Due in part by 11:59PM on Tuesday, November 14, and in part at the beginning of class on Wednesday, November 15. (Please see the notes below!)

- The Programming Assignment should be submitted electronically by 11:59pm on Tuesday, November 14, and the required printouts should be turned in at the beginning of class on Wednesday, November 15.

- Please do this entire assignment inside a single catkin workspace. When submitting your code, please submit the entire catkin workspace you used for this assignment. You may have things other than this assignment in the workspace (e.g., you can reuse a workspace from a previous assignment), but you should submit the entire workspace, and you should document (see 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 and the texts *A Gentle Introduction To ROS* (by O’Kane) and *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.)

- **NOTE:** In exercises where you are asked to use tf2, please use the tf2 package with functionality from the tf package as needed (e.g., for conversions from quaternions to Euler angles), as demonstrated in the tf2 tutorials.

---

**Programming Assignment: Low-Level Behaviors!**

For the below, please recall that to run the turtlesim, the command is

```
rosrun turtlesim turtlesim_node
```

The purpose of this assignment is to give you more practice with ROS nodes and to have you program some low-level autonomous behaviors. We are still going to “fake” some aspects of how robots perceive their environments, but the code will work with real perception-oriented ROS topics to generate simple behaviors. The turtlesim will once again be the basis of much of the assignment.

1. (With thanks to colleagues at Union College!) As we discussed in class, you will create a SimulatedLaser Node that works with a turtlesim. The SimulatedLaser node should subscribe to the turtlesim’s pose topic, and it should publish LaserScan topics. Your simulated laser scanner should scan by ray casting a set of “lasers” in an arc in front of the turtle, and return the distances to any walls that those rays intersect; you can treat the location of the turtlesim (from its pose topic) as the source of the laser rays, and you can consider the boundaries of the turtle’s world to be the only walls. (You can experiment to find the coordinates of the turtlesim world; I believe it is 11m by

---

1
11m.) The size of the arc, the number of beams, and the max range of the rays should all be variables. Test your scanner by driving the turtle around with the teleop node.

Please see the Lecture Notes of November 8 for a link to some python code for Vector representation and line intersection. You may find it useful; feel free to modify it as you see fit.

(Hint: Start with 4-5 “beams” in your arc.)

2. (With thanks to colleagues at Union College!) Write a MoveAndAvoid node, which subscribes to the LaserScan topic, and publishes to a Twist topic. The node should constantly try to move forward, but also steer the robot away from nearby obstacles (here, the only obstacles are walls). Treat every data point from the laser scanner as a separate source of “repulsion”; when summed, they will produce the repulsion for the robot’s trajectory. Use a linear, exponential, or inverse-square ($\frac{1}{d^2}$) magnitude profile to have closer obstacles generate more “force” on the robot (e.g., turtle).

(See the course lecture notes for more information about potential fields and repulsion-based obstacle representations.)

Test your MoveAndAvoid on the turtlesim, using the SimulatedLaser node from the previous exercise. The goal is for MoveAndAvoid to work regardless of the arc or density of the Laser data.

3. Create a ROS bag file containing a demo of your MoveAndAvoid node. I should be able to replay your demo exactly on my turtlesim.

4. Finish up the Introduction to tf2 tutorial and the first six tf2 tutorials. See the Lecture Notes of November 8 for a link to the tf2 tutorials webpage.

5. Practice with tf2, pt. 1 Using tf2, make a turtlesim run from corner to corner—from coordinate (1, 1) to (1, 10) to (10, 10) to (10, 1) and around again—by introducing and changing a goal frame.

6. Practice with tf2, pt. 2 In this exercise, you’ll have two turtlesim turtles (I’ll call them turtle1 and turtle2) in the same turtlesim window. Also, for purposes of this exercise, I’ll use the directions up / down / left / right as referring to the directions on the computer screen, not the directions with respect to a turtle. (E.g., it doesn’t matter which way the turtle is facing, “left” refers to screen left, not the turtle’s left.)

Recall the turtle_tf2_demo from the Introduction to tf2 tutorial, where a turtle2 will follow a teleoperated turtle1 around—turtle2 goes to the position of turtle1.

For this exercise, you will also have two turtles in a window, where turtle1 will be teleoperated to move around the window, and turtle2 should have its pose controlled fully by tf / tf2. Here, however, instead of having turtle2 pursue the position of turtle1, turtle2 should follow a position 2 units left of turtle1—that is, it should line up 2 units left of turtle1. Moreover, turtle2 should face the same direction as turtle1 when it reaches that goal location, and if turtle1 turns but stays in the same location, turtle2 should also turn but stay in the same location—i.e., it should be 2 units left of turtle1 and facing the same direction as turtle1. So, turtle2 is in some sense following both the position and orientation of turtle1!
7. Finally, submit a printout containing some documentation accompanying your work for the programming exercises above, similar to the documentation for the previous HW assignment. The purpose of this documentation is to briefly explain important features of your work and how to run your code in context, as well as to include your code itself. Therefore, the printout should include, for each exercise:

- A brief explanation of the approach you took to solve the problems in the exercise.  
  *Note*: If you already provide a thorough explanation as part of comments in your code, you do not need to include a separate one for this documentation; including your code in the printout will suffice.

- Documentation of how to run and test your code. That is, submit a list of all the command-line ROS commands one would need to run in order to demonstrate how your code works.

- Your python code for that exercise. You need not print out any other elements of the ROS packages, just the python file you wrote for each exercise.