

# **Homework 2**

MacMillan, Kyle

September 19, 2018

# Contents

Title

|                   |    |
|-------------------|----|
| 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    | 11 |
| 6 Text Addition   | 12 |

# 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
3 from ROSwrapper.nodecontrol import NodeControl    # ROS2 controller
4 from Problem3_2a import line1                     # Line generator
5 from iknode import IkNode                         # Derived RosNode
6 from iknode2 import IkNode2                      # Derived RosNode
7 import matplotlib.pyplot as plt                   # To plot data points
8
9
10 class twolink():
11     """ This class is meant for fk and ik operations around a 2-link
12         manipulator. This was updated from problem 10 to allow for
13         starting theta values.
14     """
15
16     def __init__(self, length1, length2, path, rate):
17         """ Class initialization """
18         self.a1 = length1
19         self.a2 = length2
20         self.x = path[0]
21         self.y = path[1]
22         self.index = 0
23         self.pts = zip(path[0], path[1])
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 = []
28         self.plot_data_fk_y = []
29         self.showing_plot = False
30         self.s_plot = plt.figure()
31
32     # ROS init
33     self.nc = NodeControl()
34     self.nc.addnode(IkNode(name='node_xy',
35                             obj=self,
36                             pub_data_type=Float32MultiArray,
37                             pub_chan='/physData',
38                             pub_rate=5,
39                             pub_data=self.pts))
40     self.nc.addnode(IkNode(name='node_theta_magic',
41                             obj=self,
42                             sub_data_type=Float32MultiArray,
43                             sub_chan='/physData',
44                             pub_data_type=Float32MultiArray,
45                             pub_chan='/thetaData',
```

```

46         pub_data=self.theta))
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):
55         """ Calculates the inverse kinematics to determine the theta1
56             & theta2 values
57         """
58         x = xy[0]
59         y = xy[1]
60         theta1 = 0.0
61         theta2 = 0.0
62         D = (x * x + y * y - self.a1 * self.a1 - self.a2 * self.a2)\
63             / (2 * self.a1 * self.a2)
64         theta2 = np.arctan2(np.sqrt(1 - D * D), D)
65         gamma = np.arctan2((self.a2 * np.sin(theta2)),
66                             (self.a1 + self.a2 * np.cos(theta2)))
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         """
75         theta1 = thetas[0]
76         theta2 = thetas[1]
77         x = self.a2 * np.cos(theta1 + theta2) + \
78             self.a1 * np.cos(theta1)
79         y = self.a2 * np.sin(theta1 + theta2) + \
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:
85             self.plot_data_ik_x.append(data[0])
86             self.plot_data_ik_y.append(data[1])
87
88     def append_plot_data_fk(self, data):
89         if len(self.plot_data_fk_x) < 100:
90             self.plot_data_fk_x.append(data[0])
91             self.plot_data_fk_y.append(data[1])
92         elif not self.showing_plot:
93             plt.scatter(self.plot_data_ik_x,
94                         self.plot_data_ik_y,
95                         c='g',

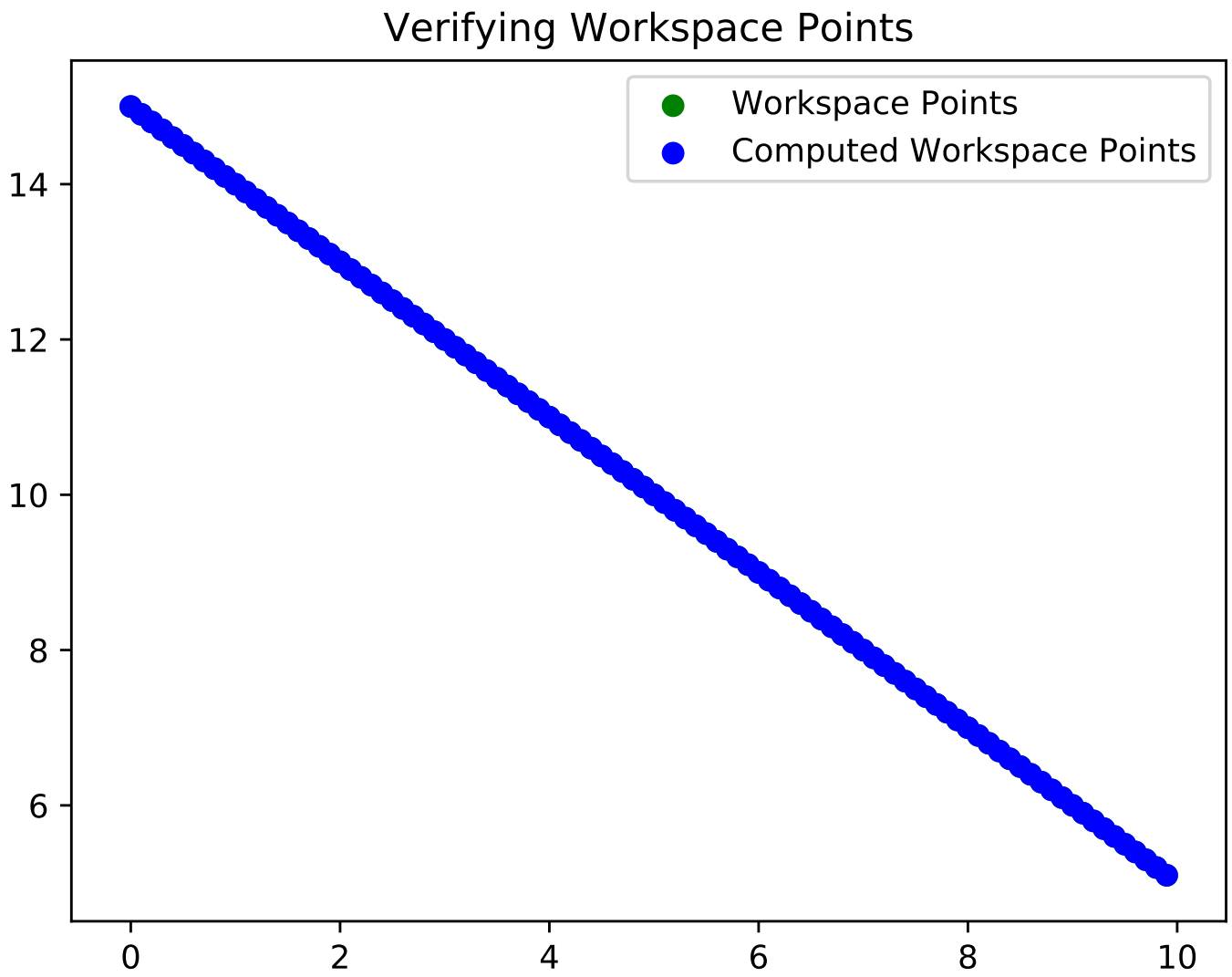
```

```

96         label='Workspace Points')
97     plt.scatter(self.plot_data_fk_x,
98                 self.plot_data_fk_y,
99                 c='b',
100                 label='Computed Workspace Points')
101     self.showing_plot = True
102     plt.title('Verifying Workspace Points')
103     plt.legend()
104     plt.show()
105     self.s_plot.savefig('Problem3_2c.pdf',
106                         format='pdf',
107                         dpi=1200)
108     print('Press \"ctrl\" + \"c\" to exit')
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
6 from std_msgs.msg import Float32MultiArray, Int8
7 from geometry_msgs.msg import Twist
8 import matplotlib.pyplot as plt
9
10
11 class DiffDrive():
12     def __init__(self, start, end, hz, D, L):
13         self.radius = D / 2.0
14         self.L = L
15         self.pub_rate = hz
16         self.time_step = 1.0 / hz
17         t = np.arange(start, end + self.time_step, self.time_step)
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)
21         self.inactive = False
22         self.fk_data = Twist()
23         self.fk_data.linear.x = 0.0
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,
37                                   pub_chan='/WheelVel',
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()
43         self.active_node = self.nc.addnode(RosNode(name='Active',
44                                                    obj=self,
45                                                    pub_chan='/Active',
46                                                    pub_rate=1,
47                                                    pub_data_type=Int8,
```



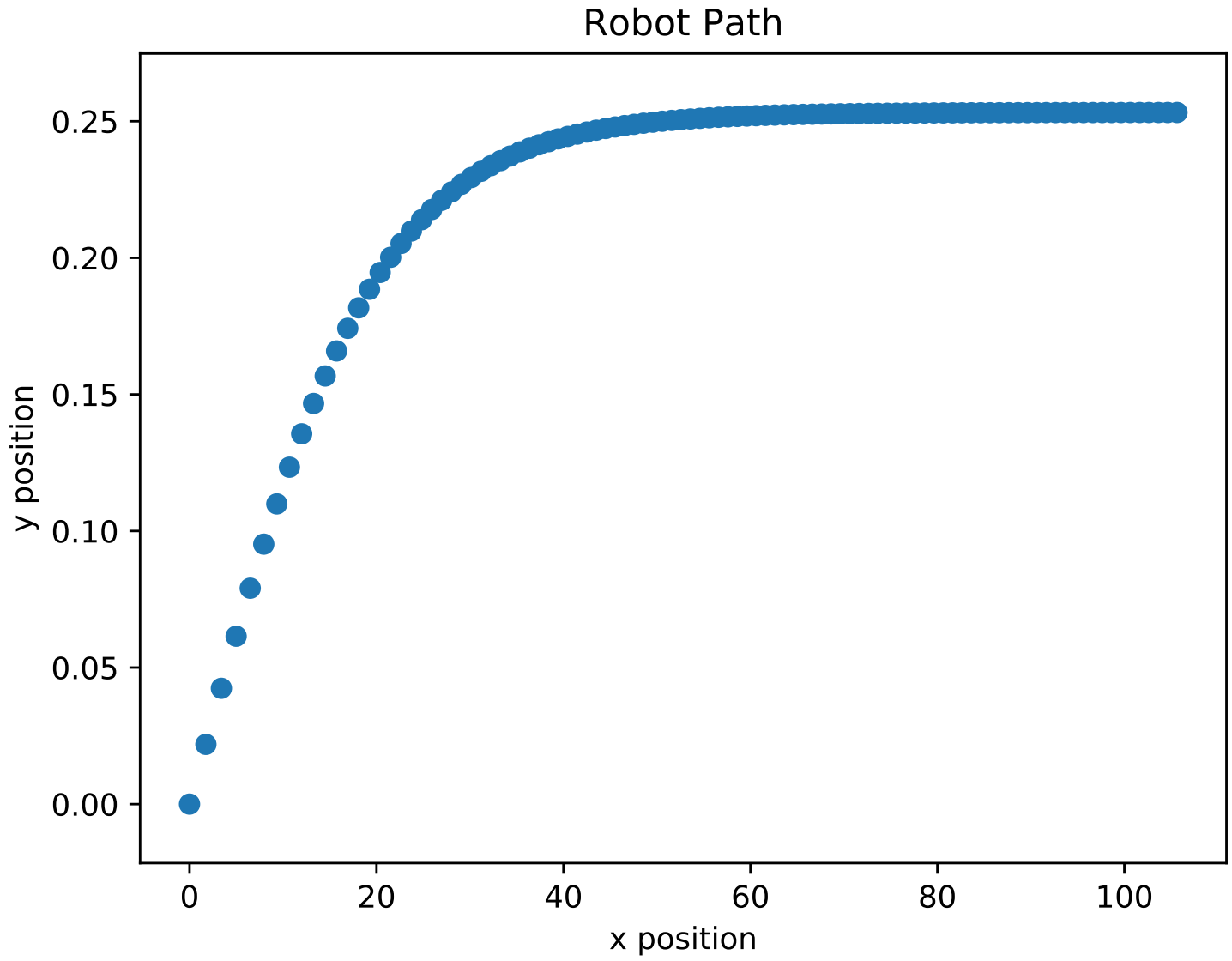
```

48                                     pub_data=1))
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
58     self.nc.addnode(FkNode(name='RobotPlot',
59                             obj=self,
60                             sub_chan='/RobotVel',
61                             sub_data_type=Twist))
62
63     self.nc.run()
64
65     def done(self):
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])
71         theta = thetaVel * self.time_step
72         gamma = (self.radius / 2.0) * (w[0] + w[1])
73
74         # Fill Twist
75         self.fk_data.linear.x = gamma * np.cos(theta)
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:
81             # position = x0 + v * time
82             x = self.plot_data_x[-1] + xy[0] * self.time_step
83             y = self.plot_data_y[-1] + xy[1] * self.time_step
84             self.plot_data_x.append(x)
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',
97                                 dpi=1200)

```

```
98         print('Press \'ctrl\' + \'c\' to exit')
99
100
101 if __name__ == '__main__':
102     DiffDrive(start=0, end=10, hz=10, D=10, L=20)
```

Figure 2: Problem 4.3



## 4 Problem 4.13

```
1  
2  
3  
4  
5 if __name__ == '__main__':  
6     print('started')
```

## 5 Problem 4.14

```
1  
2  
3  
4  
5 if __name__ == '__main__':  
6     print('started')
```

## 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!