Ensure that the following packages are installed on your system:

ros-kinetic-stage

ros-kinetic-turtlebot-teleop

The attached zip file contains a package called world. Unzip this file into your catkin_ws/src directory:

Execute the following command:

roslaunch world world.launch

You will see a little robot and a red target square. A laser range finder is simulated, but we will not be using this sensor for this assignment (although you should feel free to experiment). To teleoperate the robot, open another terminal and execute the following:

roslaunch world teleop.launch

Note that will need to have the terminal selected for your keypresses to be able to control the robot. You can combine the launch of the world with teleoperation by launching "all.launch". Take a few minutes to drive around, check out the topics published by Stage, and mouse around with the Simulator. Once you are ready to work, create a package called world as follows:

catkin_create_pkg ros2 rospy

Note: In the tasks that follow we are assuming that the robot has perfect access to its own pose. This is an unrealistic assumption that we will tackle later in the course.

Create a scripts directory in ros2 and place go_to_goal.py in that directory. Attached is the go_to_goal.py file.

Implement the controllers that are outlined in the attached pdf.

Task 1: Smooth Controller 1

Implement "smooth controller 1". Your task is to create a class called SmoothController1 that fulfills the requirements in go_to_goal.py. This class should go in a file called smooth1.py. Note that the parameter settings for "smooth controller 1" are not critical (both can be set to 1 or you can experiment with different values). You can use the following template for smooth1.py (make the appropriate changes for tasks 2 and 3):

Test your code by launching the Stage simulation in one terminal (roslaunch world world.launch) and executing the following in another terminal:

rosrun ros2 go_to_goal.py

The controller should drive the robot to the red square (which the robot can pass through). Note that there is no goal angle specified with smooth controller 1. Important: you should not use the tf library here. The arguments passed into the constructor and the get_twist method provide everything that you need. The same goes for task 2.

Task 2: Smooth Controller 2

As above, only implement smooth controller 2. Your class should go in smooth2.py (which can start from the same template as smooth1.py). You will have to edit go_to_goal.py in order to use smooth2 instead of smooth1.

Note that this controller should guide the robot to the red square such that it ends up pointing down the negative x-axis. You should experiment with setting different goal angles to ensure that it works reliably. Note that smooth controller 2 will work well only with certain parameter settings. I have found good results with the following:

- K_RHO = 0.6
- K_ALPHA = 1.6
- K_THETA = 0.3

Task 3: Smooth Controller 1 using TF

Complete the first set of tutorials (http://wiki.ros.org/tf/Tutorials) on the tf library under “Learning tf”. You should follow the Python stream of tutorials. Re-implement smooth controller 1 to utilize tf. This should allow you to eliminate almost all of the math from the get_twist method (which now takes no arguments, since we are using tf internally).

The class should be named SmoothController1TF and should be placed in smooth1_tf.py. Once again, you will need to edit go_to_goal.py in order to activate smooth1_tf.

Note that all of the required functions from the tf library are demonstrated through the tf tutorials. For this part you should pay special attention to the “Adding a frame” tutorial. However, you should be able to avoid creating an additional “broadcaster” node. Please take a look at the code that already exists within go_to_goal.py.

Submit smooth1.py, smooth2.py, and smooth1_tf.py