Submission for Final Project:

1) The goal of Engineering Problem #6 is to create plots of velocity, and acceleration for the robot's path, use polynomial curve fitting techniques to create a smoother picture of how the robot travels, and find the velocity and acceleration values given the robot travels in a circular path. All these are asked of us in the question, and we are given the function for velocity and acceleration and also, we are given a robpos.mat file which contains a matrix with 61 rows and 3 columns, with positions of the robot measured every 1 second during an interval of 1 minute. Mathematically, I know that if I am given position vs time, I can derive that function to achieve the velocity vs time function. Furthermore, I can derive the velocity vs time function to achieve the acceleration function. Therefore, I know that I must make use of MatLab's "diff" function when I am finding the plot of velocity and acceleration with respect to x and y for the circular path.
2) Since this is a long problem, it is important to approach it in a step by step manner. First, I am given the robpos.mat file which I need to import into my MatLab script. I see that the robpos file has 61 rows and 3 columns; the columns representing time in seconds and the position of x and y in meters. First, we would need to define the time (first column of robpos file), x position (second column of robpos file) and y position (third column of robpos file). Then we can plot time vs x position and time vs y position, thus getting two graphs. We must be sure to label the plots on the x and y axes and to also include a title. Then, we can create a plot for x position vs y position because the question asks us to do so.
   Now that we have finished answering the first part of part a) of the problem, we can move onto the second part of part a) – the velocity and acceleration. Now, the formula for the velocity and acceleration are given to us; velocity has an elaborate function with indexes for x and y. For example, there is x(t+1) and y(t+1). Since we cannot write this in MatLab straight away, we need another way to represent this. First, we need to use the "hold off" function to create a new plot and to ensure that the previous data is on a separate plot. Then we use "figure" and "hold on" to ensure that we can start a new plot. We are restricted to the domain of 0 to 59 seconds for time. Thus the command for time in this case should be "time_pts=robpos(1:60,1)", where time_pts is the variable, 1:60 represents the domain and the 1 after the 60 represents that the time is located in column 1 of robpos. For x(t), we write "x_pts=robpos(1:60,2)", where x_pts is x(t), 1:60 represents the time and the 2 after the 60 represents that the x position is located in column 2 of robpos. Thus, x_pts2=robpos(2:61,2), where x_pts2 is x(t+1), 2:61 represents 1 to 60 for time and the 2 after the 61 represents that the x position is located in column 2 of robpos. Hence, we can write the y position y(t) and y(t+1) as well using the same logic. Using these indices, we can write the velocity function that was given to us in the question. We would then plot time_pts against velocity to achieve the plot of the velocity function. Now, we need to plot acceleration vs time. This can be done by creating x(t+1), x(t), and x(t-1) using the same logic used before. However, this in case, x(t-1) would have the lowest value for its domain, spanning 0 to 58, while x(t) would be 1 to 59 and x(t+1) would be 2 to 60. We must not exceed 60 seconds as time is not defined past

60 seconds. We also have the acceleration function given to us (that being v(t)-v(t-1)), thus we can sub v(t-1) into the v(t) equation to obtain the function for v(t-1). Using that, we can plot acceleration vs time.

Now, it is time to move onto part b). For the polyfit technique, I recalled it very well because I remembered learning it in MatLab 4. Essentially, for velocity for say, we would plot the plot again and then write coefs=polyfit(time,velocity,order). The order would be determined based on how the curve of best fit matches with the velocity function. We would need to write y=polyval(coefs,x), with x already being defined with linspace earlier. Then we could plot(x,y) and then play around with the order of the function to match something which we could deem as satisfactory enough to represent the velocity function. Similarly, we would repeat this process except with the acceleration function to determine the order of that function.

For part c), we are given information about a circle. The circle has diameter of 2.4 meters which means that it has a radius of 1.2 meters. We can assign a variable, say k, that spans from 0 seconds to 60 seconds (which is required in the question). We need to make equations for the circle for x and y. Knowing that 360 degrees divided by 60 seconds is 6, we know that for the x function, 6 degrees or pi/60 would be needed inside the cosine function (as cosine correlates to x). The entire function: x_circ=cos(k*pi/30) needs to be multiplied by the radius (1.2 meters). After obtaining the code for the x circle, we can similarly deduce the y circle to be: y_circ=1.2cos(k*pi/30)+1.2. We can plot the circle by plot(x_circ,y_circ). To continue, we need to find the velocity and acceleration for x and y respectively. We can now make use of the diff function on MatLab. For velocity, we can find the diff of k and the diff of x_circ and write find diff(x_circ)./diff(k). Then we can plot k against diff(x_circ)./diff(k) to graph the velocity function. We do the similar method for y. The acceleration function is similar. It is the second derivative of the position function. So it would be k plotted against diff(diff(x_circ))./diff(diff(k)). We would similarly make one for the acceleration for y as well. In total, we will have two plots for velocity (one for x and y) and two plots for acceleration (one for x and y) for the circular path that the robot travels.

3) **Code:**

```
%Loading robpos file
robpos
%Naming the given robpos data
time_pts = robpos(1:61,1)
x_pts = robpos(1:61,2)
y_pts = robpos(1:61,3)
%Creating the plots
grid on
hold on
plot(time_pts,x_pts)
plot(time_pts,y_pts)
legend ('x vs t','y vs t')
title ('Position of the Robot through the 2D space')
xlabel ('Time')
ylabel ('Position')
hold off
```

```
figure
hold on
grid on
plot(x_pts,y_pts)
xlabel('X Position')
ylabel('Y Position')
title('X Position vs Y Position')
%Plotting Velocity Function
hold off
figure
hold on
grid on
time_pts = robpos(1:60,1);
x_pts = robpos(1:60,2);
x_pts2 = robpos(2:61,2);
y_pts = robpos(1:60,3);
y_pts2 = robpos(2:61,3);
v=sqrt((((x_pts2)-(x_pts)).^2)+(((y_pts2)-(y_pts)).^2))
plot(time_pts,v)
title('Velocity Function')
ylabel('Velocity')
xlabel('Time')
%Plotting Acceleration
hold off
figure
hold on
grid on
time_pts = robpos(2:60,1);
x_pts = robpos(2:60,2);
x_pts2 = robpos(3:61,2);
y_pts = robpos(2:60,3);
y_pts2 = robpos(3:61,3);
x_pts3 = robpos(1:59,2);
y_pts3 = robpos(1:59,3);
v=sqrt((((x_pts2)-(x_pts)).^2)+(((y_pts2)-(y_pts)).^2));
v2=sqrt((((x_pts)-(x_pts3)).^2)+(((y_pts)-(y_pts3)).^2));
a=v-v2
plot(time_pts,a)
title('Acceleration Function')
ylabel('Acceleration')
xlabel('Time')
%Polyfit for Velocity
hold off
figure
hold all
grid on
x=linspace(0,60,101)
time_pts = robpos(2:60,1);
v=sqrt((((x_pts2)-(x_pts)).^2)+(((y_pts2)-(y_pts)).^2));
plot(time_pts,v,'r')
ylabel('Velocity')
xlabel('Time')
title('Velocity Function')
coefs=polyfit(time_pts,v,9)
y=polyval(coefs,x)
plot(x,y)
legend('Velocity Function','9-th Order Fit')
```

```
%Polyfit for Acceleration
hold off
figure
hold all
grid on
x=linspace(0,60,101)
time_pts = robpos(2:60,1);
x_pts = robpos(2:60,2);
x_pts2 = robpos(3:61,2);
y_pts = robpos(2:60,3);
y_pts2 = robpos(3:61,3);
x_pts3 = robpos(1:59,2);
y_pts3 = robpos(1:59,3);
v=sqrt((((x_pts2)-(x_pts)).^2)+(((y_pts2)-(y_pts)).^2));
v2=sqrt((((x_pts)-(x_pts3)).^2)+(((y_pts)-(y_pts3)).^2));
a=v-v2
plot(time_pts,a,'r')
ylabel('Acceleration')
xlabel('Time')
title('Acceleration Function')
coefs=polyfit(time_pts,a,11)
y=polyval(coefs,x)
plot(x,y)
legend('Acceleration Function','11-th Order Fit')
%Plotting the circle
figure
hold on
grid on
%A for loop can be used here instead of doing this; explanantion of this
%can be found in my description of ways I had to adjust the script
k=0:60;
x_circ=(1.2*cos(k*(pi/30)));
y_circ=1.2*sin(k*(pi/30))+1.2;
plot(x_circ,y_circ);
xlabel('X Position');
ylabel('Y Position');
title('Circular Path');
%Plotting Velocity for X
hold off
figure
hold on
grid on
dy=diff(x_circ)
dx=diff(k)
dydx=dy./dx
xd=k(2:end)
plot(xd,dydx,'b')
xlabel('Time for X')
ylabel('Velocity')
title('Velocity vs Time for X')
%Plotting Velocity for Y
hold off
figure
hold on
grid on
dq=diff(y_circ);
da=diff(k)
```

```
dqda=dq./da
ad=k(2:end)
plot(ad,dqda,'b')
xlabel('Time for Y')
ylabel('Velocity')
title('Velocity vs Time for Y')
%Plotting Acceleration for X
hold off
figure
hold on
grid on
dz=diff(diff(x_circ));
l=2:60;
plot(l,dz,'r')
xlabel('Time for X')
ylabel('Acceleration')
title('Acceleration for X')
%Plotting Acceleration for Y
hold off
figure
hold on
grid on
dm=diff(diff(y_circ));
l=2:60;
plot(l,dm,'r')
xlabel('Time for Y')
ylabel('Acceleration')
title('Acceleration for Y')
```

Plots:

Figure 1 – Position of the Robot through the 2D Space (X vs T and Y vs T):

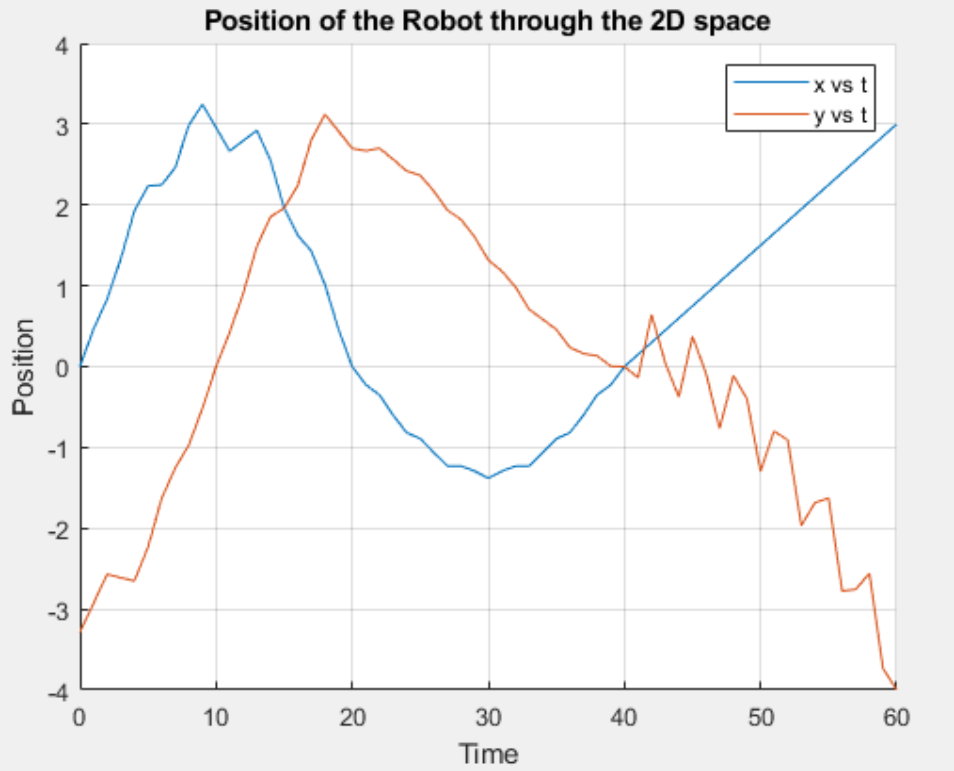


Figure 2 – X Position vs Y Position:

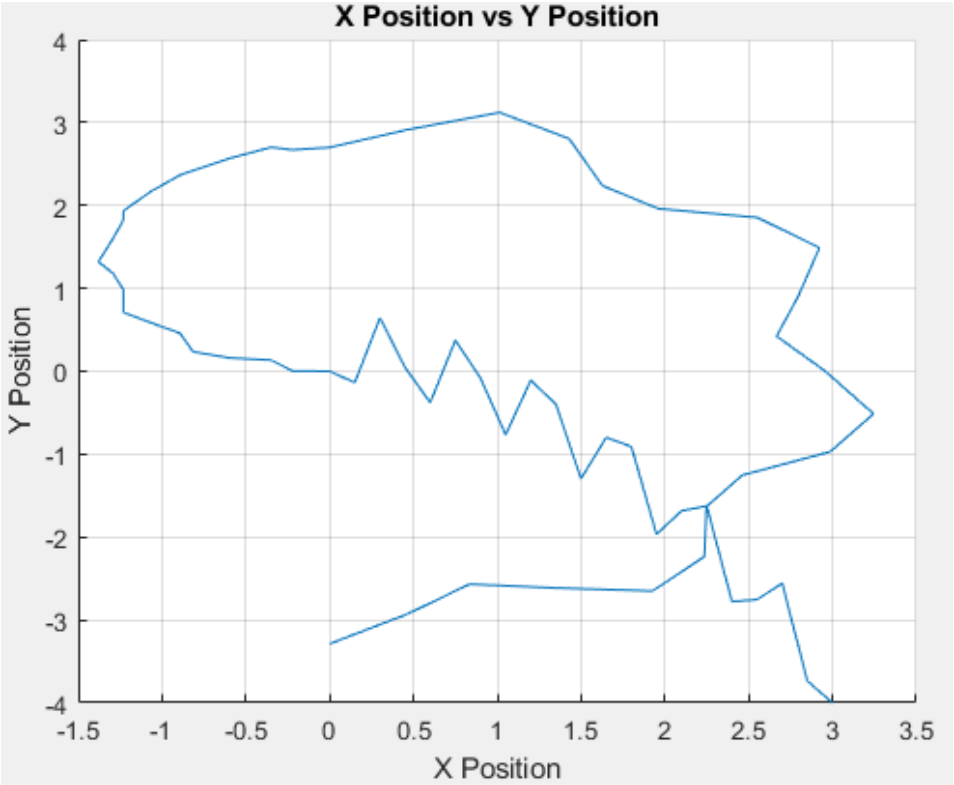


Figure 3 - Velocity vs Time:

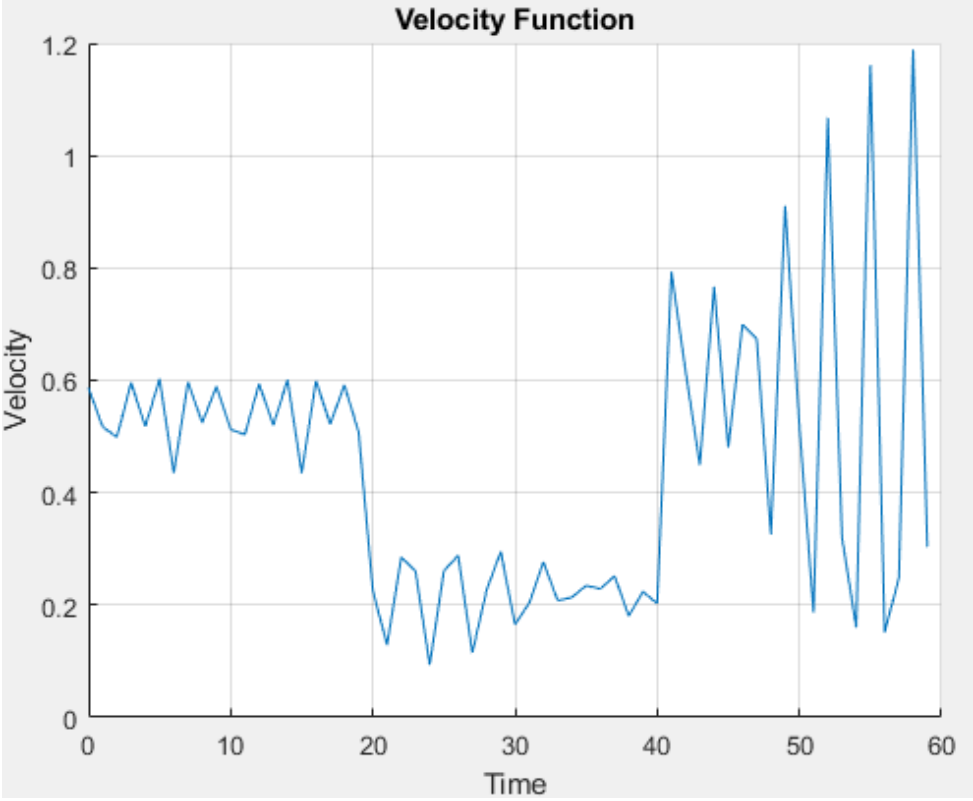


Figure 4 – Acceleration vs Time:

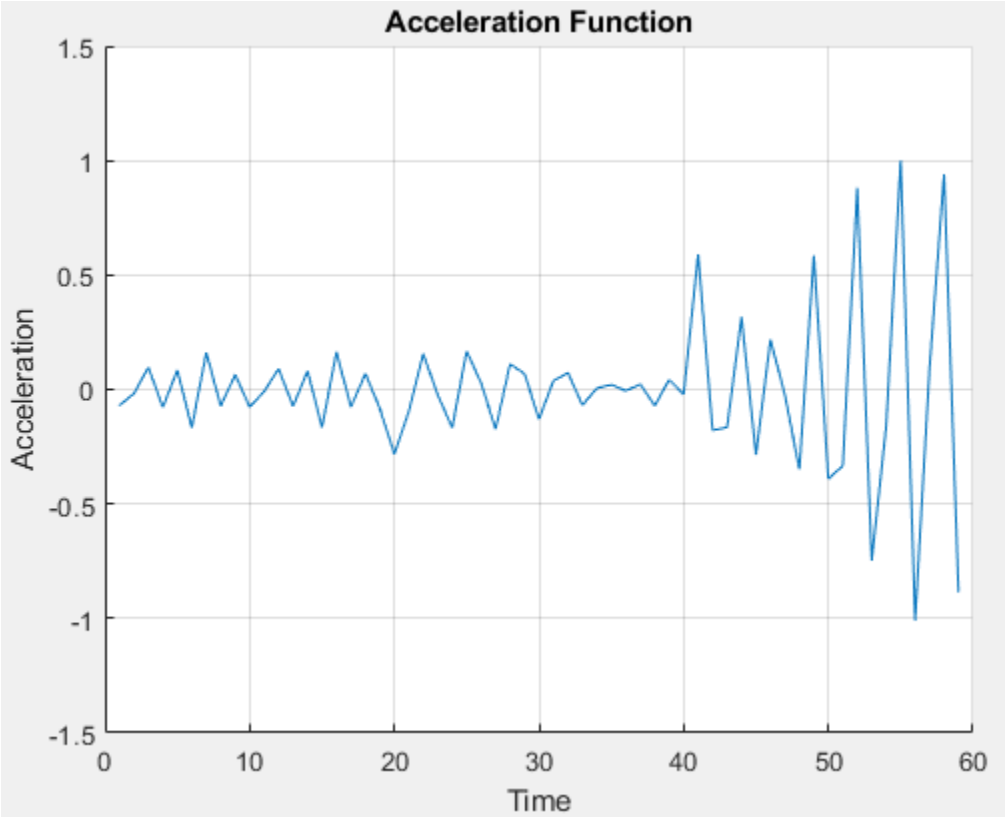


Figure 5 – Velocity with Polynomial Fit (Best Fit was 9th Order):

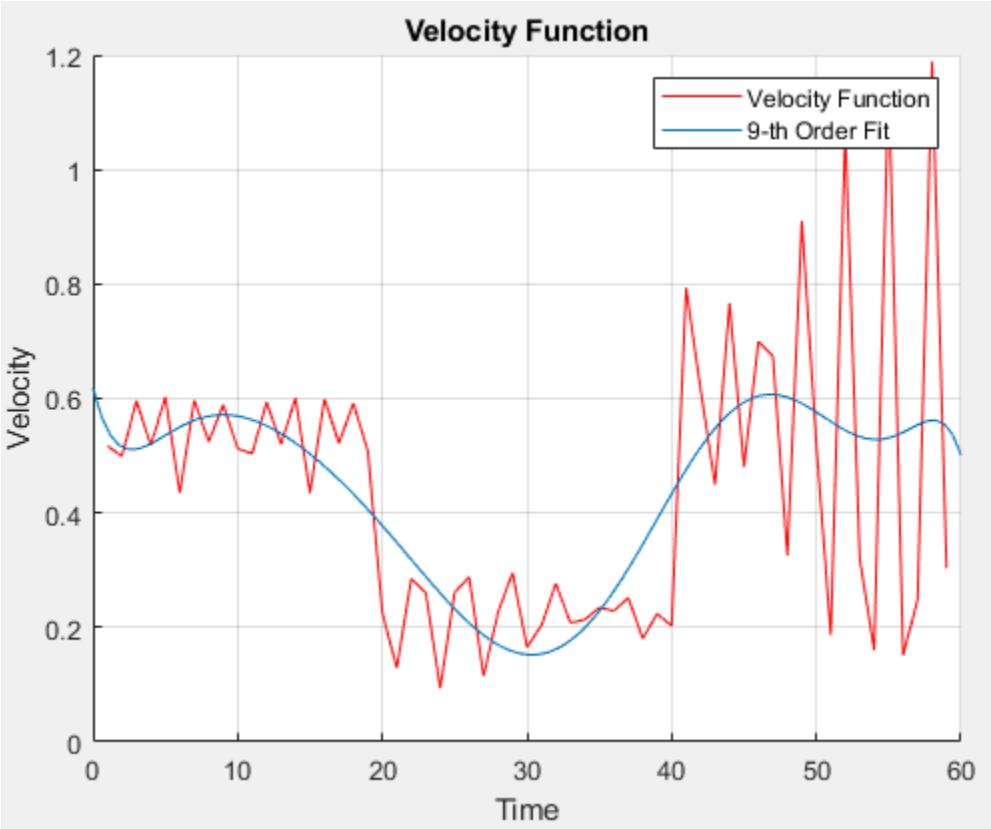


Figure 6 – Acceleration Fuction with Polynomial Fit (Best Fit was 11th Order:

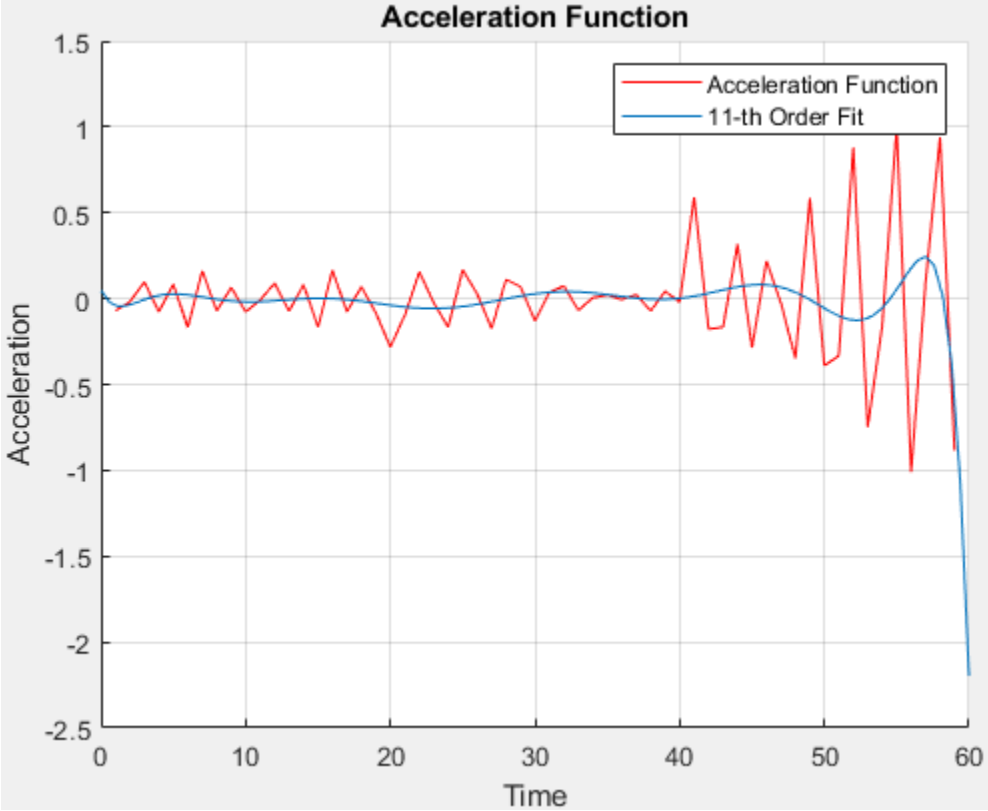


Figure 7 – Circular Path of the Robot:

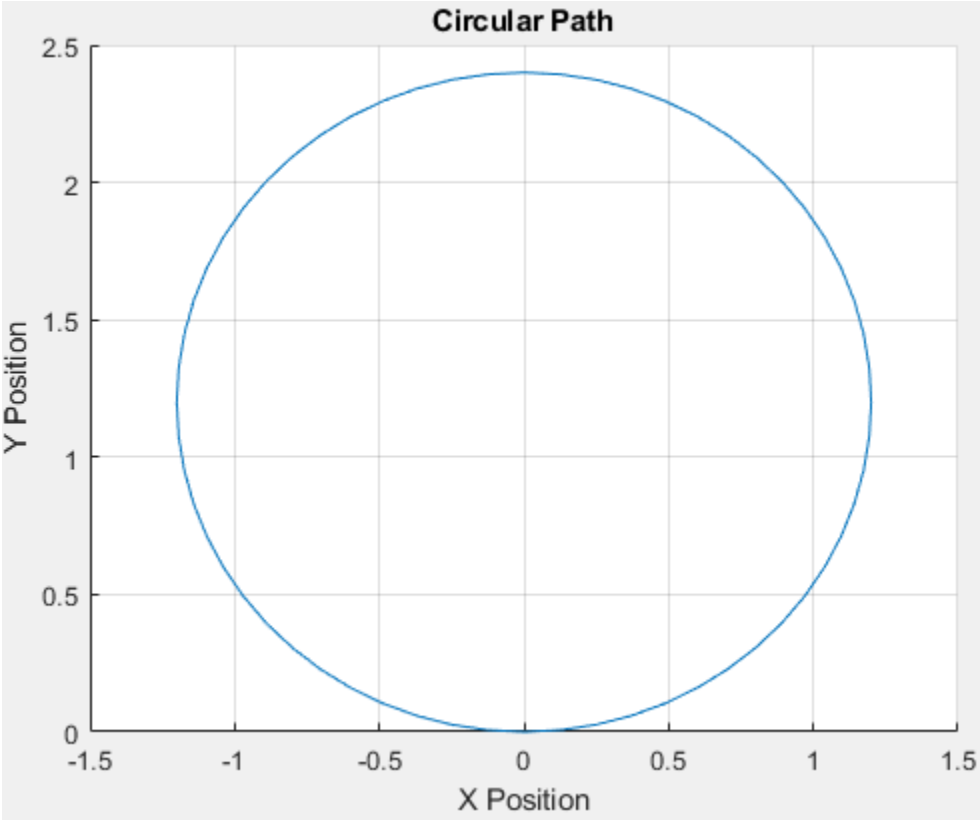


Figure 8 – Velocity vs Time for X Position:

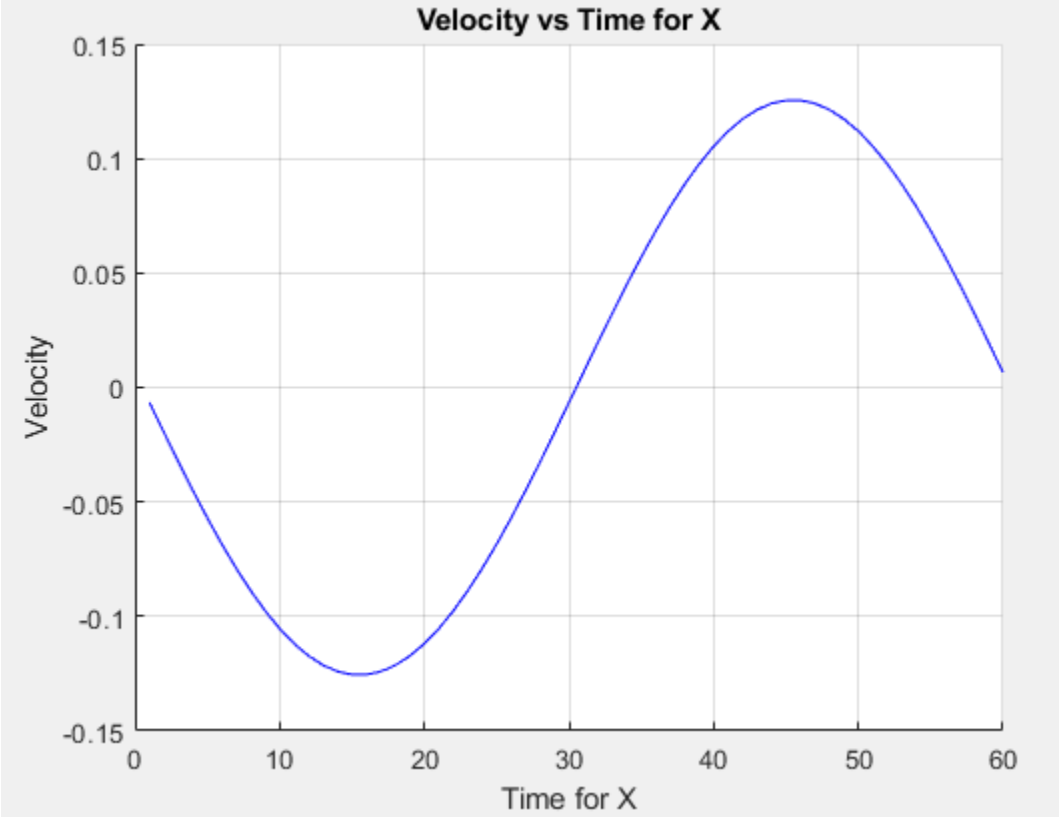


Figure 9 – Velocity vs Time for Y Position:

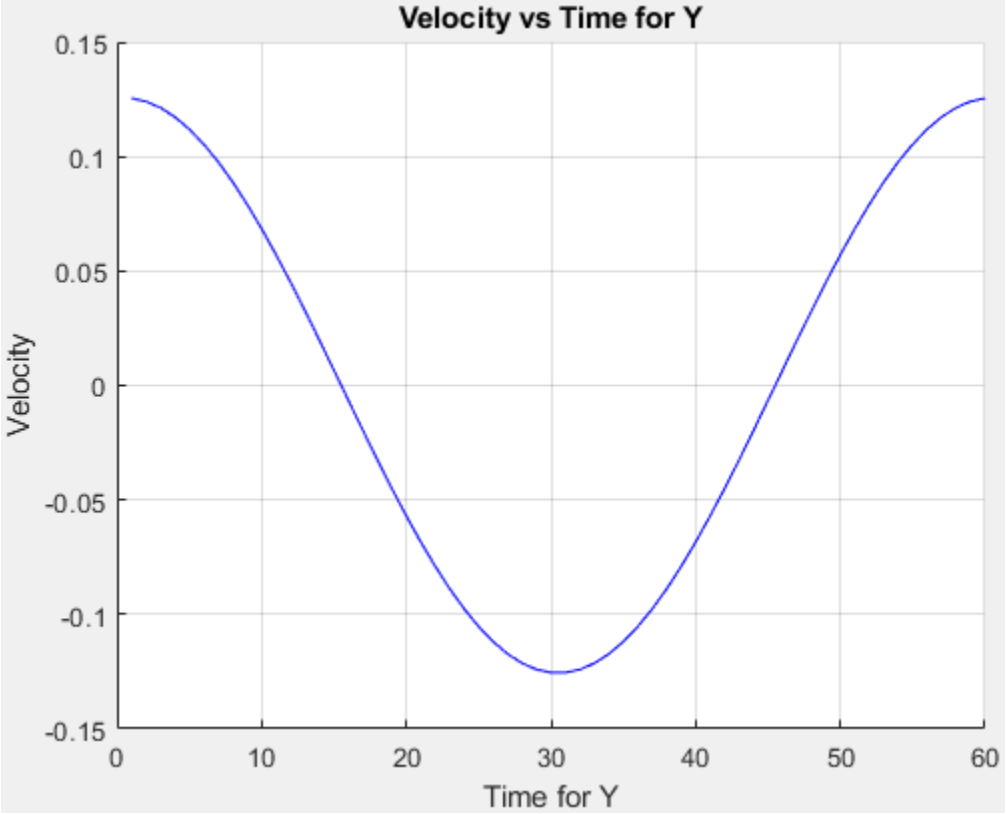


Figure 10 – Acceleration vs Time for X Position:

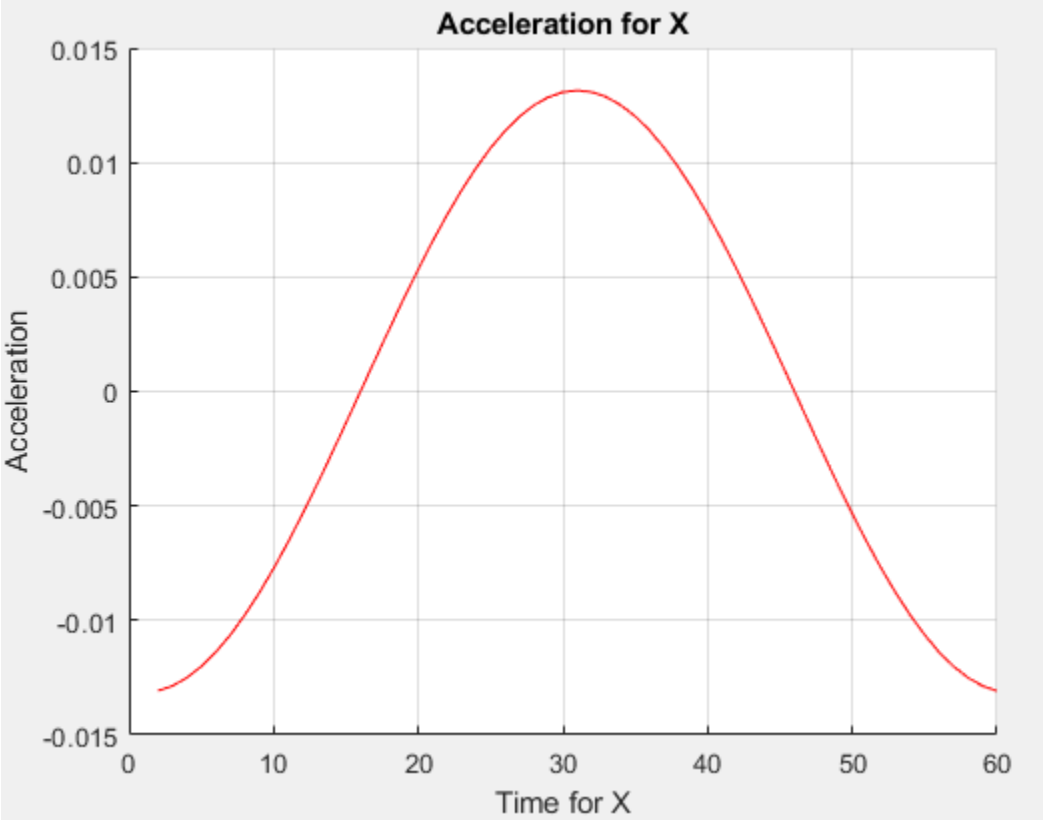
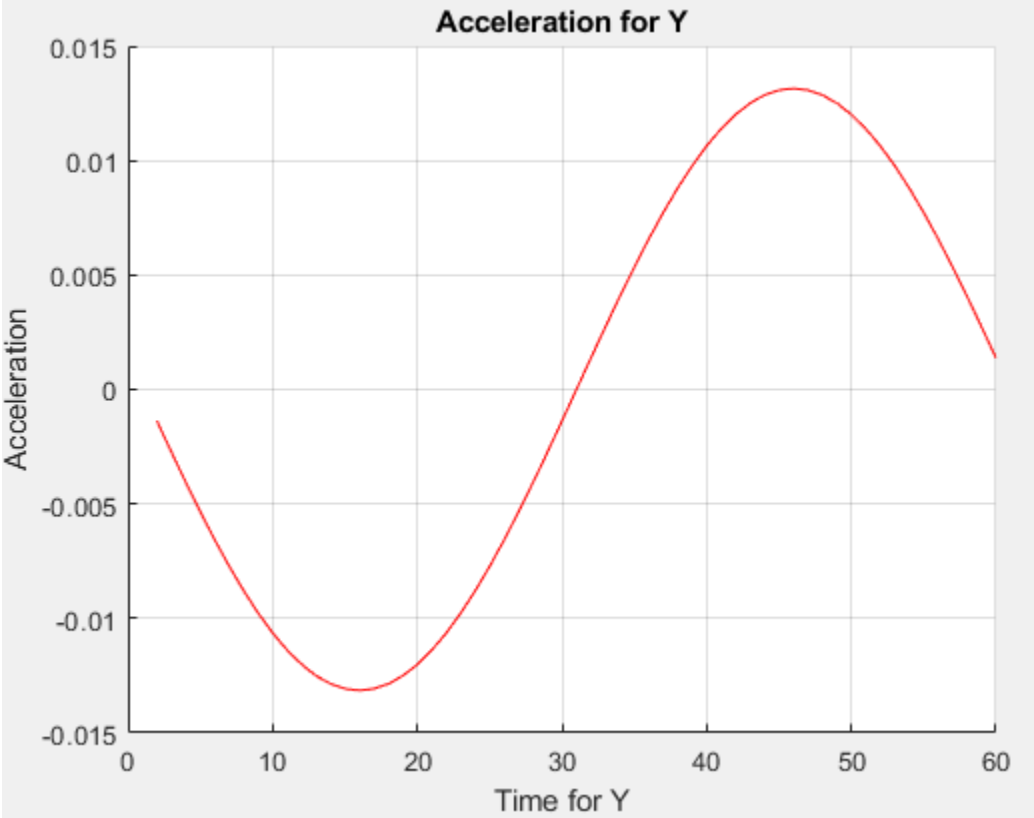


Figure 11 – Acceleration vs Time for Y Position:



Description of the Ways I had to adjust my Script:

Looking back, my plan in Step 2 did not work to perfection. I had to change some things about my initial plan afterwards. For example, I learned that it was required for all MatLab students to incorporate a for loop or if statement in the script. Thus, I decided that I had to include a for loop somewhere in my script, thus changing it up a little. I had the idea to write a for loop for the creation of the circle. I did this by writing:

```
for k=1:61
    xd(t)=1.2.*cos(t.*(pi/30));
    yd(t)=1.2.*sin(t.*(pi/30))+1.2;
end
```

Thus, I could incorporate a for loop in my script.

4) I know that I have adequately addressed all the problems because my code answers the questions that were asked of me in Engineering Problem #6. The code runs perfectly fine without the need to troubleshoot. My answers make sense; for instance, the circular path graph is a circle. Also, the velocity and acceleration functions for x and y respectively in part c) of the problem are sinusoidal and cosine graphs, which make sense given that the equation of the circle for x and y are cosine and sine respectively. The derivatives of the cosine and sine functions would give sine and cosine functions, and the derivative of those would give cosine and sine functions, which is clearly displayed from my plots of velocity and acceleration vs x and y respectively. I have learned about the power and ability of MatLab over the course of this final project. MatLab is not only just a great computational tool (being able to solve large matrices in the matter of seconds), but it also has the ability to import data, run plots, find derivatives and plot the derivatives, and find the curve of best fit – all accurately and quickly. MatLab is a great scientific and mathematical tool that has the potential to help me analyze, create and explore complicated plots and solve complicated computational problems. For the purposes of Engineering Problem #6, I can conclude these findings:
   - For part a), I have plotted the required velocity and acceleration functions against time, and also I have plotted the y position vs x position.\
   - For part b), with the use of polyfit and polyval, I can deduce that the best order function to represent the velocity function is 9^{th} order and the best order function to represent the acceleration function is 11^{th} order.
   - For part c), I have plotted the circular path function. Furthermore, I have plotted the respective velocity and accelerations against x and y. Since the question asks for the velocity and acceleration values in each direction that the robot travels, we can extract the data points from MatLab and examine the tables that they give us.

   Ultimately, MatLab was a great tool in solving this problem, as evident by my ability to solve all three parts (a, b, c) in Engineering Problem #6.