Advanced Special Topics
CS 395
Fall 2017
Homework 4
Due in part by 11:59PM on Tuesday, Dec. 5, and in part by 10:30am on Wednesday, Dec. 6. (Please see the notes below!)

- The Programming Assignment should be submitted electronically to our course drop-box by 11:59pm on Tuesday, Dec. 5. Please use the submit395 command, as usual:

  \texttt{submit395 hw4 <your-hw4-dir>}

  If you have any questions about submit395, please ask me \textit{well before the deadline}!

- The required printouts should be turned in by 10:30am on Wednesday, Dec. 6. \textbf{Please turn in the required printouts to Jennie Colabella in the main CS office.}

  As discussed in class, time permitting, we will also schedule a meeting to demo your code.

- Please do this entire assignment inside the catkin workspace used for our HW3. When submitting your code, please submit the entire catkin workspace, and document (as noted in the exercises below) exactly what commands are needed to run your code for this assignment.

- Collaboration is restricted for this assignment. You should use only the ROS wiki pages (except for those that may tell you answers in ways that violate the course policy on Academic Integrity) and the texts \textit{A Gentle Introduction To ROS} (by O’Kane) and \textit{Programming Robots with ROS} (by Quigley, Gerkey, and Smart) as outside resources relating to ROS; you are not permitted to share code with classmates or look at the code of a classmate. (You may also use any Python resources for help with coding, but nothing else that is ROS-related.)

\section*{Programming Assignment: Finite State Machine Control!}

1. MoveAndAvoid in Gazebo! This exercise will ask you to explore a Gazebo simulated world with a simulated turtlebot. See the ROS wiki page on the turtlebot simulator (http://wiki.ros.org/turtlebot_simulator/Tutorials) for useful information.

   For this exercise, use your MoveAndAvoid code from HW3 so that it can work with a simulated turtlebot (not just a turtlesim!) in the Gazebo simulator. In particular, you should start up our default world with a turtlebot in it in Gazebo

   \texttt{roslaunch turtlebot_gazebo turtlebot_world.launch}

   and then remove the obstacles already there and insert a Starting Pen (along with other obstacles to create a good test environment for MoveAndAvoid, if you’d like). Then, try your MoveAndAvoid node in it. To visualize your simulated turtlebot, you can use rviz:
As you modify your MoveAndAvoid from HW3, please make sure it also still works to control a turtlesim! That is, it should be able to control the motion of both a turtlebot and a turtlesim (although the main point of this assignment is its function with a turtlebot).

Also, create a ROS bag file containing a demo of your MoveAndAvoid in Gazebo, containing only the messages needed for the turtlebot (not the turtlesim). (Also, time permitting, please be prepared to meet with me and give me a live demo!)

2. Finite State Machines (FSMs)! (With thanks to colleagues at Union College!) Our Lecture Notes page of November 17 contains a link to some python (pseudo-)code—a structure that can implement a finite state machine (FSM) framework to control a robot with multiple low-level behaviors.

(a) Please read through that code carefully. Note that the provided code includes a framework for a state called bumpreact, which is intended to work with the bump and cliff sensors on a turtlebot. You will not be using a bumpreact state—or bump and cliff sensors—for this exercise, but you will be building a framework to control a robot with two different behaviors!

In a finite state machine controller, each state (and there are only finitely many of them—thus, “finite state”) can be thought of as a robot behavior. The robot may switch from one state to another—i.e., from one behavior to another—based on events that occur in the robot’s environment. For example, in the provided FSMMaster.py code, when the robot bumps into something, there is a bump event, and the robot might make a state transition from the moveandavoid state (just navigating happily along) to the bumpreact state (responding to a bump). Similarly, when an allclear event occurs as defined for that controller, the robot can transition out of a bumpreact state and into a moveandavoid state. In particular, please see lines 16-26 of FSSMaster.py for more about the mechanism for FSM control in that code.

(b) Based on the provided code, create a new node named FSMMaster2. In it, break MoveAndAvoid into two separate states, Move and Avoid, and have them be separate states in the FSM. If there are no obstacles within some minimum distance, the Move state should steadily move the robot forward. If there are obstacles, the Avoid state should rotate the robot away from them. The events to transition between states can be named something like laserObstacle and laserClear. Use your LaserScanCallback to trigger these events. Modify the FSMMaster.py code to handle the events and state transitions properly.

Note that each state is only responsible for setting desired linear and angular velocities; the Steer() method should turn those values into a Twist topic sent to the robot.

(c) Create a ROS bag file containing a demo of your FSMMaster2 node in the Gazebo world (or one very similar to it) that you used for the ROS bag in exercise 1.