/products/robotics.html`