Homework 2

MacMillan, Kyle

September 19, 2018

Contents

1 Repository

Here is the repository for all homework, and here is the repository for this assignment.

2 Problem 3.6.2

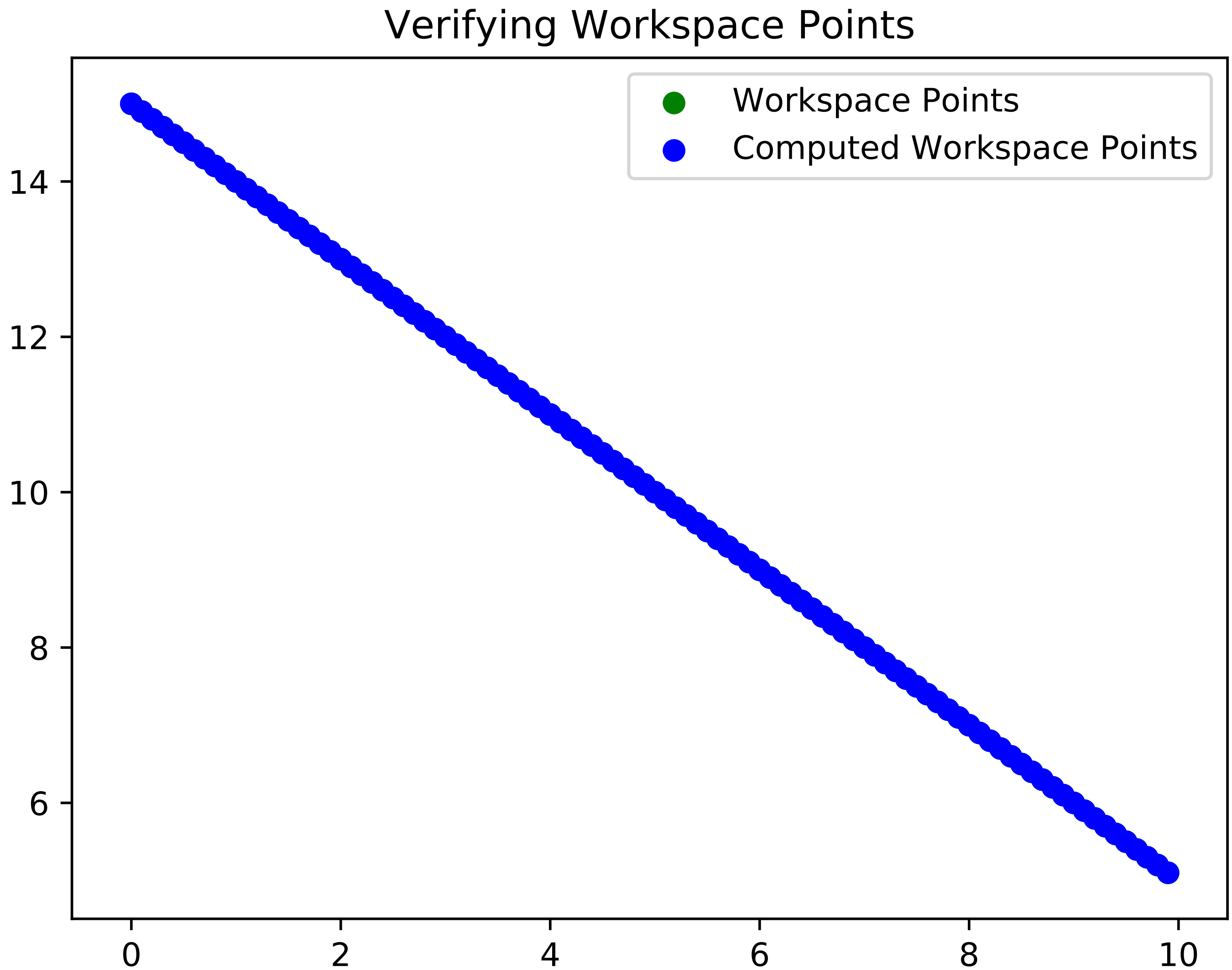
This code assumes you have downloaded ROSwrapper and included it in the folder.

```
import numpy as np                                # Numerical library
from std_msgs.msg import Float32MultiArray        # Message type
from ROSwrapper.nodecontrol import NodeControl    # ROS2 controller
from Problem3_2a import line1                     # Line generator
from iknode import IkNode                         # Derived RosNode
from iknode2 import IkNode2                       # Derived RosNode
import matplotlib.pyplot as plt                   # To plot data points


class twolink():
    """ This class is meant for fk and ik operations around a 2-link
        manipulator. This was updated from problem 10 to allow for
        starting theta values.
    """

    def __init__(self, length1, length2, path, rate):
        """ Class initialization """
        self.a1 = length1
        self.a2 = length2
        self.x = path[0]
        self.y = path[1]
        self.index = 0
        self.pts = zip(path[0], path[1])
        self.theta = (0.0, 0.0)
        self.plot_data_ik_x = []
        self.plot_data_ik_y = []
        self.plot_data_fk_x = []
        self.plot_data_fk_y = []
        self.showing_plot = False
        self.s_plot = plt.figure()

        # ROS init
        self.nc = NodeControl()
        self.nc.addnode(IkNode(name='node_xy',
                               obj=self,
                               pub_data_type=Float32MultiArray,
                               pub_chan='/physData',
                               pub_rate=5,
                               pub_data=self.pts))
        self.nc.addnode(IkNode(name='node_theta_magic',
                               obj=self,
                               sub_data_type=Float32MultiArray,
                               sub_chan='/physData',
                               pub_data_type=Float32MultiArray,
                               pub_chan='/thetaData',
```

```
                                pub_data=self.theta))
        self.nc.addnode(IkNode2(name='node_dual_sub',
                                obj=self,
                                sub_data_type=Float32MultiArray,
                                sub_chan=('/physData', '/thetaData')))

        self.nc.run()

    def getik(self, xy):
        """ Calculates the inverse kinematics to determine the theta1
            & theta2 values
        """
        x = xy[0]
        y = xy[1]
        theta1 = 0.0
        theta2 = 0.0
        D = (x * x + y * y - self.a1 * self.a1 - self.a2 * self.a2)\
            / (2 * self.a1 * self.a2)
        theta2 = np.arctan2(np.sqrt(1 - D * D), D)
        gamma = np.arctan2((self.a2 * np.sin(theta2)),
                           (self.a1 + self.a2 * np.cos(theta2)))
        theta1 = np.arctan2(y, x) - gamma

        return theta1, theta2

    def getfk(self, thetas):
        """ Calculate the forward kinematics to determine the x & y
            values
        """
        theta1 = thetas[0]
        theta2 = thetas[1]
        x = self.a2 * np.cos(theta1 + theta2) + \
            self.a1 * np.cos(theta1)
        y = self.a2 * np.sin(theta1 + theta2) + \
            self.a1 * np.sin(theta1)
        return x, y

    def append_plot_data_ik(self, data):
        if len(self.plot_data_ik_x) < 100:
            self.plot_data_ik_x.append(data[0])
            self.plot_data_ik_y.append(data[1])

    def append_plot_data_fk(self, data):
        if len(self.plot_data_fk_x) < 100:
            self.plot_data_fk_x.append(data[0])
            self.plot_data_fk_y.append(data[1])
        elif not self.showing_plot:
            plt.scatter(self.plot_data_ik_x,
                        self.plot_data_ik_y,
                        c='g',
```

```
                        label='Workspace Points')
            plt.scatter(self.plot_data_fk_x,
                        self.plot_data_fk_y,
                        c='b',
                        label='Computed Workspace Points')
            self.showing_plot = True
            plt.title('Verifying Workspace Points')
            plt.legend()
            plt.show()
            self.s_plot.savefig('Problem3_2c.pdf',
                                format='pdf',
                                dpi=1200)
            print('Press \"ctrl\" + \"c\" to exit')


def main():
    twolink(10, 10, path=line1(0, 10, 100), rate=5)


if __name__ == "__main__":
    main()
```

Figure 1: Problem 3.2



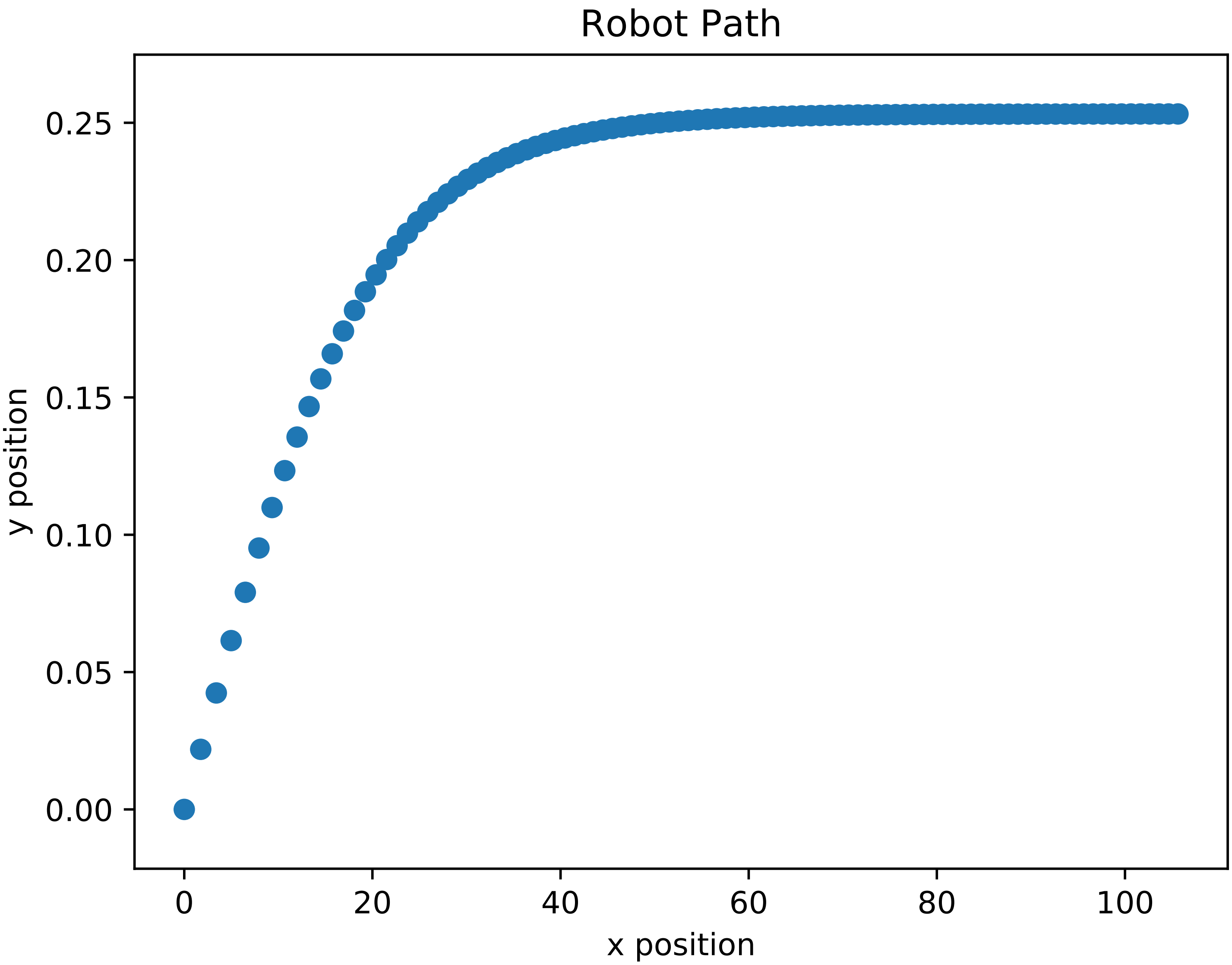
3 Problem 4.3

```
import numpy as np
from ROSwrapper.nodecontrol import NodeControl
from ROSwrapper.rosnode import RosNode
from wheelnode import WheelNode
from fknode import FkNode
from std_msgs.msg import Float32MultiArray, Int8
from geometry_msgs.msg import Twist
import matplotlib.pyplot as plt


class DiffDrive():
    def __init__(self, start, end, hz, D, L):
        self.radius = D / 2.0
        self.L = L
        self.pub_rate = hz
        self.time_step = 1.0 / hz
        t = np.arange(start, end + self.time_step, self.time_step)
        self.phi_1 = 2.0 + 2.0 * np.exp(-t)
        self.phi_2 = 2.0 + np.exp(-2.0 * t)
        self.phi_data = zip(self.phi_1, self.phi_2)
        self.inactive = False
        self.fk_data = Twist()
        self.fk_data.linear.x = 0.0
        self.fk_data.linear.y = 0.0
        self.fk_data.angular.x = 0.0
        self.plot_data_x = [0.0, ]
        self.plot_data_y = [0.0, ]
        self.showing_plot = False
        self.s_plot = plt.figure()

        self.initRosNodes()

    def initRosNodes(self):
        self.nc = NodeControl()
        self.nc.addnode(WheelNode(name='Control',
                                  obj=self,
                                  pub_chan='/WheelVel',
                                  pub_rate=self.pub_rate,
                                  pub_data_type=Float32MultiArray,
                                  pub_data=self.phi_data))

        # Naming this one to access it in done()
        self.active_node = self.nc.addnode(RosNode(name='Active',
                                                   obj=self,
                                                   pub_chan='/Active',
                                                   pub_rate=1,
                                                   pub_data_type=Int8,
```

```
                                                            pub_data=1))

        self.nc.addnode(FkNode(name='ForwardK',
                               obj=self,
                               sub_chan='/WheelVel',
                               sub_data_type=Float32MultiArray,
                               pub_chan='/RobotVel',
                               pub_data_type=Twist,
                               pub_data=self.fk_data))

        self.nc.addnode(FkNode(name='RobotPlot',
                               obj=self,
                               sub_chan='/RobotVel',
                               sub_data_type=Twist))

        self.nc.run()

    def done(self):
        self.active_node.pub_msg.data = 0
        self.inactive = True

    def fillFkData(self, w):
        thetaVel = (self.radius / (2.0 * self.L)) * (w[0] - w[1])
        theta = thetaVel * self.time_step
        gamma = (self.radius / 2.0) * (w[0] + w[1])

        # Fill Twist
        self.fk_data.linear.x = gamma * np.cos(theta)
        self.fk_data.linear.y = gamma * np.sin(theta)
        self.fk_data.angular.x = thetaVel

    def fillPlotData(self, xy):
        if len(self.plot_data_x) < 100:
            # position = x0 + v * time
            x = self.plot_data_x[-1] + xy[0] * self.time_step
            y = self.plot_data_y[-1] + xy[1] * self.time_step
            self.plot_data_x.append(x)
            self.plot_data_y.append(y)
        elif not self.showing_plot:
            self.showing_plot = True
            plt.scatter(self.plot_data_x,
                        self.plot_data_y,
                        label='Position')
            plt.xlabel('x position')
            plt.ylabel('y position')
            plt.title('Robot Path')
            plt.show()
            self.s_plot.savefig('Problem4_3c.pdf',
                                format='pdf',
                                dpi=1200)
```

```
            print('Press \"ctrl\" + \"c\" to exit')


if __name__ == '__main__':
    DiffDrive(start=0, end=10, hz=10, D=10, L=20)
```

Figure 2: Problem 4.3



4 Problem 4.13

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
from geometry_msgs.msg import Pose2D
import numpy as np
import itertools


class DiffDrive():
    def __init__(self, problem):
        self.problem = problem

        self.tri_pts = itertools.cycle(iter([(0.0, 0.0),
                                             (15.0, 0.0),
                                             (5.0, 20.0)]))

        self.sqr_pts = itertools.cycle(iter([(0,0),
                                             (10,0),
                                             (10,10),
                                             (0,10)]))
        self.position = (0.0, 0.0)
        self.theta = 0.0
        self.r = 0.25

        self.radius = 1.0
        self.L = 0.6
        self.time_step = 0.1

        # Buffer
        self.nose = (self.position[0] + self.r * np.cos(self.theta),
                     self.position[1] + self.r * np.sin(self.theta))

        # Radians
        self.tri_thetas = itertools.cycle(iter([0.0,
                                                0.000001,
                                                0.0]))
        self.sqr_thetas = itertools.cycle(iter([0.0,        # 0
                                                1.5707963,  # 90
                                                3.1415296,  # 180
                                                4.7123889]))  # 270

        self.rosinit()

```

```
    def rosinit(self):
        node_name = 'circle'
        rclpy.init()
        node = Node(node_name)
        msg = Float32()

        print('Spinning: {}'.format(node_name))
        self.publeft = node.create_publisher(Float32, '/wheel_left')
        self.pubright = node.create_publisher(Float32, '/wheel_right')
        if self.problem == 'triangle':
            self.goal_pt = next(self.tri_pts)
            self.goal_pt = next(self.tri_pts)
            self.goal_theta = next(self.tri_thetas)
            self.goal_theta = next(self.tri_thetas)
            msg.data = 1.0
            self.publeft.publish(msg)
            msg.data = -msg.data
            self.pubright.publish(msg)
            node.create_subscription(Pose2D, '/GPS', self.tri_callback)
        elif self.problem == 'square':
            self.goal_pt = next(self.sqr_pts)
            self.goal_pt = next(self.sqr_pts)
            self.goal_theta = next(self.sqr_thetas)
            self.goal_theta = next(self.sqr_thetas)
            msg.data = 1.0
            self.publeft.publish(msg)
            msg.data = -msg.data
            self.pubright.publish(msg)
            node.create_subscription(Pose2D, '/GPS', self.sqr_callback)
        else:
            # Circle
            msg.data = 5.0
            self.publeft.publish(msg)
            msg.data = msg.data * (15.0 + 1.2) / 15
            self.pubright.publish(msg)

        try:
            rclpy.spin(node)
        except KeyboardInterrupt:
            print('Shutting down...')
            msg.data = 0.0
            self.publeft.publish(msg)
            self.pubright.publish(msg)
            node.destroy_node()
            rclpy.shutdown()

    def dist(self, pt, xy):
        return np.sqrt((pt[0] - xy[0]) * (pt[0] - xy[0]) + ((pt[1] - xy[1]) * (pt[1

    def thetadiff(self, goal, theta):
```

```
        return theta - goal


    def tri_callback(self, msg):
        print('x,y:   {}, {}'.format(msg.x, msg.y))
        print('theta: {}'.format(msg.theta))
        dist = self.dist(self.goal_pt, (msg.x, msg.y))
        print('dist: {}'.format(dist))
        thetadist = self.thetadiff(self.goal_theta, msg.theta)
        print('thetadist: {}'.format(thetadist))

        # Ordering one wiggly bot!
        if dist > 1:
            new_msg = Float32()
            new_msg.data = 2.0 + 5 * thetadist
            self.publeft.publish(new_msg)
            new_msg.data = 2.0 - 5 * thetadist
            self.pubright.publish(new_msg)
        else:
            self.goal = next(self.tri_pts)
            self.goal_theta = next(self.tri_thetas)

    def sqr_callback(self, msg):
        print('x,y:   {}, {}'.format(msg.x, msg.y))
        print('theta: {}'.format(msg.theta))
        dist = self.dist(self.goal_pt, (msg.x, msg.y))
        print('dist: {}'.format(dist))
        thetadist = self.thetadiff(self.goal_theta, msg.theta)
        print('thetadist: {}'.format(thetadist))

        # Ordering one wiggly bot!
        if dist > 1:
            new_msg = Float32()
            new_msg.data = 2.0 + 5 * thetadist
            self.publeft.publish(new_msg)
            new_msg.data = 2.0 - 5 * thetadist
            self.pubright.publish(new_msg)
        else:
            self.goal_pt = next(self.sqr_pts)
            self.goal_theta = next(self.sqr_thetas)

if __name__ == '__main__':
    DiffDrive(problem='square')
```

Unfortunately Veranda and/or ROS2 (rclpy) has some sort of issue where even when rclpy is shut down correctly topics persist. Nothing is shown in the hidden topic list but they are there. This made simulation near impossible because I had to restart my computer to reset the robot. I had emailed Kali about it but she didn't respond. I eventually sought her out but it was too late. Here is and example of what happens if you kill the program (as demonstrated in the code) and then try to start it back up after a few minutes:

```
x,y:   -1.7935065362221152e+308, -1.7375489279272539e+308
theta: -1.737548927901094e+308
dist: inf
thetadist: 1.737548927901094e+308
```

Figure 3: Simulation Issues

5 Problem 4.14

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
from geometry_msgs.msg import Pose2D
import numpy as np
import itertools


class DiffDrive():
    def __init__(self, problem):
        self.problem = problem

        self.tri_pts = itertools.cycle(iter([(0.0, 0.0),
                                            (15.0, 0.0),
                                            (5.0, 20.0)]))

        self.sqr_pts = itertools.cycle(iter([(0,0),
                                            (10,0),
                                            (10,10),
                                            (0,10)]))
        self.position = (0.0, 0.0)
        self.theta = 0.0
        self.r = 0.25

        self.radius = 1.0
        self.L = 0.6
        self.time_step = 0.1

        # Buffer
        self.nose = (self.position[0] + self.r * np.cos(self.theta),
                     self.position[1] + self.r * np.sin(self.theta))

        # Radians
        self.tri_thetas = itertools.cycle(iter([0.0,
                                                0.000001,
                                                0.0]))
        self.sqr_thetas = itertools.cycle(iter([0.0,
# 0
                                                1.5707963,
# 90
                                                3.1415296,
# 180
                                                4.7123889]))
# 270

        self.rosinit()

```

```
    def rosinit(self):
        node_name = 'circle'
        rclpy.init()
        node = Node(node_name)
        msg = Float32()

        print('Spinning: {}'.format(node_name))
        self.publeft = node.create_publisher(Float32, '/wheel_left')
        self.pubright = node.create_publisher(Float32, '/wheel_right')
        if self.problem == 'triangle':
            self.goal_pt = next(self.tri_pts)
            self.goal_pt = next(self.tri_pts)
            self.goal_theta = next(self.tri_thetas)
            self.goal_theta = next(self.tri_thetas)
            msg.data = 1.0
            self.publeft.publish(msg)
            msg.data = -msg.data
            self.pubright.publish(msg)
            node.create_subscription(Pose2D, '/GPS', self.tri_callback)
        elif self.problem == 'square':
            self.goal_pt = next(self.sqr_pts)
            self.goal_pt = next(self.sqr_pts)
            self.goal_theta = next(self.sqr_thetas)
            self.goal_theta = next(self.sqr_thetas)
            msg.data = 1.0
            self.publeft.publish(msg)
            msg.data = -msg.data
            self.pubright.publish(msg)
            node.create_subscription(Pose2D, '/GPS', self.sqr_callback)
        else:
            # Circle
            msg.data = 5.0
            self.publeft.publish(msg)
            msg.data = msg.data * (15.0 + 1.2) / 15
            self.pubright.publish(msg)

        try:
            rclpy.spin(node)
        except KeyboardInterrupt:
            print('Shutting down...')
            msg.data = 0.0
            self.publeft.publish(msg)
            self.pubright.publish(msg)
            node.destroy_node()
            rclpy.shutdown()

    def dist(self, pt, xy):
        return np.sqrt((pt[0] - xy[0]) * (pt[0] - xy[0]) + ((pt[1] - xy[1]) * (pt[1

    def thetadiff(self, goal, theta):
```

```
        return theta - goal


    def tri_callback(self, msg):
        print('x,y:   {}, {}'.format(msg.x, msg.y))
        print('theta: {}'.format(msg.theta))
        dist = self.dist(self.goal_pt, (msg.x, msg.y))
        print('dist: {}'.format(dist))
        thetadist = self.thetadiff(self.goal_theta, msg.theta)
        print('thetadist: {}'.format(thetadist))

        # Ordering one wiggly bot!
        if dist > 1:
            new_msg = Float32()
            new_msg.data = 2.0 + 5 * thetadist
            self.publeft.publish(new_msg)
            new_msg.data = 2.0 - 5 * thetadist
            self.pubright.publish(new_msg)
        else:
            self.goal = next(self.tri_pts)
            self.goal_theta = next(self.tri_thetas)

    def sqr_callback(self, msg):
        print('x,y:   {}, {}'.format(msg.x, msg.y))
        print('theta: {}'.format(msg.theta))
        dist = self.dist(self.goal_pt, (msg.x, msg.y))
        print('dist: {}'.format(dist))
        thetadist = self.thetadiff(self.goal_theta, msg.theta)
        print('thetadist: {}'.format(thetadist))

        # Ordering one wiggly bot!
        if dist > 1:
            new_msg = Float32()
            new_msg.data = 2.0 + 5 * thetadist
            self.publeft.publish(new_msg)
            new_msg.data = 2.0 - 5 * thetadist
            self.pubright.publish(new_msg)
        else:
            self.goal_pt = next(self.sqr_pts)
            self.goal_theta = next(self.sqr_thetas)

if __name__ == '__main__':
    DiffDrive(problem='square')
```

6 Text Addition

The text addition I have included is ROSwrapper. The README gives an accurate description of what it is. Example usage is seen here and here. ROSwrapper is under development and as such there are some issues. Please be aware of them while using!