# Homework 2

MacMillan, Kyle September 19, 2018

## Contents

Table of Contents		i
1	Repository	1
2	Problem 3.6.2	2
3	Problem 4.3	6
4	Problem 4.13	10
5	Problem 4.14	14
6	Text Addition	17

# 1 Repository

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

#### 2 Problem 3.6.2

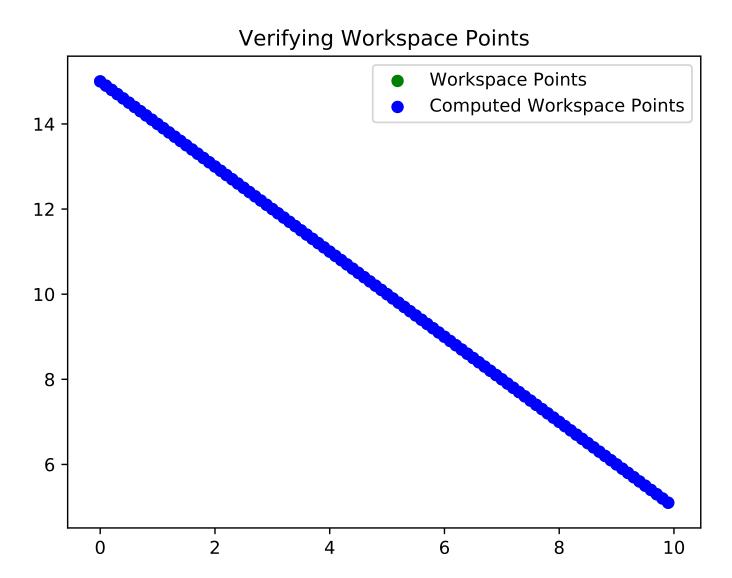
This code assumes you have downloaded ROSwrapper and included it in the folder.

```
1 | import numpy as np
                                                      # Numerical library
2 | 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
7
   import matplotlib.pyplot as plt
                                                      # To plot data points
8
9
   class twolink():
10
       """ This class is meant for fk and ik operations around a 2-link
11
12
           manipulator. This was updated from problem 10 to allow for
13
           starting theta values.
       . . . .
14
15
       def __init__(self, length1, length2, path, rate):
16
            """ Class initialization """
17
           self.a1 = length1
18
           self.a2 = length2
19
           self.x = path[0]
20
21
           self.y = path[1]
           self.index = 0
22
           self.pts = zip(path[0], path[1])
23
24
           self.theta = (0.0, 0.0)
25
           self.plot_data_ik_x = []
26
           self.plot_data_ik_y = []
27
           self.plot_data_fk_x = []
           self.plot_data_fk_y = []
28
           self.showing_plot = False
29
           self.s_plot = plt.figure()
30
31
32
           # ROS init
            self.nc = NodeControl()
33
           self.nc.addnode(IkNode(name='node_xy',
34
35
                                    obj=self,
36
                                    pub_data_type=Float32MultiArray,
37
                                    pub_chan='/physData',
38
                                    pub_rate=5,
39
                                    pub_data=self.pts))
           self.nc.addnode(IkNode(name='node_theta_magic',
40
41
                                    obj=self,
                                    sub_data_type=Float32MultiArray,
42
                                    sub_chan='/physData',
43
44
                                    pub_data_type=Float32MultiArray,
45
                                    pub_chan='/thetaData',
```

```
pub_data=self.theta))
46
47
            self.nc.addnode(IkNode2(name='node_dual_sub',
48
                                      obj=self,
49
                                      sub_data_type=Float32MultiArray,
50
                                      sub_chan=('/physData', '/thetaData')))
51
52
            self.nc.run()
53
54
       def getik(self, xy):
            """ Calculates the inverse kinematics to determine the theta1
55
                & theta2 values
56
57
58
            x = xy[0]
            y = xy[1]
59
60
            theta1 = 0.0
            theta2 = 0.0
61
62
            D = (x * x + y * y - self.a1 * self.a1 - self.a2 * self.a2) \setminus
                / (2 * self.a1 * self.a2)
63
64
            theta2 = np.arctan2(np.sqrt(1 - D * D), D)
65
            gamma = np.arctan2((self.a2 * np.sin(theta2)),
                                (self.a1 + self.a2 * np.cos(theta2)))
66
67
            theta1 = np.arctan2(y, x) - gamma
68
69
            return theta1, theta2
70
71
       def getfk(self, thetas):
72
            """ Calculate the forward kinematics to determine the x \& y
73
                values
74
            0.00
75
            theta1 = thetas[0]
76
            theta2 = thetas[1]
77
            x = self.a2 * np.cos(theta1 + theta2) + 
78
                self.a1 * np.cos(theta1)
            y = self.a2 * np.sin(theta1 + theta2) + 
79
80
                self.a1 * np.sin(theta1)
81
            return x, y
82
83
       def append_plot_data_ik(self, data):
84
            if len(self.plot_data_ik_x) < 100:</pre>
85
                self.plot_data_ik_x.append(data[0])
                self.plot_data_ik_y.append(data[1])
86
87
       def append_plot_data_fk(self, data):
88
            if len(self.plot_data_fk_x) < 100:</pre>
89
90
                self.plot_data_fk_x.append(data[0])
91
                self.plot_data_fk_y.append(data[1])
            elif not self.showing_plot:
92
93
                plt.scatter(self.plot_data_ik_x,
94
                             self.plot_data_ik_y ,
95
                             c='g',
```

```
label='Workspace Points')
96
97
                 plt.scatter(self.plot_data_fk_x,
98
                              self.plot_data_fk_y ,
                             c='b',
99
                             label='Computed Workspace Points')
100
101
                 self.showing_plot = True
                 plt.title('Verifying Workspace Points')
102
103
                plt.legend()
104
                plt.show()
105
                 self.s_plot.savefig('Problem3_2c.pdf',
                                      format = 'pdf',
106
                                      dpi=1200)
107
                 print('Press \"ctrl\" + \"c\" to exit')
108
109
110
111
    def main():
112
        twolink(10, 10, path=line1(0, 10, 100), rate=5)
113
114
115 | if __name__ == "__main__":
116
    main()
```

Figure 1: Problem 3.2

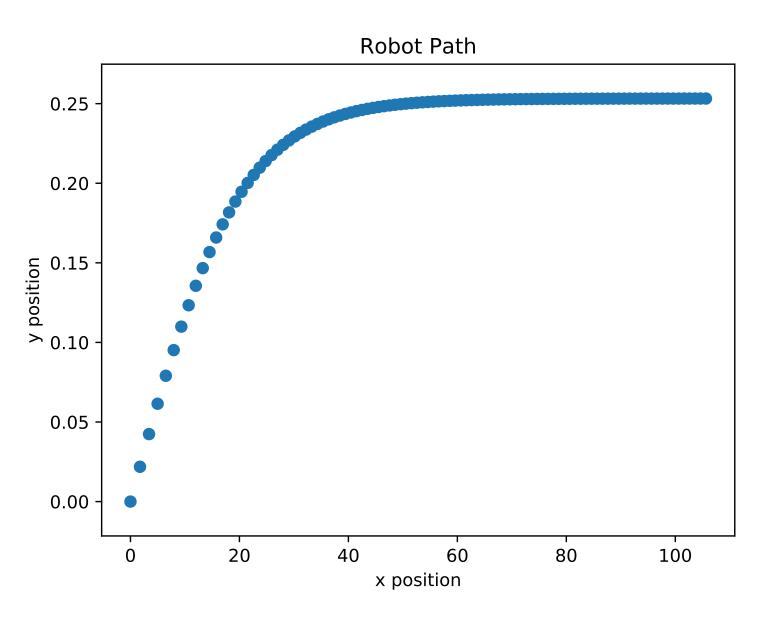


#### 3 Problem 4.3

```
1 | import numpy as np
2 | from ROSwrapper.nodecontrol import NodeControl
3 from ROSwrapper.rosnode import RosNode
4 from wheelnode import WheelNode
5 from fknode import FkNode
   from std_msgs.msg import Float32MultiArray, Int8
7
   from geometry_msgs.msg import Twist
   import matplotlib.pyplot as plt
9
10
   class DiffDrive():
11
       def __init__(self, start, end, hz, D, L):
12
            self.radius = D / 2.0
13
14
           self.L = L
           self.pub_rate = hz
15
16
           self.time\_step = 1.0 / hz
           t = np.arange(start, end + self.time_step, self.time_step)
17
18
           self.phi_1 = 2.0 + 2.0 * np.exp(-t)
19
           self.phi_2 = 2.0 + np.exp(-2.0 * t)
20
           self.phi_data = zip(self.phi_1, self.phi_2)
           self.inactive = False
21
22
           self.fk_data = Twist()
           self.fk_data.linear.x = 0.0
23
24
           self.fk_data.linear.y = 0.0
25
           self.fk_data.angular.x = 0.0
26
           self.plot_data_x = [0.0,]
27
           self.plot_data_y = [0.0, ]
28
           self.showing_plot = False
29
           self.s_plot = plt.figure()
30
31
           self.initRosNodes()
32
33
       def initRosNodes(self):
34
           self.nc = NodeControl()
35
           self.nc.addnode(WheelNode(name='Control',
36
                                       obj=self,
                                       pub_chan='/WheelVel',
37
38
                                       pub_rate=self.pub_rate,
39
                                       pub_data_type=Float32MultiArray,
40
                                       pub_data=self.phi_data))
41
42
           # Naming this one to access it in done()
           self.active_node = self.nc.addnode(RosNode(name='Active',
43
44
                                                         obj=self,
45
                                                         pub_chan='/Active',
46
                                                         pub_rate=1,
47
                                                         pub_data_type=Int8,
```

```
pub_data=1))
48
49
50
            self.nc.addnode(FkNode(name='ForwardK',
51
                                     obj=self,
52
                                     sub_chan='/WheelVel',
53
                                     sub_data_type=Float32MultiArray,
54
                                     pub_chan='/RobotVel',
55
                                     pub_data_type=Twist,
56
                                     pub_data=self.fk_data))
57
            self.nc.addnode(FkNode(name='RobotPlot',
58
59
                                     obj=self,
60
                                     sub_chan='/RobotVel',
                                     sub_data_type=Twist))
61
62
            self.nc.run()
63
64
       def done(self):
65
66
            self.active_node.pub_msg.data = 0
67
            self.inactive = True
68
69
       def fillFkData(self, w):
70
            thetaVel = (self.radius / (2.0 * self.L)) * (w[0] - w[1])
            theta = thetaVel * self.time_step
71
            gamma = (self.radius / 2.0) * (w[0] + w[1])
72
73
74
            # Fill Twist
            self.fk_data.linear.x = gamma * np.cos(theta)
75
76
            self.fk_data.linear.y = gamma * np.sin(theta)
77
            self.fk_data.angular.x = thetaVel
78
79
       def fillPlotData(self, xy):
80
            if len(self.plot_data_x) < 100:</pre>
                \# position = x0 + v * time
81
82
                x = self.plot_data_x[-1] + xy[0] * self.time_step
83
                y = self.plot_data_y[-1] + xy[1] * self.time_step
                self.plot_data_x.append(x)
84
85
                self.plot_data_y.append(y)
86
            elif not self.showing_plot:
87
                self.showing_plot = True
88
                plt.scatter(self.plot_data_x,
89
                             self.plot_data_y,
90
                             label='Position')
91
                plt.xlabel('x position')
92
                plt.ylabel('y position')
93
                plt.title('Robot Path')
94
                plt.show()
95
                self.s_plot.savefig('Problem4_3c.pdf',
96
                                      format = 'pdf',
                                      dpi=1200)
97
```

Figure 2: Problem 4.3



### 4 Problem 4.13

```
1 | 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
7
8
9
   class DiffDrive():
10
       def __init__(self, problem):
            self.problem = problem
11
12
13
            self.tri_pts = itertools.cycle(iter([(0.0, 0.0),
14
                                               (15.0, 0.0),
15
                                               (5.0, 20.0)]))
16
17
            self.sqr_pts = itertools.cycle(iter([(0,0),
18
                                               (10,0),
19
                                               (10,10),
20
                                               (0,10)])
21
            self.position = (0.0, 0.0)
            self.theta = 0.0
22
            self.r = 0.25
23
24
            self.radius = 1.0
25
            self.L = 0.6
26
27
            self.time\_step = 0.1
28
29
            # Buffer
            self.nose = (self.position[0] + self.r * np.cos(self.theta),
30
31
                          self.position[1] + self.r * np.sin(self.theta))
32
33
            # Radians
34
            self.tri_thetas = itertools.cycle(iter([0.0,
                                                       0.000001,
35
36
                                                       0.0]))
            self.sqr_thetas = itertools.cycle(iter([0.0,
37
                                                       1.5707963,
38
     90
                                                       3.1415296,
39
   # 180
                                                       4.7123889]))
40
   # 270
41
42
            self.rosinit()
43
```

```
def rosinit(self):
44
45
            node_name = 'circle'
46
           rclpy.init()
47
           node = Node(node_name)
           msg = Float32()
48
49
50
           print('Spinning: {}'.format(node_name))
51
            self.publeft = node.create_publisher(Float32, '/wheel_left')
            self.pubright = node.create_publisher(Float32, '/wheel_right')
52
            if self.problem == 'triangle':
53
                self.goal_pt = next(self.tri_pts)
54
55
                self.goal_pt = next(self.tri_pts)
56
                self.goal_theta = next(self.tri_thetas)
                self.goal_theta = next(self.tri_thetas)
57
                msg.data = 1.0
58
                self.publeft.publish(msg)
59
60
                msg.data = -msg.data
61
                self.pubright.publish(msg)
62
                node.create_subscription(Pose2D, '/GPS', self.tri_callback)
63
            elif self.problem == 'square':
                self.goal_pt = next(self.sqr_pts)
64
                self.goal_pt = next(self.sqr_pts)
65
                self.goal_theta = next(self.sqr_thetas)
66
67
                self.goal_theta = next(self.sqr_thetas)
68
                msg.data = 1.0
69
                self.publeft.publish(msg)
70
                msg.data = -msg.data
71
                self.pubright.publish(msg)
72
                node.create_subscription(Pose2D, '/GPS', self.sqr_callback)
            else:
73
74
                # Circle
75
                msg.data = 5.0
76
                self.publeft.publish(msg)
77
                msg.data = msg.data * (15.0 + 1.2) / 15
78
                self.pubright.publish(msg)
79
80
            try:
81
                rclpy.spin(node)
82
            except KeyboardInterrupt:
                print('Shutting down...')
83
84
                msg.data = 0.0
                self.publeft.publish(msg)
85
                self.pubright.publish(msg)
86
87
                node.destroy_node()
88
                rclpy.shutdown()
89
90
       def dist(self, pt, xy):
            return np.sqrt((pt[0] - xy[0]) * (pt[0] - xy[0]) + ((pt[1] - xy[1]) * (pt[1
91
92
93
       def thetadiff(self, goal, theta):
```

```
94
            return theta - goal
95
96
97
        def tri_callback(self, msg):
                           {}, {}'.format(msg.x, msg.y))
98
            print('x,y:
99
            print('theta: {}'.format(msg.theta))
100
            dist = self.dist(self.goal_pt, (msg.x, msg.y))
101
            print('dist: {}'.format(dist))
102
            thetadist = self.thetadiff(self.goal_theta, msg.theta)
            print('thetadist: {}'.format(thetadist))
103
104
105
            # Ordering one wiggly bot!
106
            if dist > 1:
                 new_msg = Float32()
107
                 new_msg.data = 2.0 + 5 * thetadist
108
                 self.publeft.publish(new_msg)
109
                 new_msg.data = 2.0 - 5 * thetadist
110
111
                 self.pubright.publish(new_msg)
112
            else:
113
                 self.goal = next(self.tri_pts)
114
                 self.goal_theta = next(self.tri_thetas)
115
        def sqr_callback(self, msg):
116
                           {}, {}'.format(msg.x, msg.y))
117
            print('x,y:
            print('theta: {}'.format(msg.theta))
118
119
            dist = self.dist(self.goal_pt, (msg.x, msg.y))
            print('dist: {}'.format(dist))
120
121
            thetadist = self.thetadiff(self.goal_theta, msg.theta)
            print('thetadist: {}'.format(thetadist))
122
123
            # Ordering one wiggly bot!
124
125
            if dist > 1:
126
                 new_msg = Float32()
                new_msg.data = 2.0 + 5 * thetadist
127
128
                 self.publeft.publish(new_msg)
129
                new_msg.data = 2.0 - 5 * thetadist
                 self.pubright.publish(new_msg)
130
131
            else:
132
                 self.goal_pt = next(self.sqr_pts)
133
                 self.goal_theta = next(self.sqr_thetas)
134
135
    if __name__ == '__main__':
136
        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

```
1 | 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
7
8
9
   class DiffDrive():
10
       def __init__(self, problem):
            self.problem = problem
11
12
13
            self.tri_pts = itertools.cycle(iter([(0.0, 0.0),
14
                                               (15.0, 0.0),
15
                                               (5.0, 20.0)]))
16
17
            self.sqr_pts = itertools.cycle(iter([(0,0),
18
                                               (10,0),
19
                                               (10,10),
20
                                               (0,10)]))
21
            self.position = (0.0, 0.0)
            self.theta = 0.0
22
            self.r = 0.25
23
24
            self.radius = 1.0
25
            self.L = 0.6
26
27
            self.time\_step = 0.1
28
            # Buffer
29
            self.nose = (self.position[0] + self.r * np.cos(self.theta),
30
31
                          self.position[1] + self.r * np.sin(self.theta))
32
33
            # Radians
34
            self.tri_thetas = itertools.cycle(iter([0.0,
                                                       0.000001,
35
36
                                                       0.0]))
            self.sqr_thetas = itertools.cycle(iter([0.0,
37
                                                       1.5707963,
38
     90
                                                       3.1415296,
39
   # 180
                                                       4.7123889]))
40
   # 270
41
42
            self.rosinit()
43
```

```
def rosinit(self):
44
45
            node_name = 'circle'
46
           rclpy.init()
47
           node = Node(node_name)
           msg = Float32()
48
49
50
           print('Spinning: {}'.format(node_name))
51
            self.publeft = node.create_publisher(Float32, '/wheel_left')
            self.pubright = node.create_publisher(Float32, '/wheel_right')
52
            if self.problem == 'triangle':
53
                self.goal_pt = next(self.tri_pts)
54
55
                self.goal_pt = next(self.tri_pts)
56
                self.goal_theta = next(self.tri_thetas)
                self.goal_theta = next(self.tri_thetas)
57
                msg.data = 1.0
58
                self.publeft.publish(msg)
59
60
                msg.data = -msg.data
61
                self.pubright.publish(msg)
62
                node.create_subscription(Pose2D, '/GPS', self.tri_callback)
63
            elif self.problem == 'square':
                self.goal_pt = next(self.sqr_pts)
64
                self.goal_pt = next(self.sqr_pts)
65
                self.goal_theta = next(self.sqr_thetas)
66
67
                self.goal_theta = next(self.sqr_thetas)
68
                msg.data = 1.0
69
                self.publeft.publish(msg)
70
                msg.data = -msg.data
71
                self.pubright.publish(msg)
72
                node.create_subscription(Pose2D, '/GPS', self.sqr_callback)
            else:
73
74
                # Circle
75
                msg.data = 5.0
76
                self.publeft.publish(msg)
77
                msg.data = msg.data * (15.0 + 1.2) / 15
78
                self.pubright.publish(msg)
79
80
            try:
81
                rclpy.spin(node)
82
            except KeyboardInterrupt:
                print('Shutting down...')
83
84
                msg.data = 0.0
                self.publeft.publish(msg)
85
                self.pubright.publish(msg)
86
87
                node.destroy_node()
88
                rclpy.shutdown()
89
90
       def dist(self, pt, xy):
            return np.sqrt((pt[0] - xy[0]) * (pt[0] - xy[0]) + ((pt[1] - xy[1]) * (pt[1
91
92
93
       def thetadiff(self, goal, theta):
```

```
94
            return theta - goal
95
96
97
        def tri_callback(self, msg):
                           {}, {}'.format(msg.x, msg.y))
98
            print('x,y:
99
            print('theta: {}'.format(msg.theta))
            dist = self.dist(self.goal_pt, (msg.x, msg.y))
100
            print('dist: {}'.format(dist))
101
102
            thetadist = self.thetadiff(self.goal_theta, msg.theta)
            print('thetadist: {}'.format(thetadist))
103
104
105
            # Ordering one wiggly bot!
106
            if dist > 1:
                new_msg = Float32()
107
108
                new_msg.data = 2.0 + 5 * thetadist
109
                self.publeft.publish(new_msg)
110
                new_msg.data = 2.0 - 5 * thetadist
                self.pubright.publish(new_msg)
111
112
            else:
113
                 self.goal = next(self.tri_pts)
114
                self.goal_theta = next(self.tri_thetas)
115
        def sqr_callback(self, msg):
116
                         {}, {}'.format(msg.x, msg.y))
117
            print('x,y:
            print('theta: {}'.format(msg.theta))
118
119
            dist = self.dist(self.goal_pt, (msg.x, msg.y))
120
            print('dist: {}'.format(dist))
121
            thetadist = self.thetadiff(self.goal_theta, msg.theta)
122
            print('thetadist: {}'.format(thetadist))
123
124
            # Ordering one wiggly bot!
125
            if dist > 1:
126
                new_msg = Float32()
127
                new_msg.data = 2.0 + 5 * thetadist
128
                self.publeft.publish(new_msg)
129
                new_msg.data = 2.0 - 5 * thetadist
130
                self.pubright.publish(new_msg)
131
            else:
132
                 self.goal_pt = next(self.sqr_pts)
133
                self.goal_theta = next(self.sqr_thetas)
134
135 | 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!