Project 3: Control

Implement the project by modifying project_3 package provided to you and changing the name to look like <my_directory_id>_project_3 (the package is further referred as ‘project_3’ for simplicity). Hand in this project by uploading the package via the ELMS website.

Most of the guidelines (as well as starter code) are designed for Python. C++ developers will get some additional extra credit for their implementations.

DELIVERABLES:
- <my_directory_id>_project_3 - package with your code
- Link to video for exercise 1.4 (Testing on a real robot)
- Two pictures for part 2 (Potential fields based Control)

1. Control

In this part of the assignment, your goal is to program a robot to drive in an arbitrary polygon shape. The polygon will depend on two parameters: the number of sides to the polygon, and the length of each side. Your nodes will have to read these parameters from the command line (-n for number of sides and -d for the length of each side). For example the following command will drive the robot in a hexagon with side length 1 using open loop control:

```
rosrun <my_directory_id>_project_3 open_polygon.py -n 6 -d 1
```

To drive the robot, you will need to publish geometry_msgs/Twist messages, setting linear.x for forward motion and angular.z for turning. These speeds correspond to v and w from the Kinematics lecture notes. Velocity messages should be published on the /cmd_vel_mux/input/navi topic. Robot mobile base node will then figure out how to convert these velocities to wheel angular velocities Φ_l and Φ_r. Note that the top linear speed is 0.5 m/s and the top angular speed is 0.5 radians/s.

1.1. Open loop control

The first version of the algorithm will be open loop in project_3/src/open_polygon.py. It will figure out what velocity it needs to move for the specified angle/distance, and drive that velocity for a set period of time. The structure of the algorithm should be the following:

- Drive distance d
- Rotate by angle
- Go to step 1
To manage the time aspect of the rst script, use the rospy Time library. For instance, one way to publish a message for a set amount of time is as follows:

```python
r = rospy.Rate(10)
t = rospy.Time.now()
while rospy.Time.now() - t < rospy.Duration(number_of_seconds):
    pub.publish(msg)
r.sleep()
```

### 1.2. Proportional control

The second version of the algorithm will have proportional control in `project_3/src/prop_polygon.py`. Implement the closed loop feedback control law described in Control lecture notes slide 31:

\[
\nu = k_p \varphi \\
\omega = k_{\alpha} \alpha + k_{\beta} \beta
\]

where \( \nu \) and \( \omega \) are the linear and angular speeds of the robot, \( \varphi, \alpha, \beta \) is the configuration of the robot expressed in polar coordinates. You must choose appropriate values for \( k_p, k_\alpha \) and \( k_\beta \) to ensure that your system is stable.

To get the robot's current position you can query the **tf package**:

```python
# Initialize transform listener
tfListener = tf.TransformListener()
...
# Get current transform between robot base and robot start point
(position, orientation) = tfListener.lookupTransform("/odom", "/base_footprint", rospy.Time(0))
# Convert pose orientation from quaternion to Euler angles
orientation = tf.transformations.euler_from_quaternion(orientation)
```

For more details on how to use tf: [Python](#), [C++](#)

### 1.3. Testing in simulation

To test your open loop and proportional control programs you can use the Turtlebot simulator. To launch the simulator use

```bash
roslaunch project_3 project_3_world.launch
```

This will open a Gazebo simulator windows with a Turtlebot on an empty infinite plane. To test your code just run it in a new terminal window and see if the robot moves as you expect. If you get the following errors:

```
```
Error [Node.cc:90] No namespace found

simply relaunch the simulator until it works

1.4. Testing on a real robot
Once you have made sure that everything works fine in the simulator you will have to test your code on a real robot. You will use your proportional control program to drive a robot in a square with side length of 1.5 meters. You will have to submit a link to a video of your robot driving (it would be nice if you put it on YouTube). See course webpage for instructions on how to setup the robot and an example video.

2. Potential fields based Control

We model the potential function $U$ and its gradient field $F = -\nabla U$ as a sum of the attractive field of the goal and the repulsive field of the obstacles as given in the lecture slides (Planning and Control: slides 22 and 23). Consider a point like robot in the workspace with the area $[0; 100][0; 100]$. Represent the obstacles in the environment as circles with centers at $[50; 20]$ and $[80; 35]$ each with radius 5. Assume that the initial position of the robot is $x_0 = [20; 20]$. In potential_field_nav.py write a function which takes as input arbitrary goal position in the workspace and returns the trajectory which the robot followed to get to the goal using potential field based method. The parameters of the potential functions can be set as variables inside of the function. The control commands generated by potential field based controller are $(\dot{x}, \dot{y})^T = F$.

Create two plots demonstrating the capability of the robot to avoid obstacles and reach the goal. Your plots should include the potential field together with the path taken by your robot. Use the matplotlib library (http://matplotlib.org/) to make the plots. For C++, you can use OpenCV or export your data in CSV and plot it with an external tool.

3. Extra credit

Organize your code neatly and create launch files for all of your nodes. Create your GitHub account (if you do not have one), fork our ‘adventure’ repository and configure your machine with contrib/INSTALL.py script (use -h for the list of options). Understand how INSTALL.py works and what it can do and then:
- Make your 1.4 exercise launch at laptop startup (10 points)
- Make Xtion, web camera and gripper driver nodes launch at laptop startup, then show the TA that you can see the outputs in Rviz and control the gripper from a remote computer. We are using a generic web camera, Asus Xtion PRO RGBD sensor and PhantomX Pincher robot arm (10 points)
4. Grading

<table>
<thead>
<tr>
<th>Control Type</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open loop control</td>
<td>30</td>
</tr>
<tr>
<td>Proportional control</td>
<td>30</td>
</tr>
<tr>
<td>Real robot video</td>
<td>20</td>
</tr>
<tr>
<td>Potential fields control</td>
<td>20</td>
</tr>
<tr>
<td>Extra credit</td>
<td>20</td>
</tr>
</tbody>
</table>