Problem 1:

```
import numpy as np
from utils import wrapToPi

import rospy
from std_msgs.msg import Float64

# command zero velocities once we are this close to the goal
RHO_THRES = 0.05
ALPHA_THRES = 0.1
DELTA_THRES = 0.1

# Modified for section 6 to be a sub-node
class PoseController:
    """ Pose stabilization controller """
    def __init__(self, k1, k2, k3, V_max=0.5, om_max=1):
        self.k1 = k1
        self.k2 = k2
        self.k3 = k3

        self.V_max = V_max
        self.om_max = om_max

        self.last_alpha = None
        self.last_delta = None
        self.last_theta = None

        # ROS stuff
        self.pub_alpha = rospy.Publisher('/controller/alpha', Float64,
queue_size=10)
        self.pub_delta = rospy.Publisher('/controller/delta', Float64,
queue_size=10)
        self.pub_theta = rospy.Publisher('/controller/theta', Float64,
queue_size=10)

    def publish(self):
        if self.last_alpha is not None:
            self.pub_alpha.publish(self.last_alpha)
            self.pub_delta.publish(self.last_delta)
            self.pub_theta.publish(self.last_theta)
```

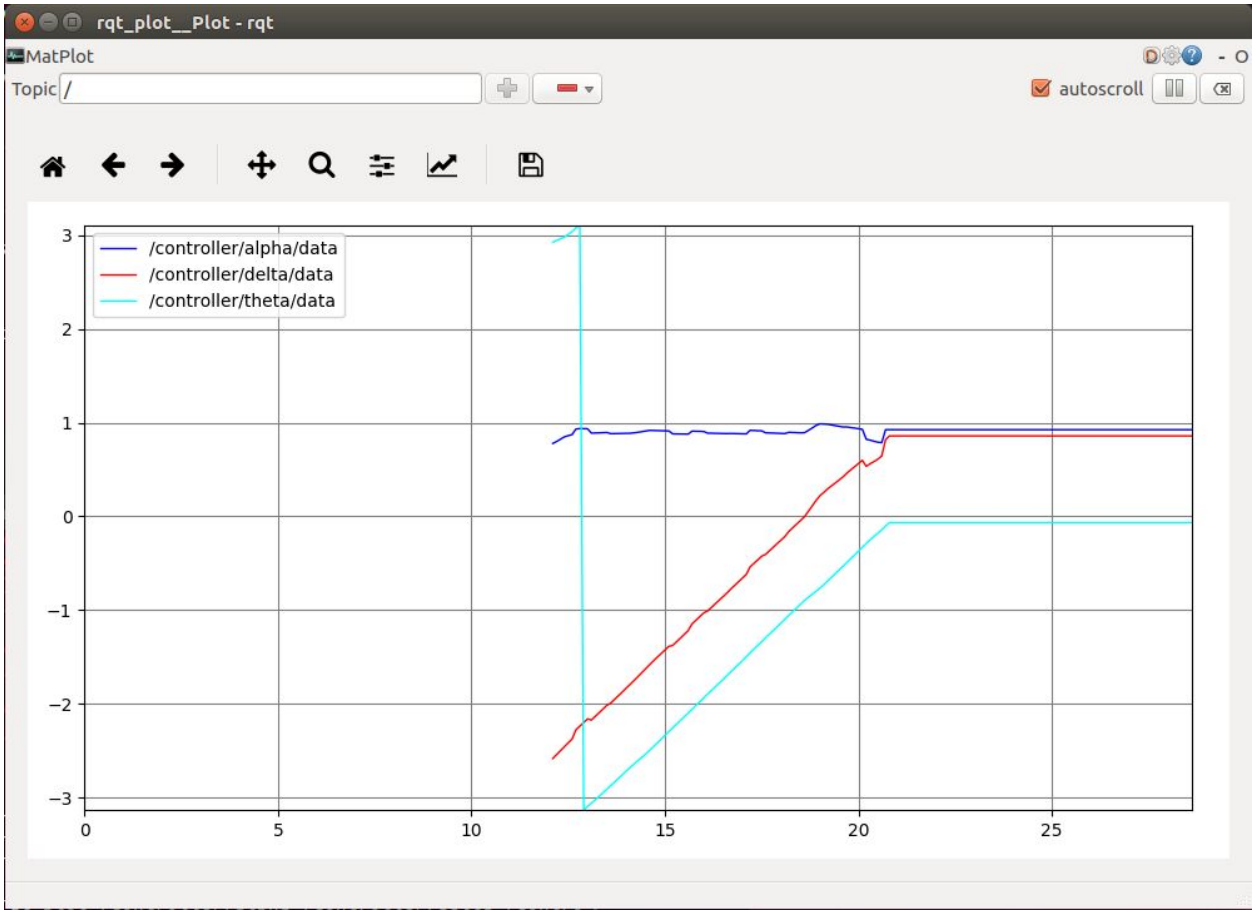
```
    def load_goal(self, x_g, y_g, th_g):
        """ Loads in a new goal position """
        self.x_g = x_g
        self.y_g = y_g
        self.th_g = th_g

    def compute_control(self, x, y, th, t):
        """
        Inputs:
            x,y,th: Current state
            t: Current time (you shouldn't need to use this)
        Outputs:
            V, om: Control actions

        Hints: You'll need to use the wrapToPi function. The np.sinc
function
        may also be useful, look up its documentation
        """
        ########## Code starts here ##########
        k1, k2, k3 = self.k1, self.k2, self.k3
        x_g, y_g, th_g = self.x_g, self.y_g, self.th_g

        # Frame translation
        x = x - x_g
        y = y - y_g

        # Frame rotation. Theta expressed in terms of original frame.
        x_rot = x*np.cos(th_g) + y*np.sin(th_g)
        y_rot = -x*np.sin(th_g) + y*np.cos(th_g)
        x, y = x_rot, y_rot

        # Relative angle
        th = th - th_g

        # These are given by equation (8) in lecture 4 notes, which
        # are derived by Lyapunov analysis.
        rho = np.sqrt(x*x + y*y)
        alpha = wrapToPi(np.arctan2(y,x) - th + np.pi)
        delta = wrapToPi(alpha + th)
```

```
        # These are given by equation (4) in the problem set. Also slide 15
lecture 4.
        # Literature: Closed loop steering of unicycle-like vehicles via
Lyapunov techniques.
        # M. Aicardi et. al. 1995.
        V = k1 * rho * np.cos(alpha)
        om = k2 * alpha + k1 * np.sinc(alpha/np.pi) * np.cos(alpha) *
(alpha + k3 * delta)

        # Clip velocity once we hit the requested threshold.
        if rho < RHO_THRES and alpha < ALPHA_THRES and delta < DELTA_THRES:
            V = 0
            om = 0

        # For section 6
        self.last_alpha = alpha
        self.last_delta = delta
        self.last_theta = th

        ########## Code ends here ##########

        # apply control limits
        V = np.clip(V, -self.V_max, self.V_max)
        om = np.clip(om, -self.om_max, self.om_max)

        return V, om
```

Problem 2:

`rosbag record -o <filepath> <topic1> <topic2> ..`

Problem 3:


rqt_plot__Plot - rqt
MatPlot
Topic /
autoscroll
/controller/alpha/data
/controller/delta/data
/controller/theta/data
3
2
1
0
−1
−2
−3
0
5
10
15
20
25