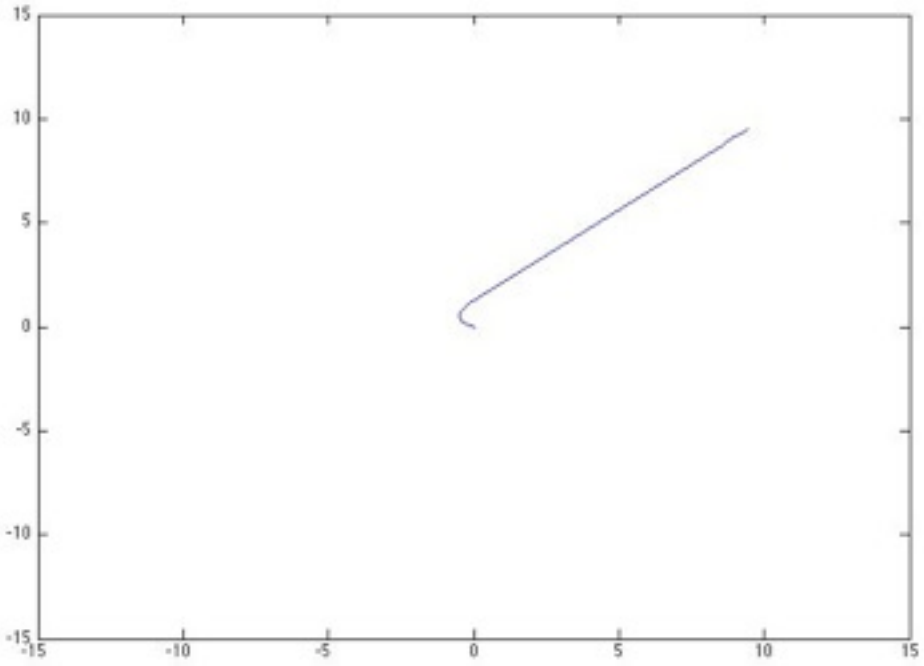
**Problem 1:**

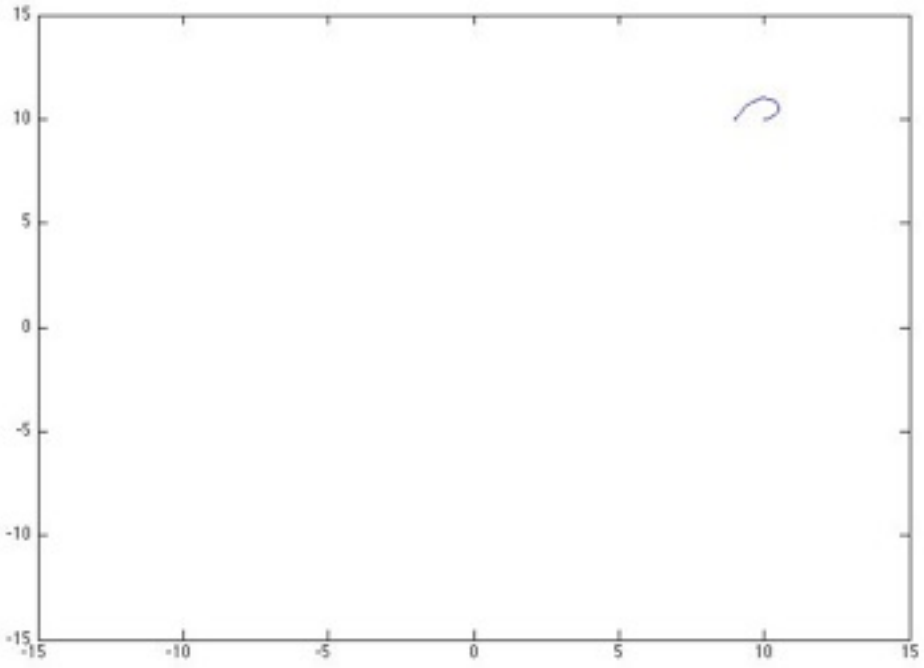
I resorted to using the toolkit because my implementation did not properly account for the range of the arc-tangent function. I have included the output of simulations using the toolkit as well as my implemented control model. This is something that I will fix in my own time, but I didn't want to delay turning in my homework because of this problem.

I have included my incorrect code and the plots indicating the problem at the end of this answer.

```
Solution 1:
x0 = [  0  0 pi ]; xg = [ 10 10    ];
```




```
Solution 2:
x0 = [ 10 10 0 ]; xg = [  9 10 ];
```

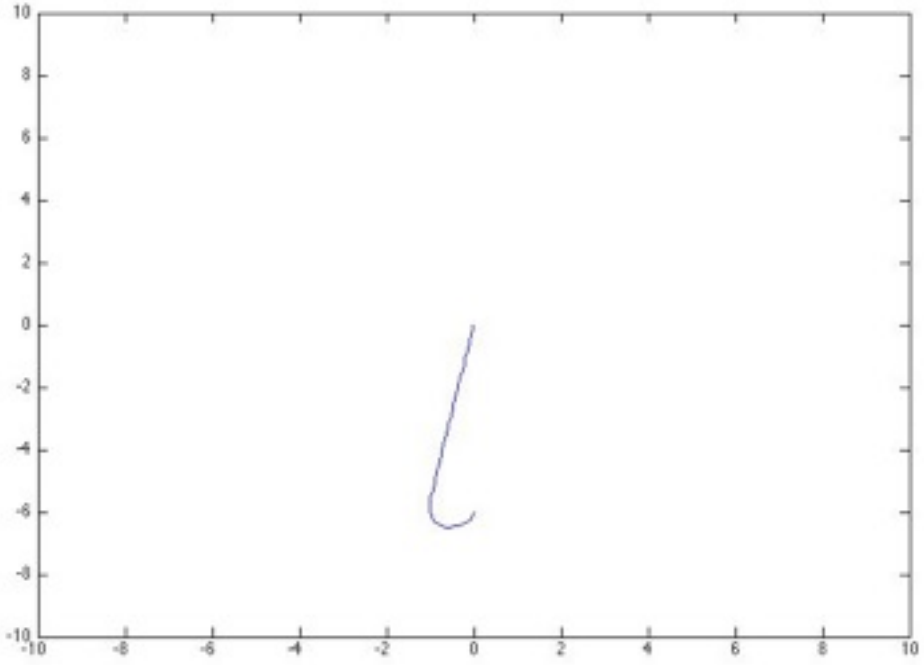



Solution 3:

```
x0 = [  0 -6 (3/2)*pi ]; xg = [  0   0 ];
```

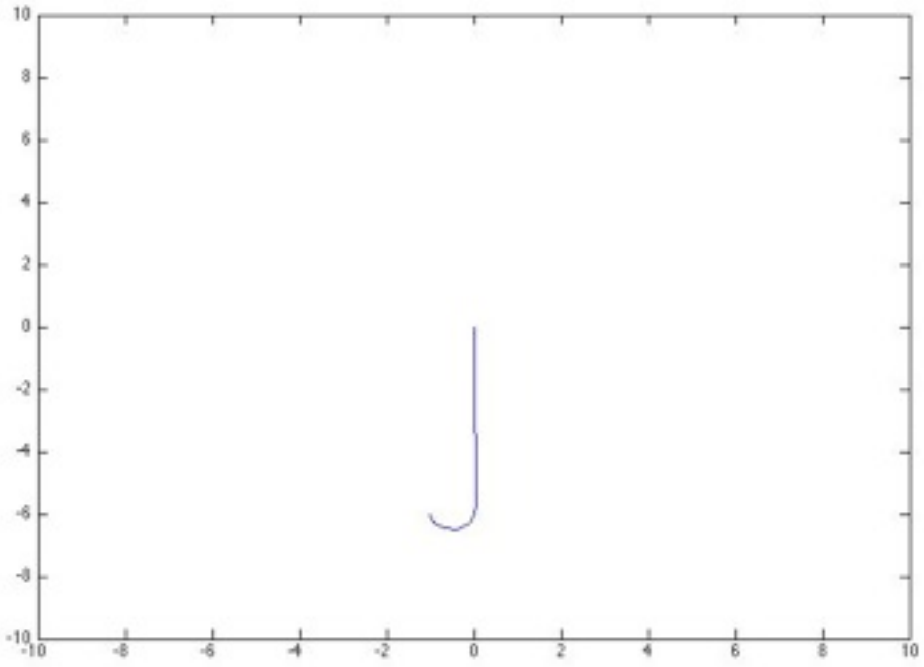
Solution 4:

```
x0 = [  -1 -6 (3/2)*pi ]; xg = [  0   0 ];
```



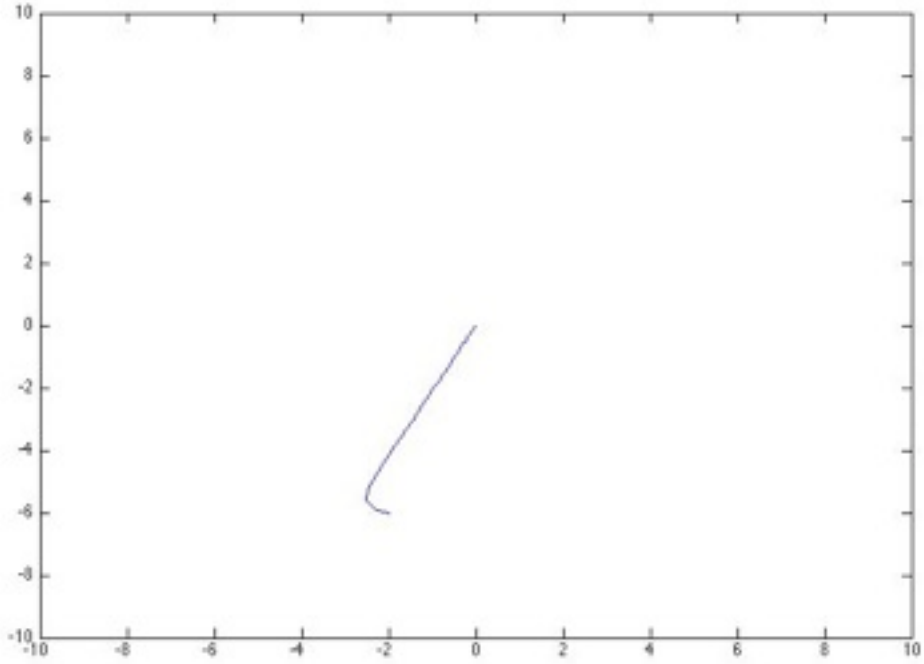
Solution 5:

```
x0 = [  -1 -6 (3/2)*pi ]; xg = [  0   0 ];
```



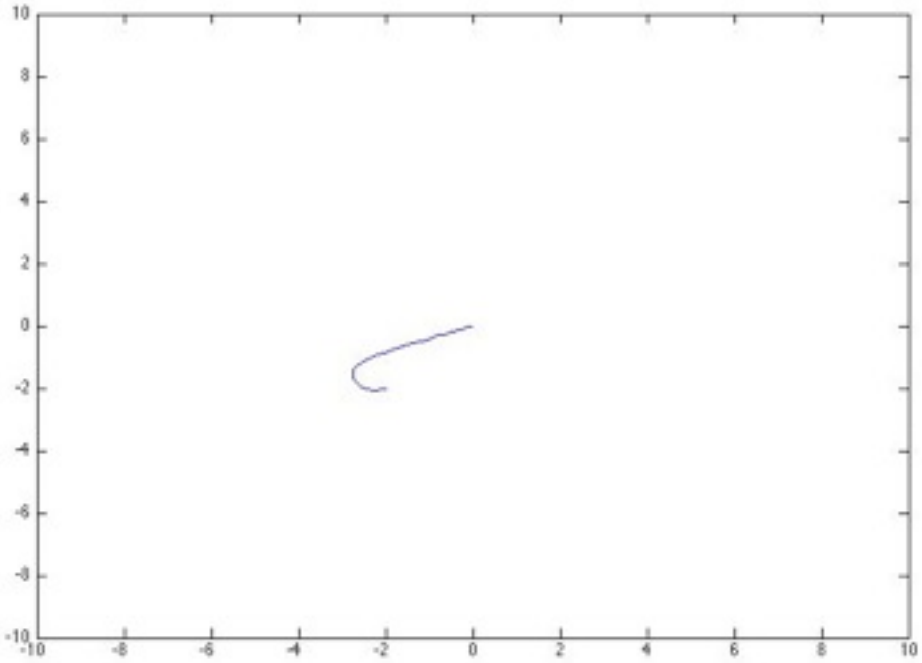

Solution 6:

```
x0 = [  -2 -6 pi ]; xg = [  0   0 ];
```



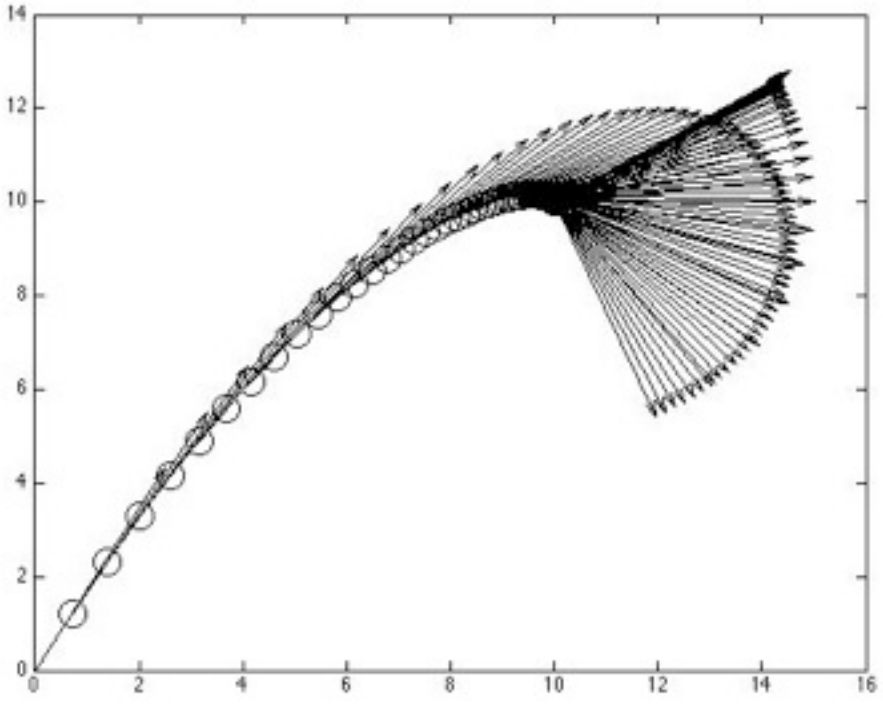
Solution 7:

```
x0 = [  -2 -2 (7*pi)/6 ]; xg = [  0   0 ];
```

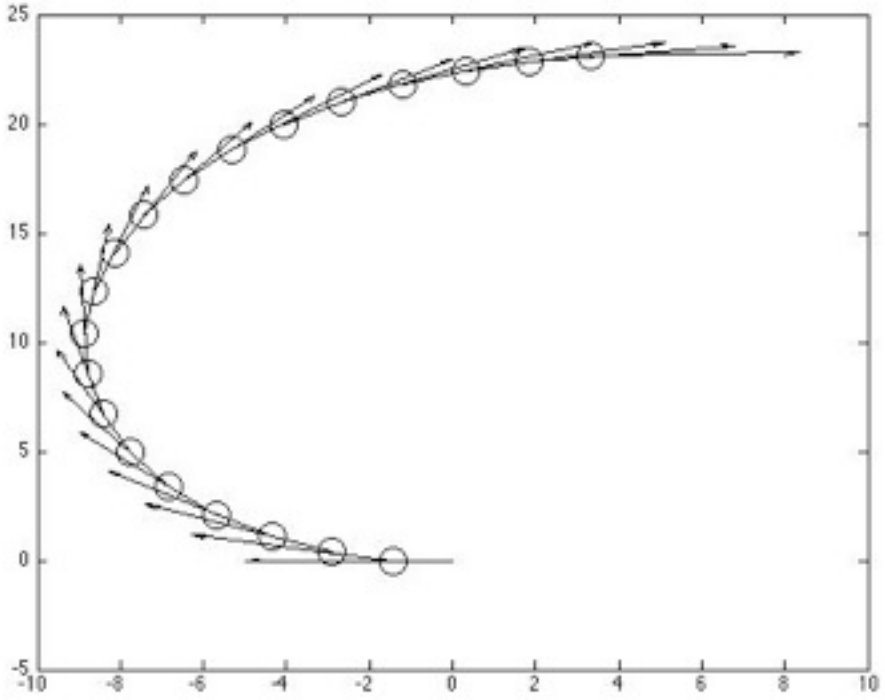



X1:

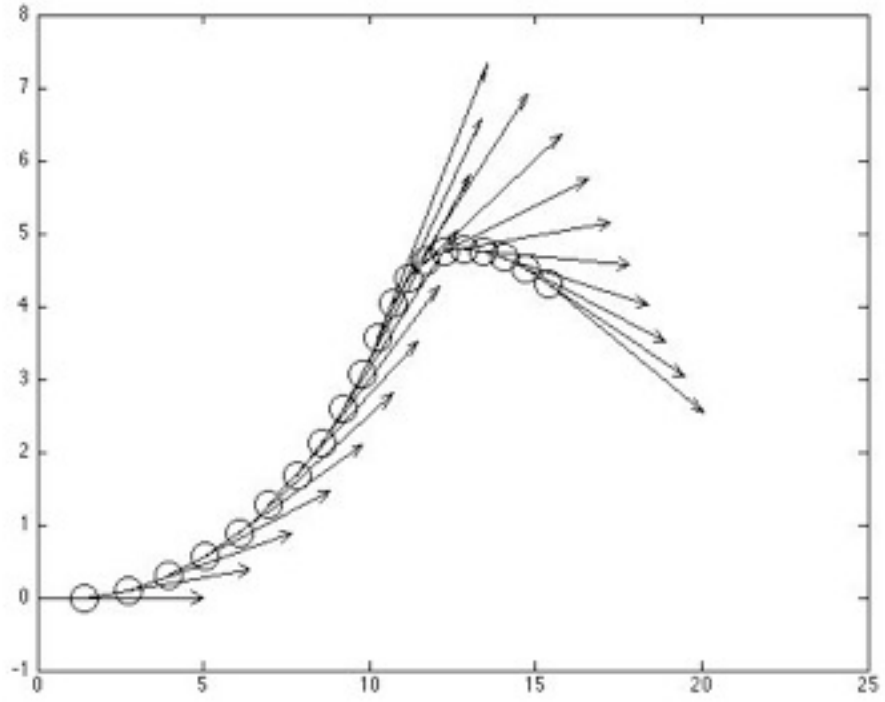
```
r = go_to_point(0, 0, pi/2, 10, 10);
```



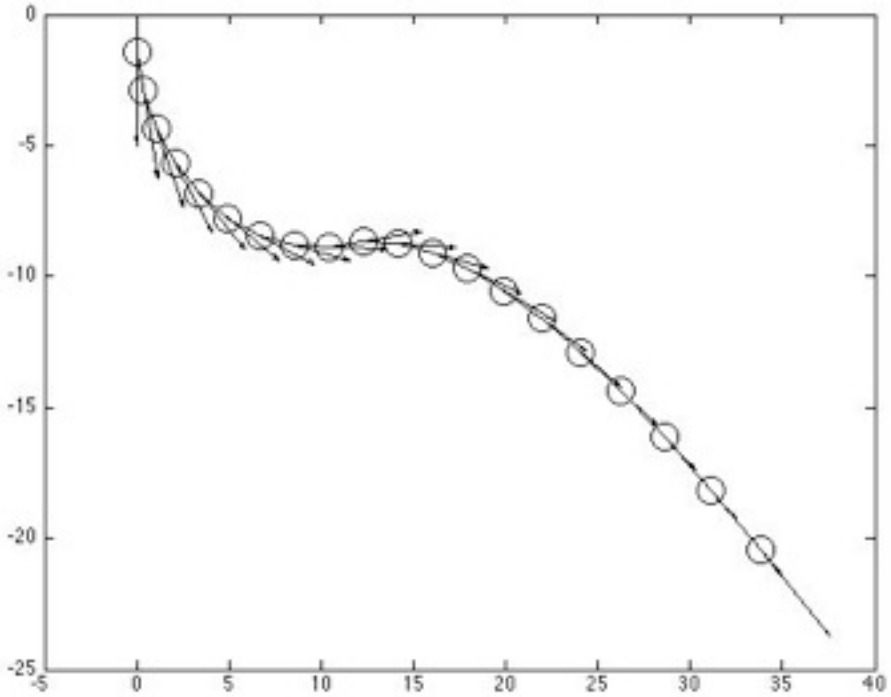
X2:
r = go_to_point(0, 0, pi, 10, 10);



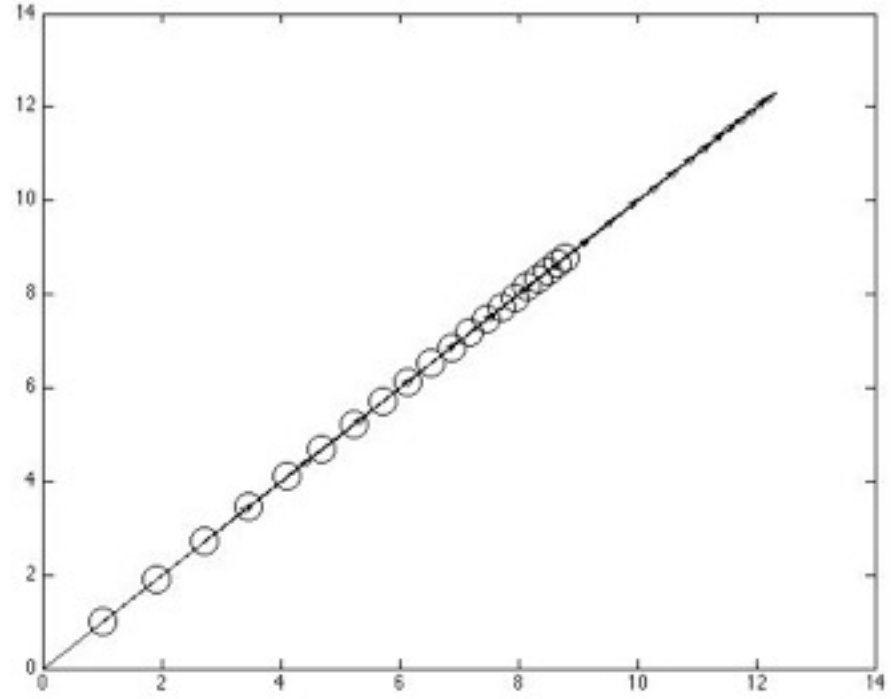

X3:
r = go_to_point(0, 0, 0, 10, 10);



X4:
r = go_to_point(0, 0, (3*pi/2), 10, 10);




X5:
r = go_to_point(0, 0, pi/4, 10, 10);



The offending code:

```
function [R] = go_to_point(x, y, theta, xd, yd)
  delta    = 1;
  xs       = [x];
  ys       = [y];
  thetas   = [theta];
  vs       = [0];
  vK      = 0.1;
  omegaK  = 0.1;
  plot_robot(x, y, theta);
  hold;

  for i=1:100 % eventually, check for being close enough to goal

    % The velocity is...
    v = vK * sqrt( (x-xd)^2 + (y-yd)^2 );

    % Relative angle between vehicle and goal
    theta_d = atan( (yd-y) / (xd-x) );

    % The angle is...
    omega  = omegaK * angdiff(theta_d, theta);

    x = cos(theta) * v * delta + x;
    y = sin(theta) * v * delta + y;
    theta = theta + omega * delta;
    xs      = [ xs, x ];
    ys      = [ ys, y ];
    thetas  = [ thetas, theta ];
    vs      = [ vs, v ];
    plot_robot(x, y, theta);
  end

  hold off

  R = [ xs ; ys ; thetas ];
end
```

Problem 2

As with the first problem, I decided to use the toolkit to perform simulations. I spent more time generating simulations to compensate for the lack of my own function.

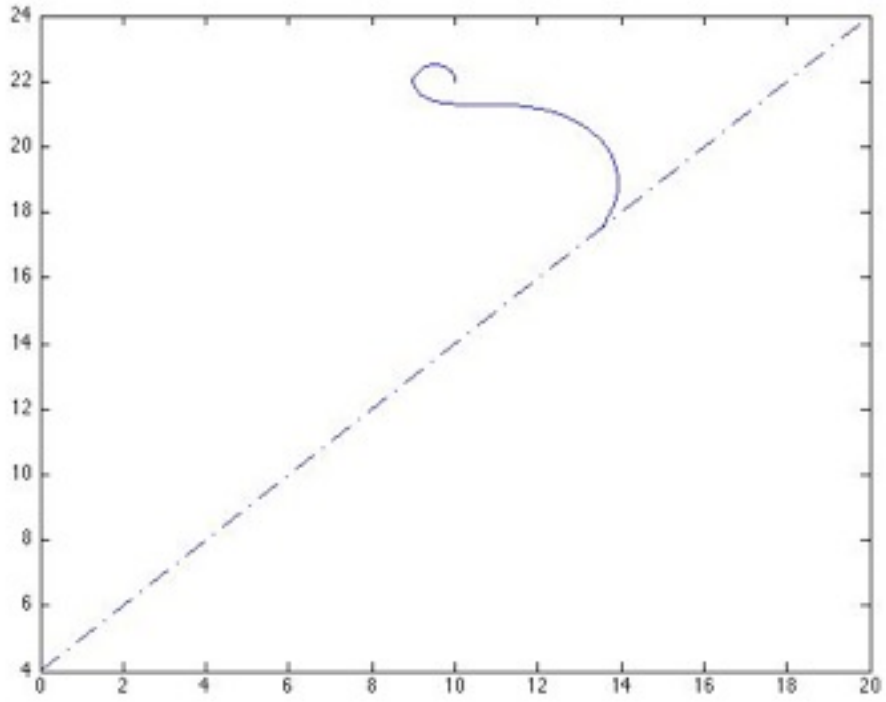
In the first experiment, I decided to create a line to follow defined by the vector:

L = [ 1 -1 4 ];

That is to say, the line:

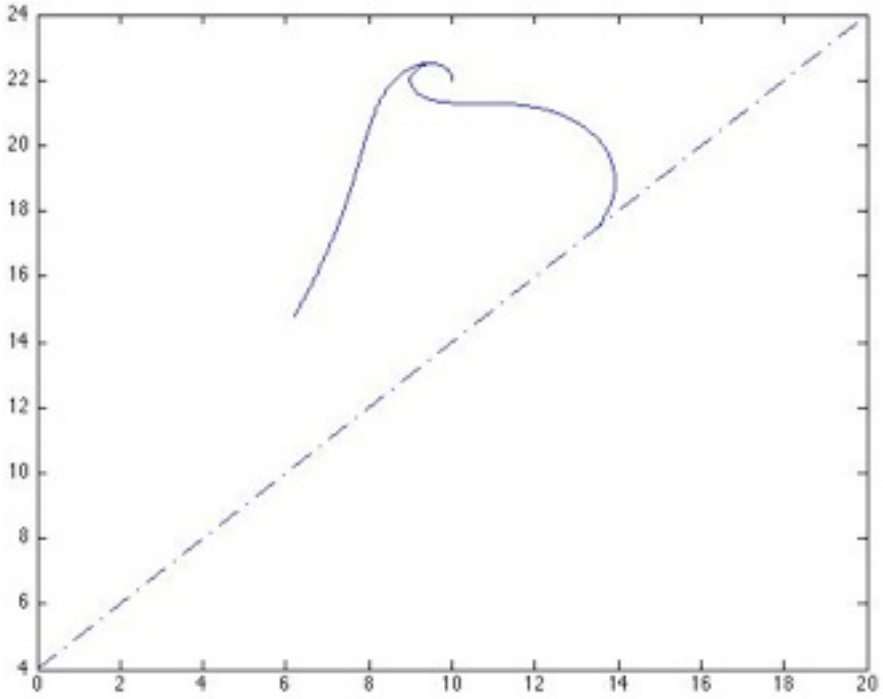
x - y + 4 => y = - (- x - 4) => y = x + 4 ;

Plot #1: Gain: 0.5, Kh: 1



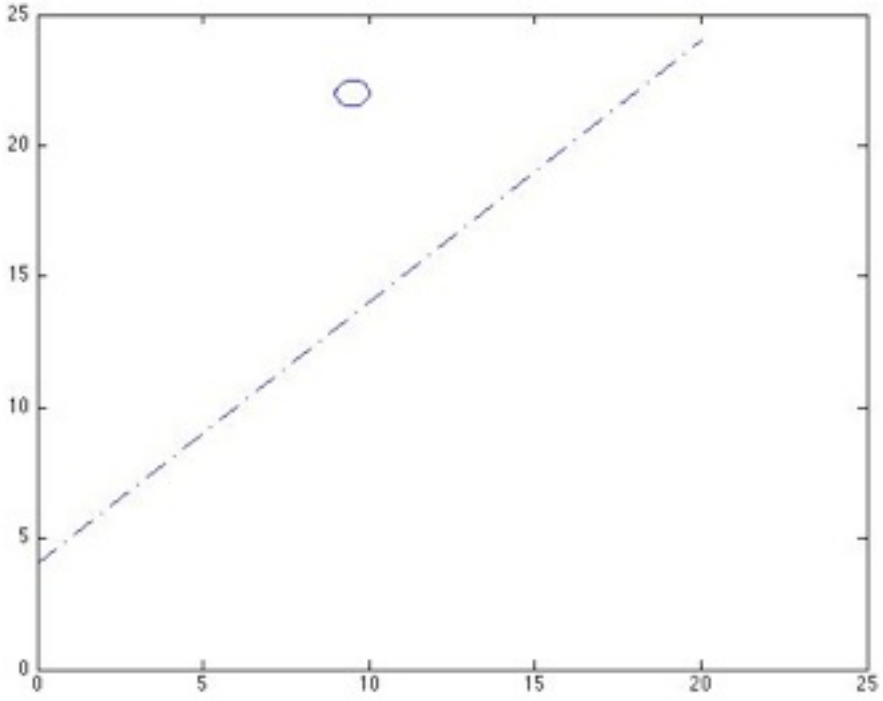

Plot #2: Gain: 0.1, Kh: 1

By reducing the velocity gain and holding the heading gain the path even more gradually approaches the line to follow. The trajectory on the left




Plot #3:

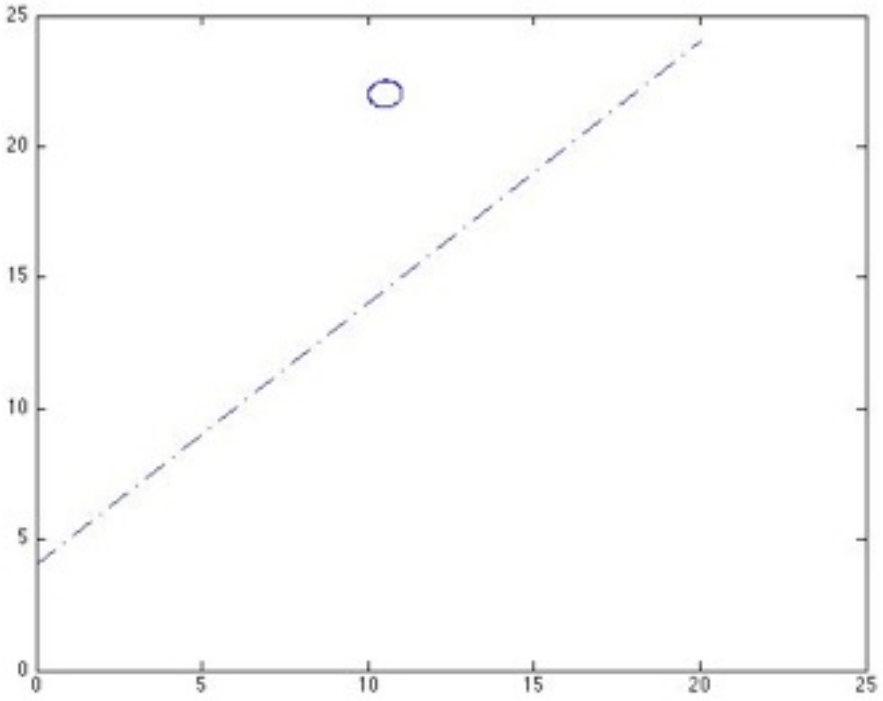
Gain: 1, Kh: 1, Initial: [ 10 22 pi/2 ]



I expected this to veer away from the line more initially because it was already far away and facing in the wrong direction. However, my prediction was wrong. The robot instead loops about and does not converge to the line.

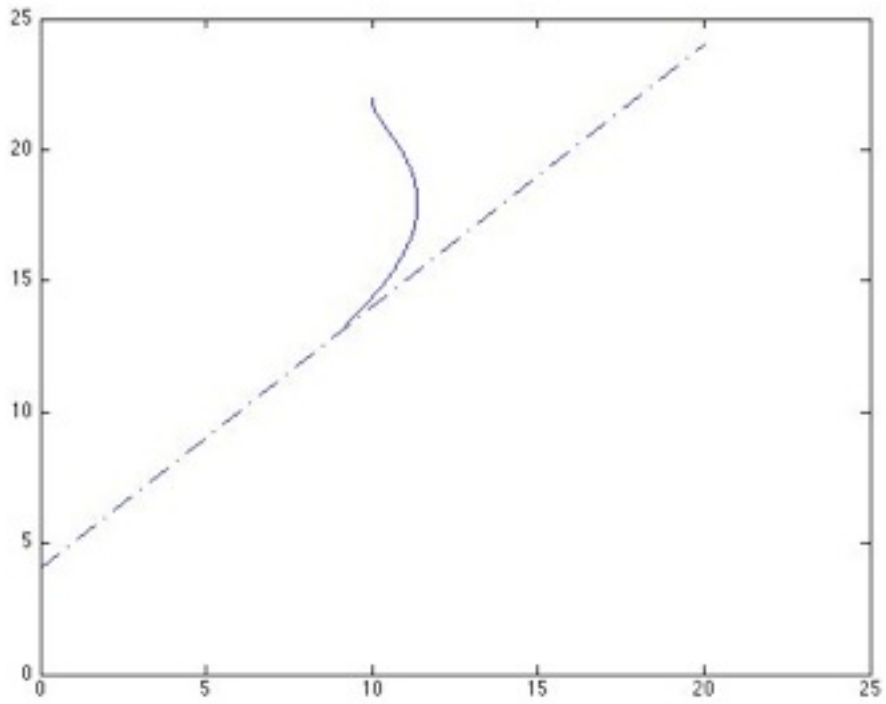
Plot #4:

Gain: 1, Kh: 1, x0 = [ 10, 22, 3*pi/2 ]




I expected this to head towards the line despite the gain being high because the robot is already heading in the "right" direction. I was wrong. Like the third plot, the robot simply went in circles. The close up plot 4b shows that the robot travels in circles repeatedly.

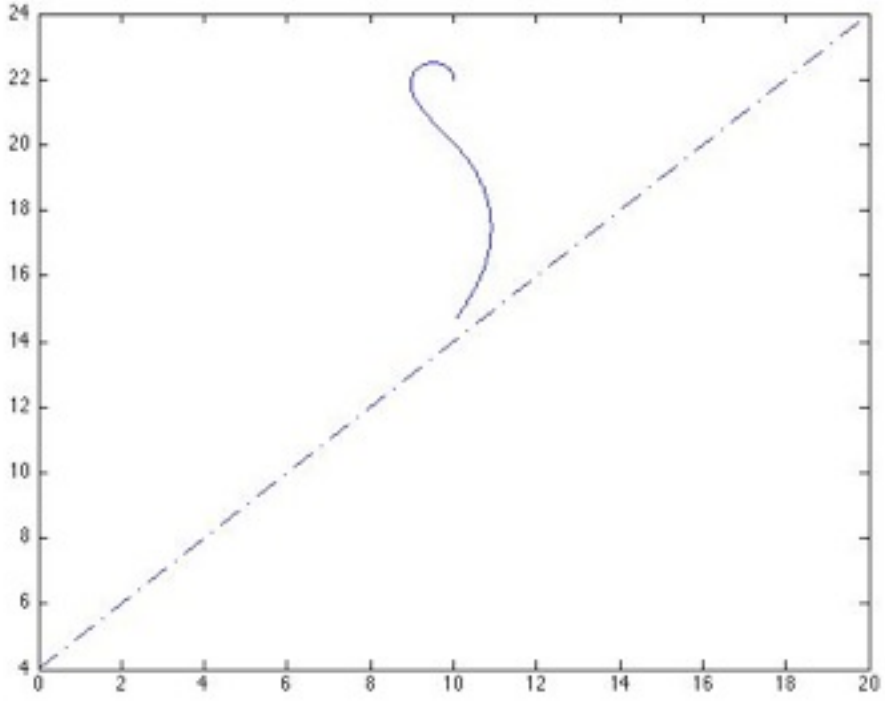
Plot #5: Gain: 0.9, Kh: 1




With a velocity gain just less than one, the robot's path approaches the line more quickly than it did with a lesser gain.

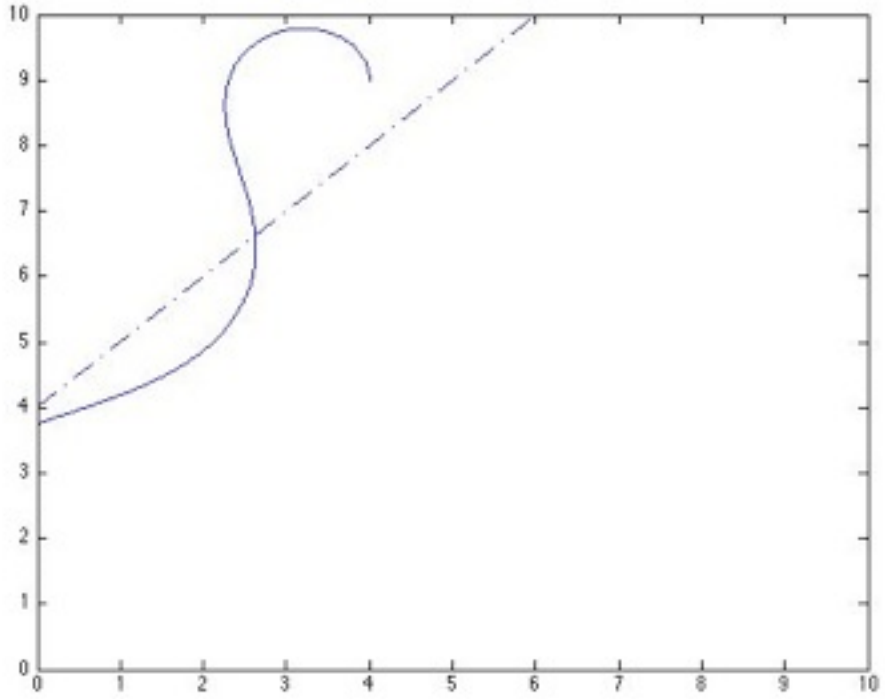
Plot #6:

Gain: 0.9, Kh: 1

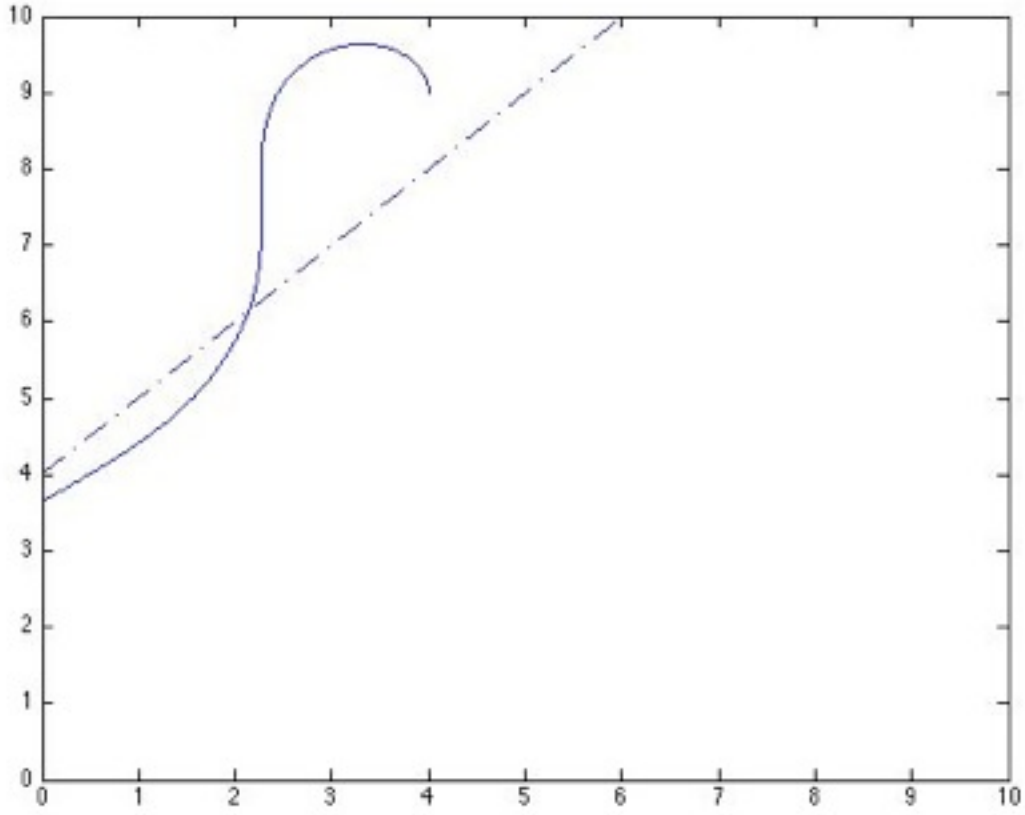


After observing the impact of velocity gain on trajectory, I decided to move onto experimenting with heading gain. Plot #10 and #11 start by facing the robot away from the line. In both cases, increased gain causes the trajectory to acquire the line in less distance. Plot #12-14 start by facing the robot toward the line and demonstrate the same principal.

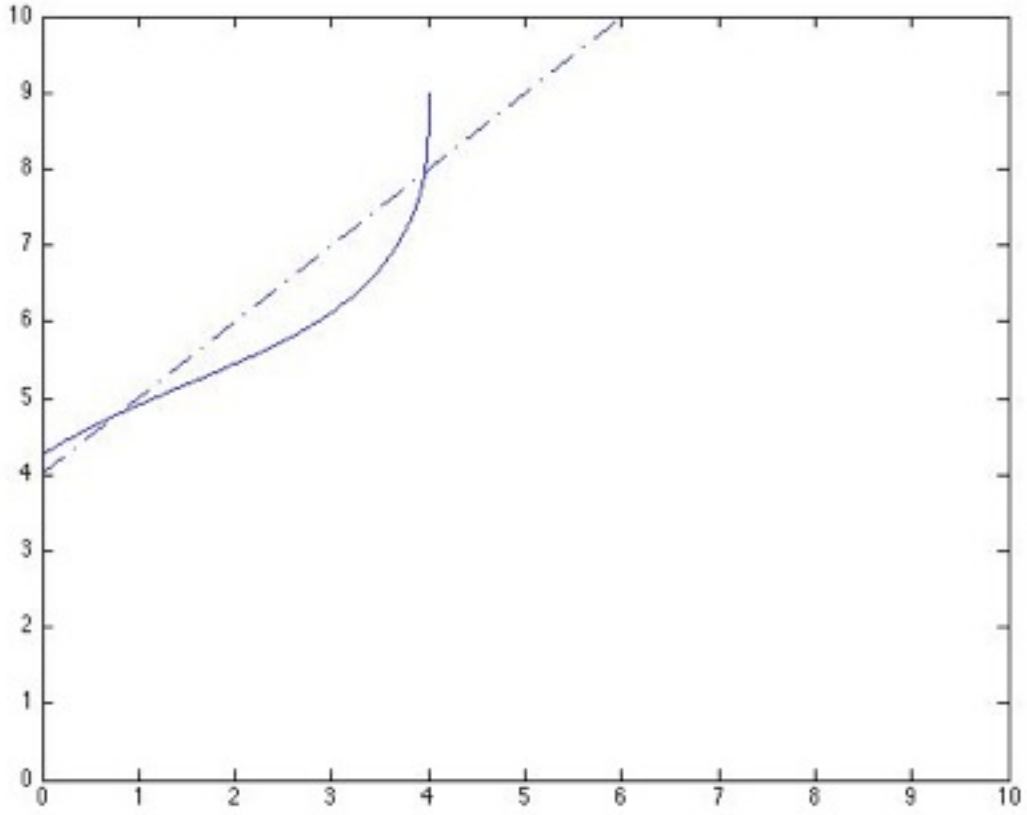
Plot #10: Gain 0.5, Kh: 0.4



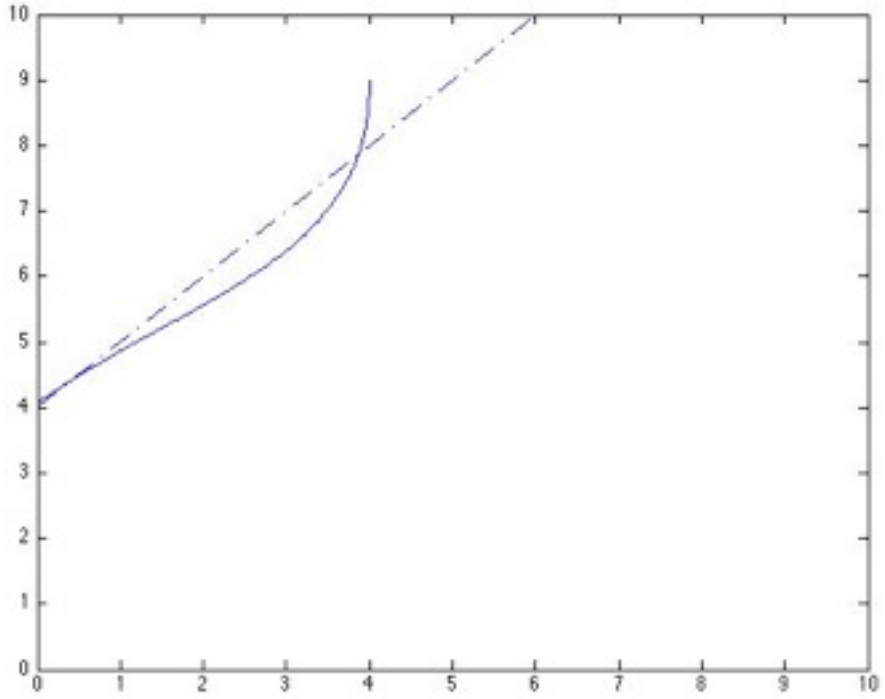
Plot #11: Gain 0.5, Kh: 0.6



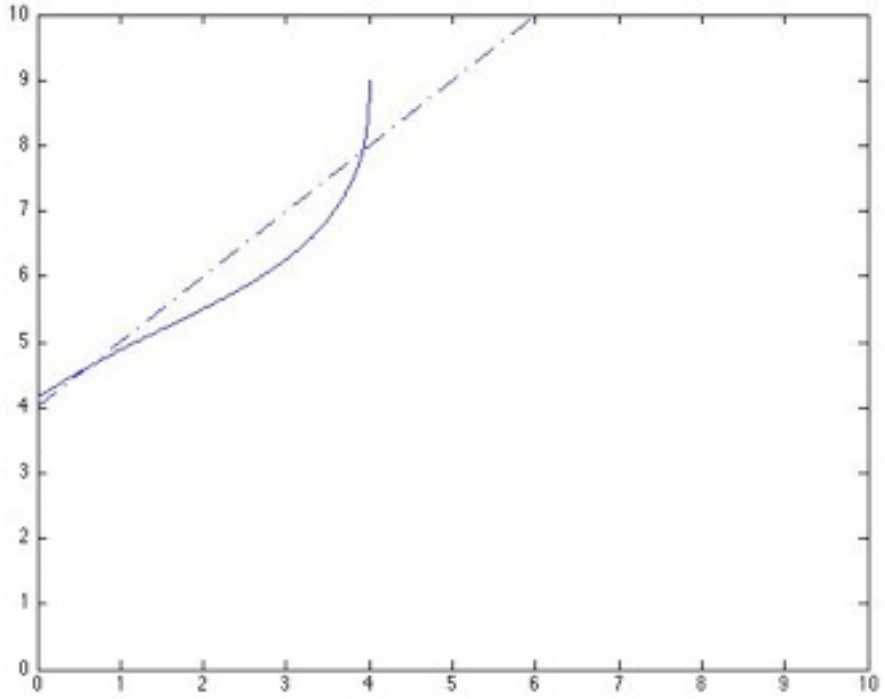
Plot #12: Gain 0.5, Kh: 0.4



Plot #13: Gain 0.5, Kh: 0.6



Plot #14: Gain 0.5, Kh: 0.5



**Problem 4**

a. Experiment with different values. I selected a single point and goal orientation. I know that the first gain parameter, rho is correlated with velocity and that the second and third parameters are correlated with angular velocity (omega).

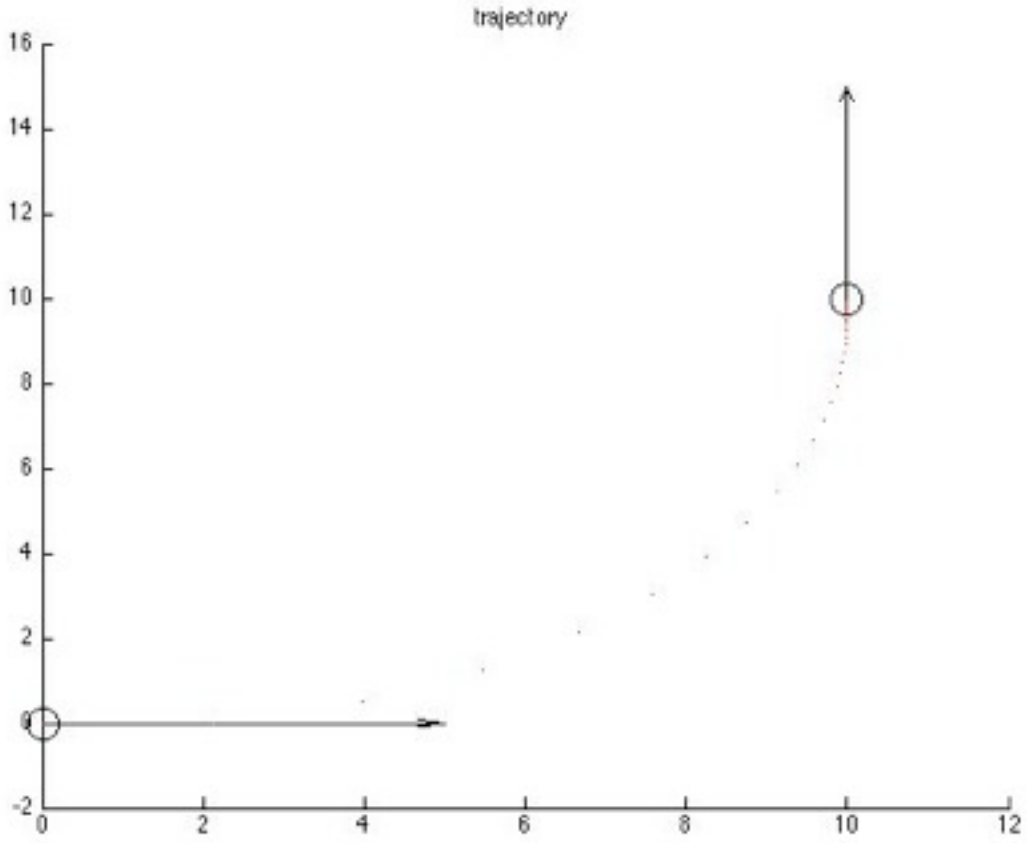
Reducing rho gain should increase the amount of time it takes to the destination. You can see the difference between a0 and a1 have an increase in the number of plots. I also observed that the robot took a more direct trajectory because it has time to start pointing in the correct direction.

Alpha and Beta gain have an influence over the angular velocity.

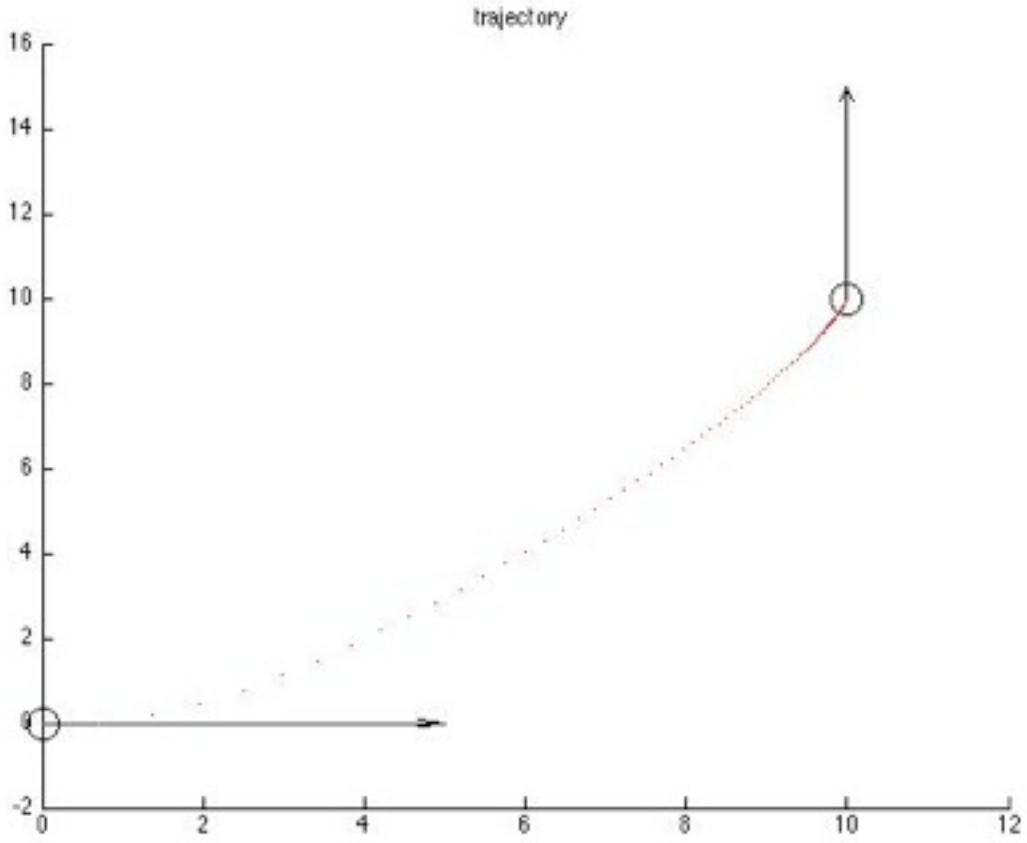
Alpha is how directly the robot steers towards the goal. Beta is how quickly the robot will attempt to orient itself to the final position. In general, it is important that the beta gain does not overtake the alpha gain in such a way that prevents the robot from reaching the goal.

I observed that the same ratio between the Alpha and Beta appear to have the same trajectory for different values of alpha and beta. For example, -12/12 and -25/25 have the same trajectories. This can be seen in a4.jpg, a5.jpg, and a6.jpg

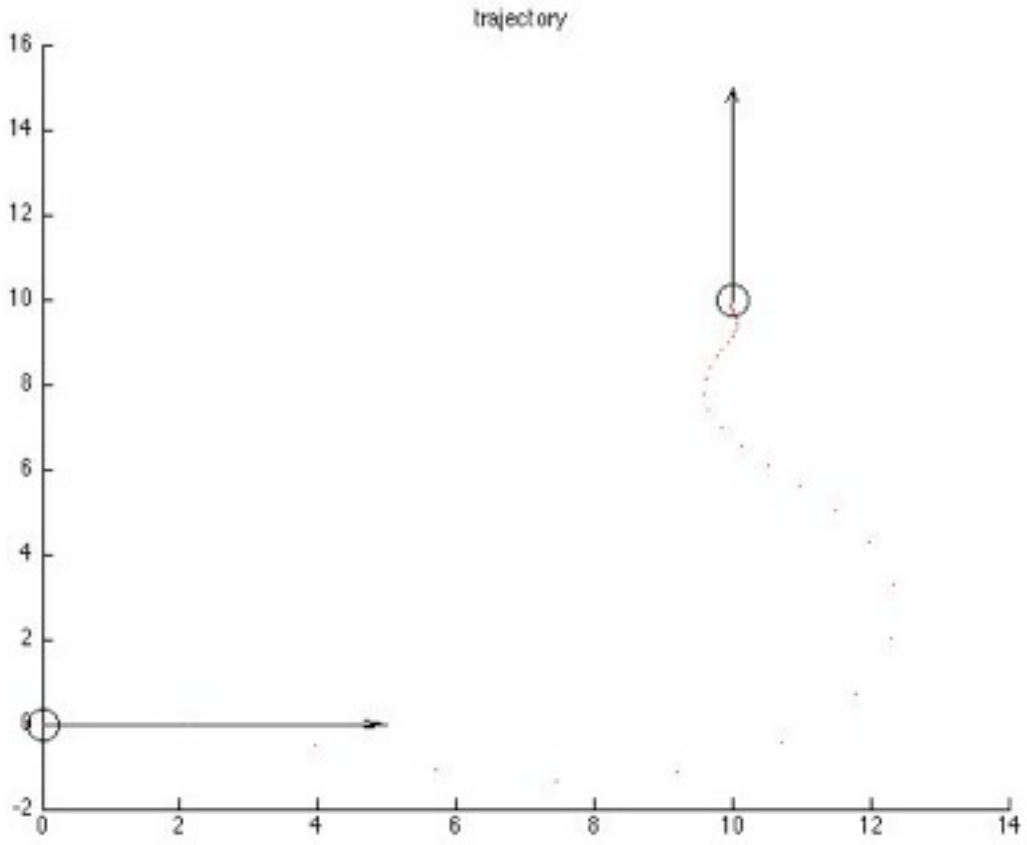
```
% a0

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [3, -1.5, 8] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



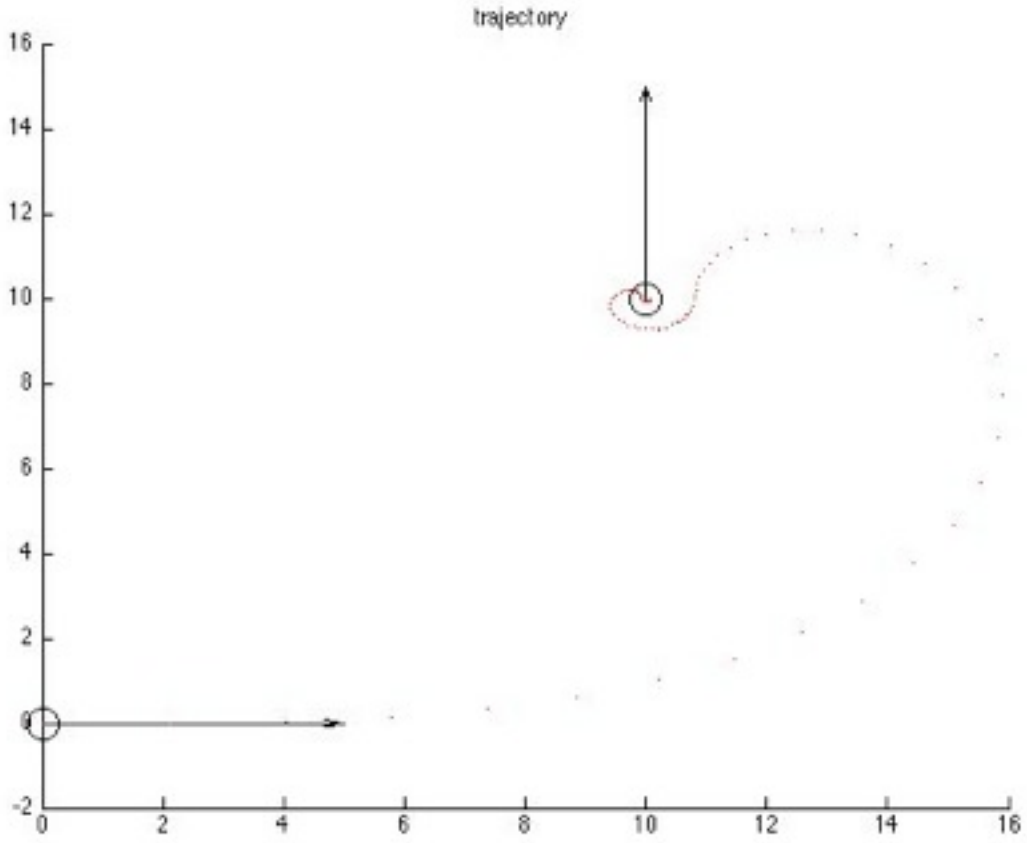
```
% a1

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [1, -1.5, 8] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



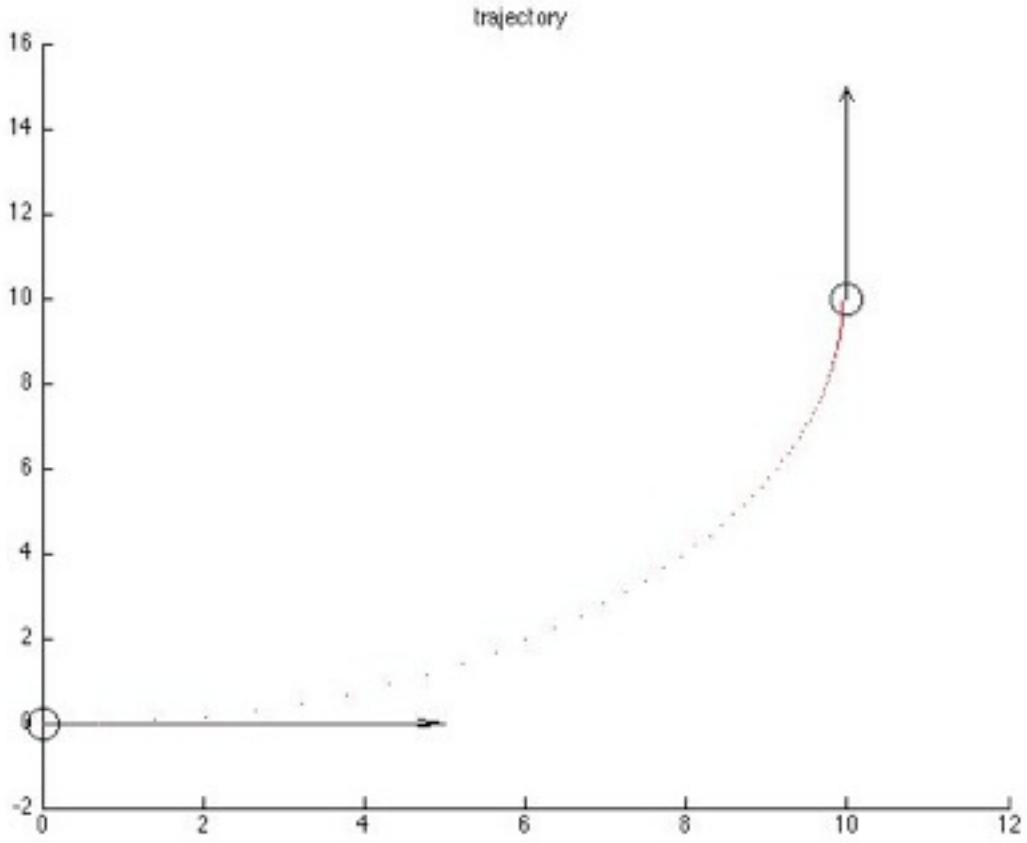
```
% a2

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [3, -15, 8] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



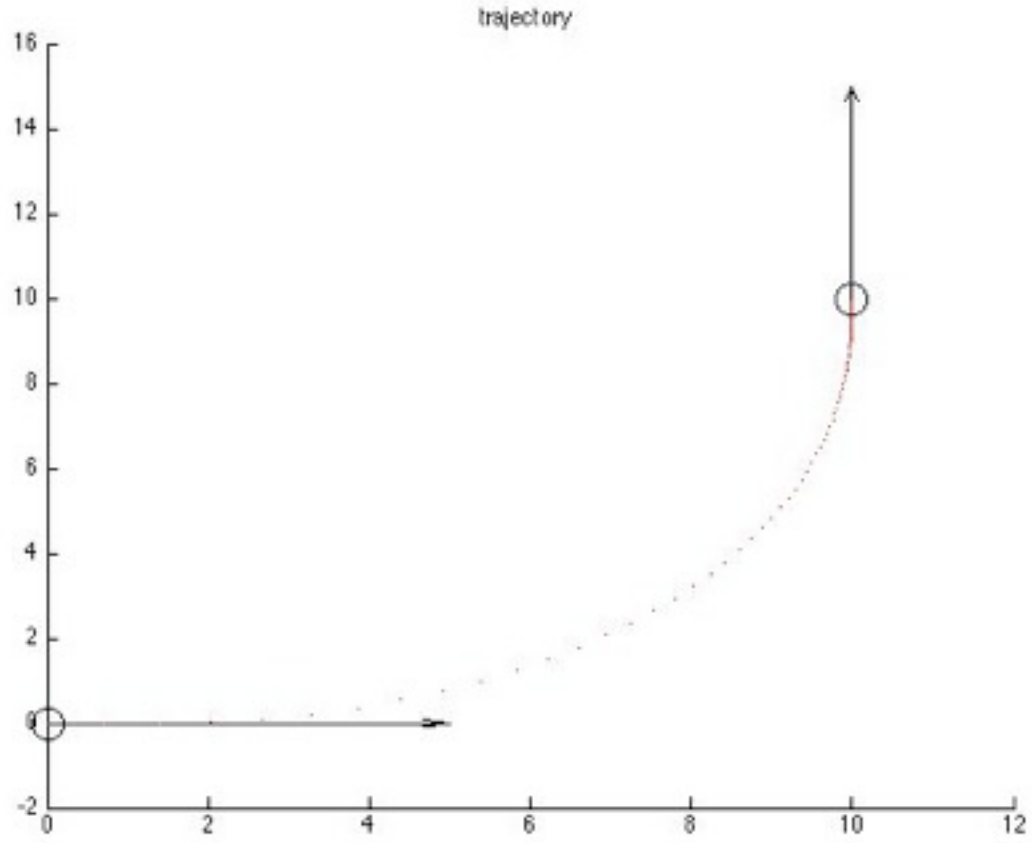
```
% a3

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [3, -1.5, 2] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



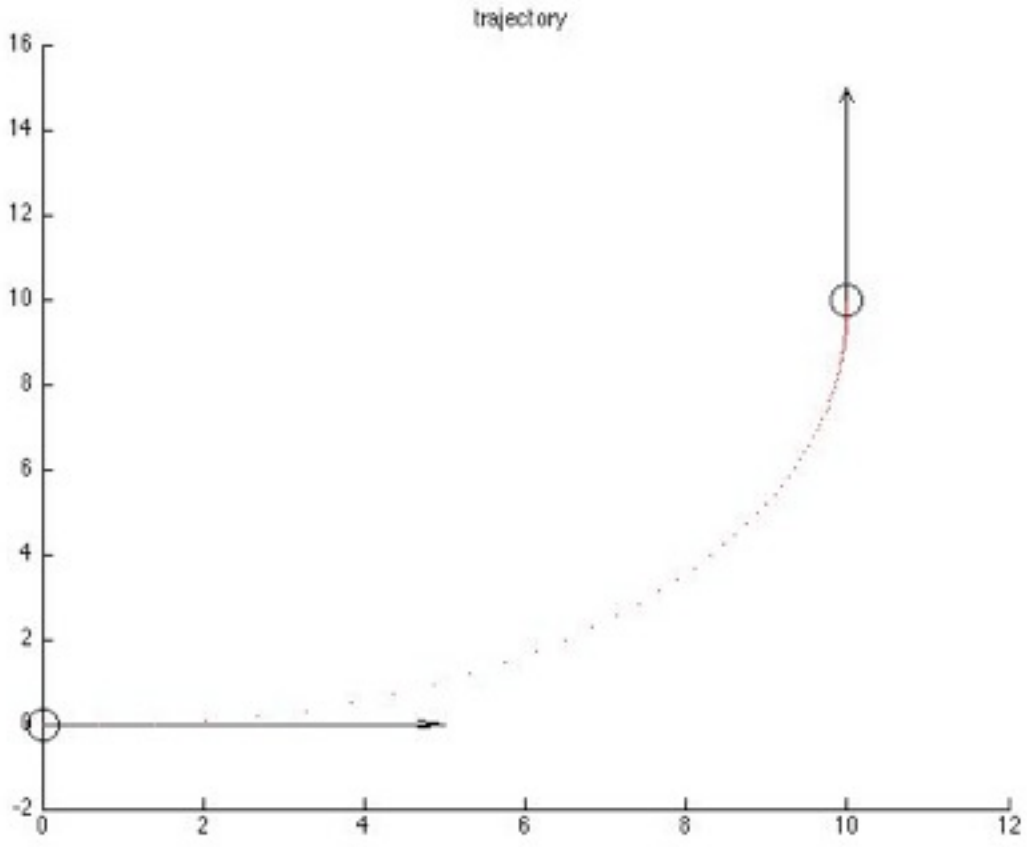
```
% a4

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [1, 1, 1] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



```
% a5

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [1, -12, 12] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



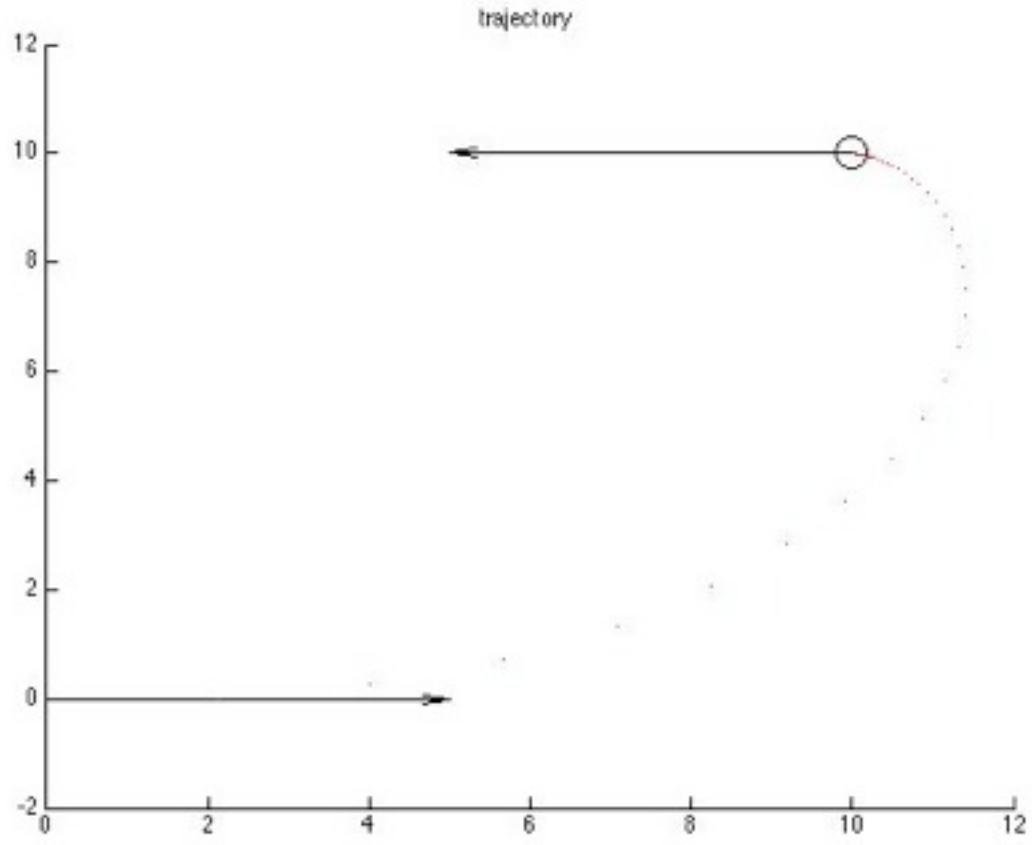
```
% a6 -- trajectory is the same as a5

figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)], [1, -25, 25] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



4b.

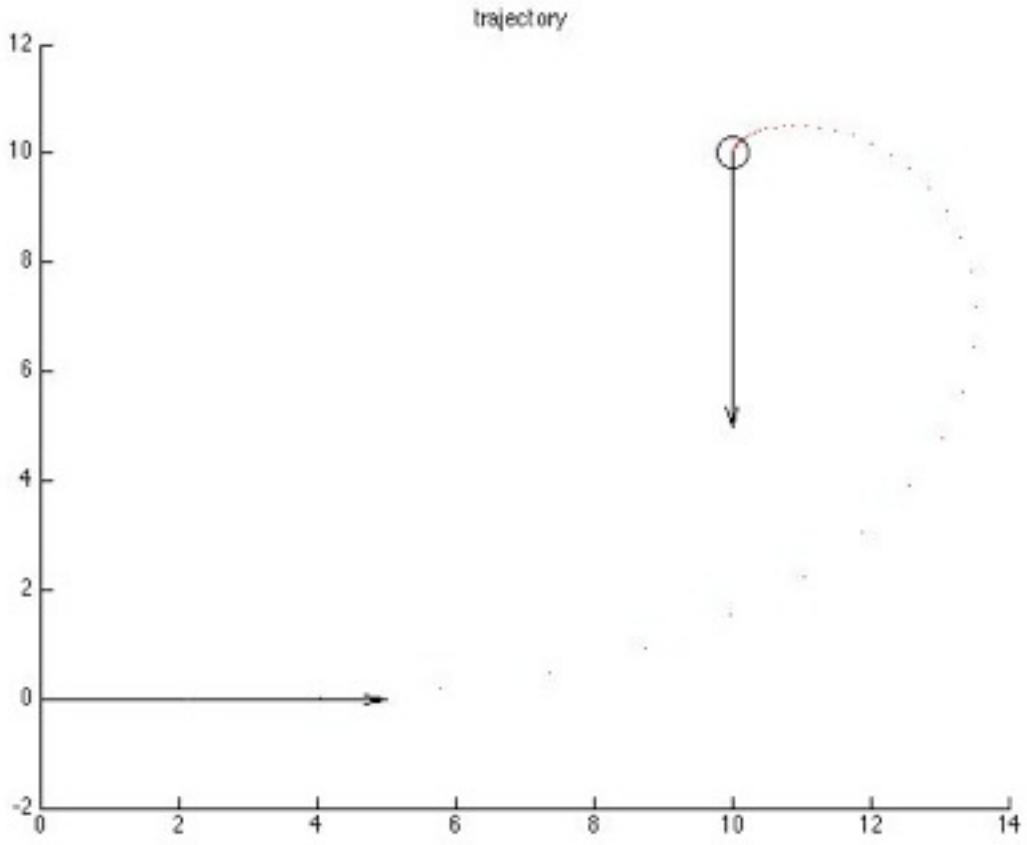
Pose #1.  Turn around 180 degrees.

```
figure; hold on;
[x,y,theta] = goTo([10,10,pi]);
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi);
```



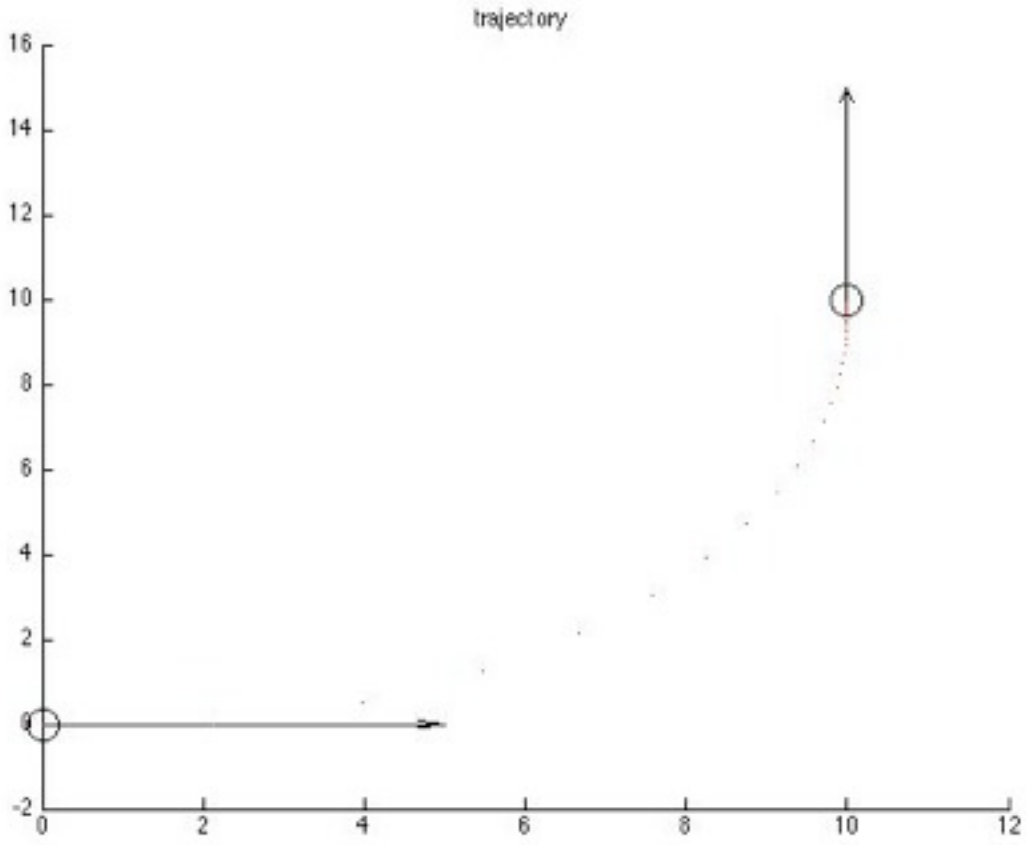
Pose #2.  Turn around 180 degrees.

```
figure; hold on;
[x,y,theta] = goTo([10,10,pi*(3/2)]);
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(3/2));
```



Pose #3.  Turn around 180 degrees.

```
figure; hold on;
[x,y,theta] = goTo([10,10,pi*(1/2)]);
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(10,10,pi*(1/2));
```



Problem 4, part c.

In order to parallel park, the robot will essentially move in a way that looks like an s-curve that is flattened out and stretched. In principal, this means that the robot will achieve the final pose very close to the goal. I had difficulty finding a good configuration for alpha and beta that could parallel park in reverse. However, I think going in reverse is done primarily to maximize control given the wheel configuration of a four wheel vehicle.

c1.jpg

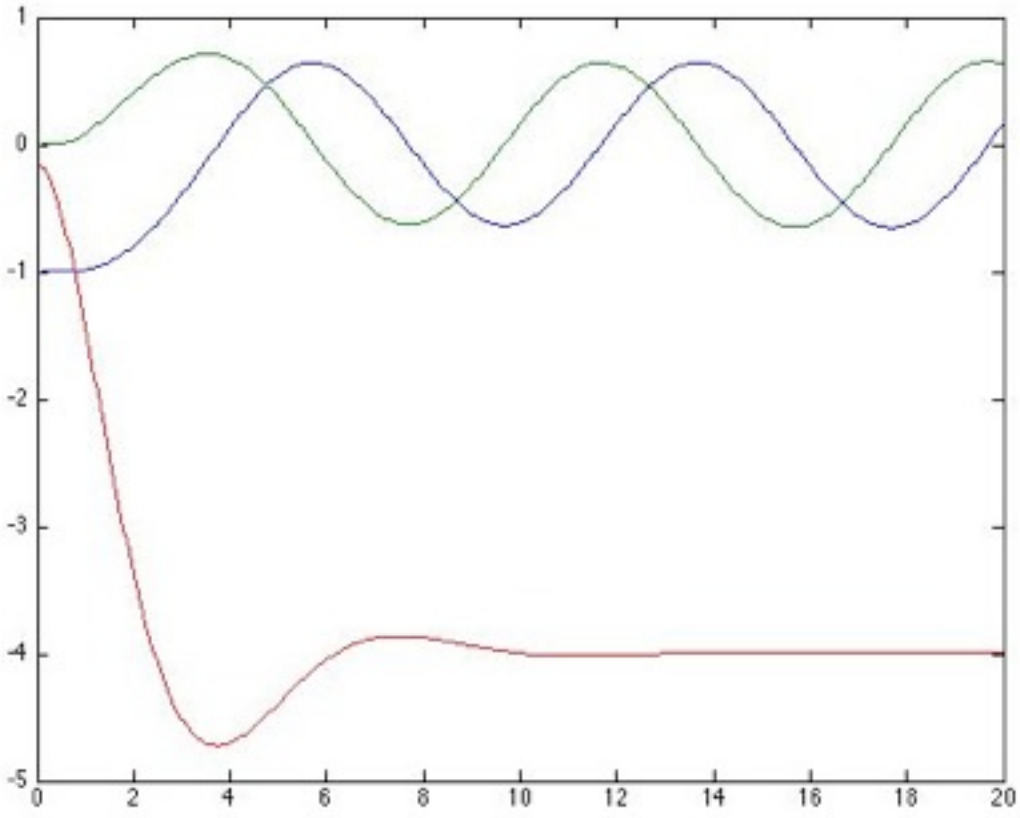
```
figure; hold on;
[x,y,theta] = goTo([6,-1.5,0], [4.5, -5, 8] );
plot_robot(0,0,0);
plot(x,y,'r.'); title('trajectory');
plot_robot(6,-1.5,0);
axis([-1 14 -5 5])
```

**Problem 5**

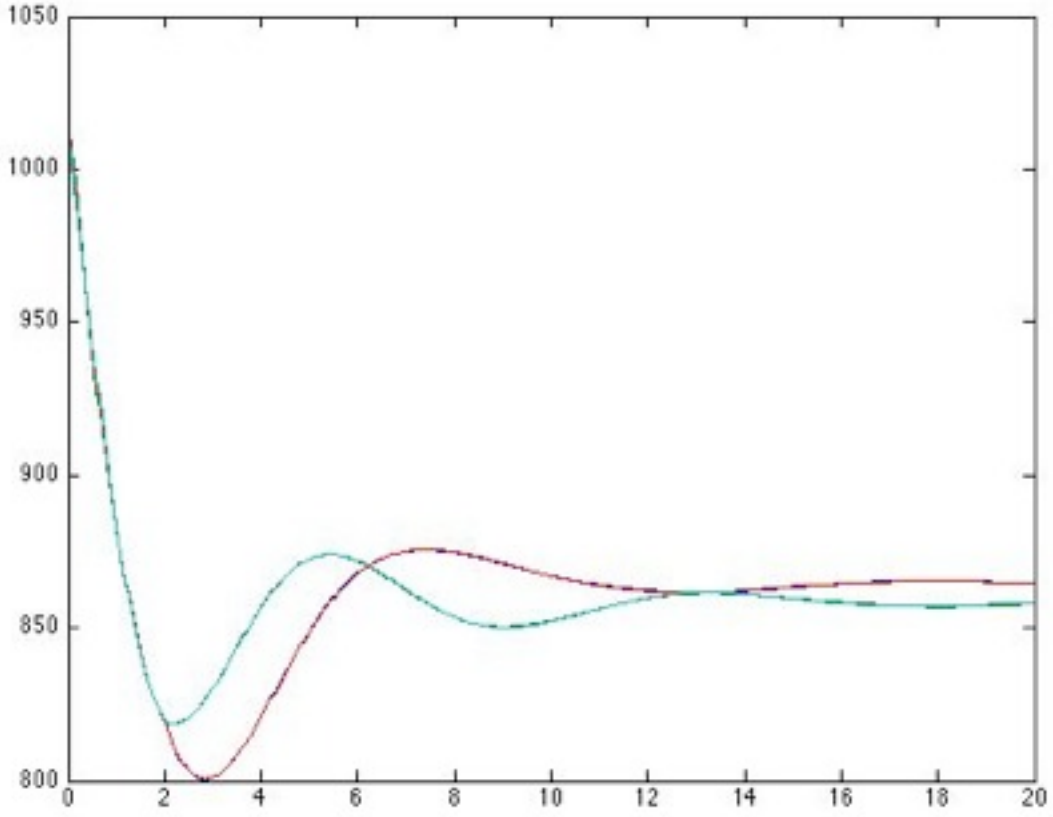
I spent some time playing with the quadcopter simulation.

I noticed that the function mdl_quadcopter created a number of variables in the workspace. For me, the primary structure to experiment with looks like it is called 'quad'. Within quad are a number of variables that appear to control the simulation. I was not able to easily locate the way to control the rotor speeds for the simulation. I imagine this is defined somewhat deeply inside the toolkit.

However, I could plot the resulting simulation's X/Y/Z commands and the rotor speeds. For some reason, the Z appears to be a negative number, though it seems consistent otherwise with how the model appeared during the simulation.



Notice that although there are four rotors, they appear to act in two pairs. This makes a lot of sense given what little I know about quadcopters. By varying opposing rotors, this creates roll by creating lift in a particular lateral direction.




Also, the rotors start at a high rate of rotation to produce lift. Apparently ~870 rotations per minute is sufficient to hover the given mass of the model.