

CSCI 545: Homework 1

Deepika Anand

February 27, 2017

Problem 1. (a)

$$\dot{\mathbf{x}} = \frac{\alpha}{\tau} * (x_f - x) \quad (1)$$

Taking laplace transformation on both sides we get

$$sx(s) = \frac{\alpha}{\tau} * (x_f - x(s)) \quad (2)$$

Taking $x(s)$ terms on one side

$$x(s) * (s + \frac{\alpha}{\tau}) = \frac{\alpha}{\tau} x_f \quad (3)$$

Simplifying further to get

$$x(s) = \frac{\tau * \alpha}{s * \tau + \alpha} * \frac{1}{\tau} x_f \quad (4)$$

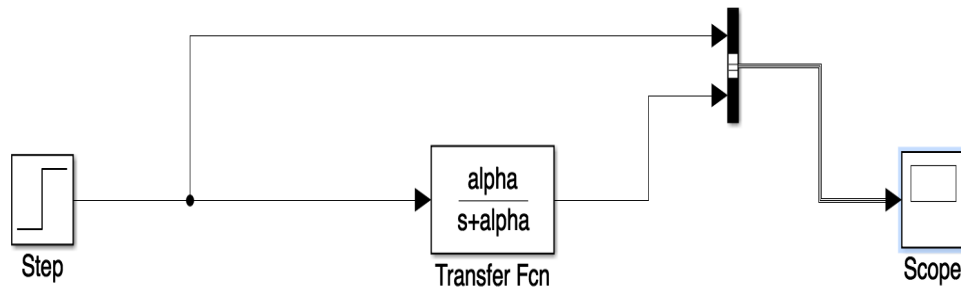
$$x(s) = \frac{\alpha}{s * \tau + \alpha} * x_f \quad (5)$$

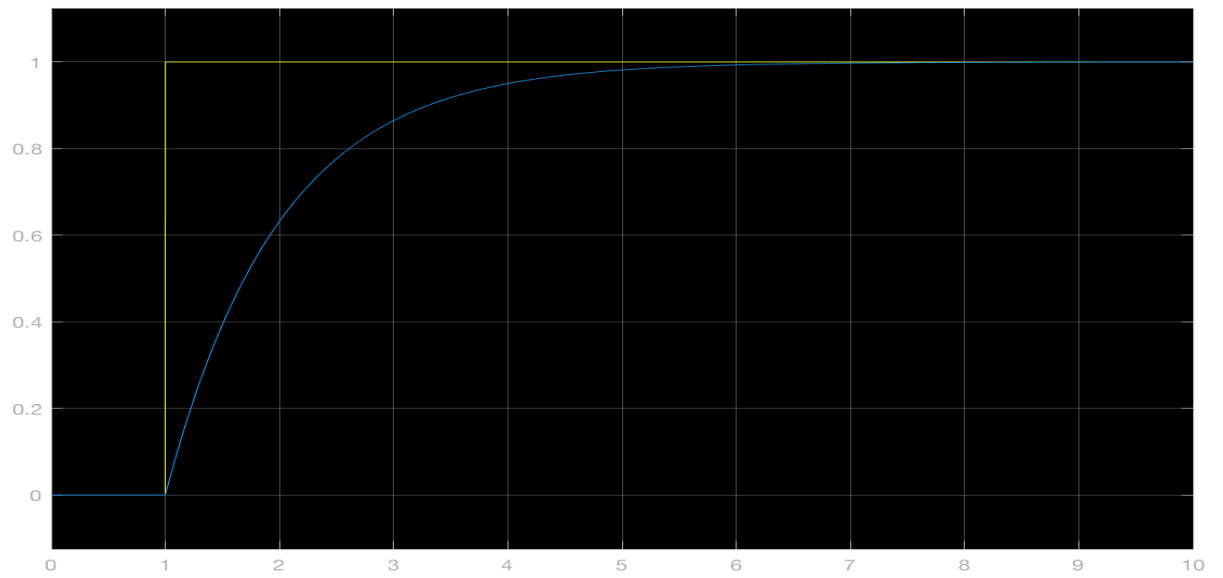
Given in that $\tau = 1$, the equation reduces to

$$x(s) = \frac{\alpha}{s + \alpha} * x_f \quad (6)$$

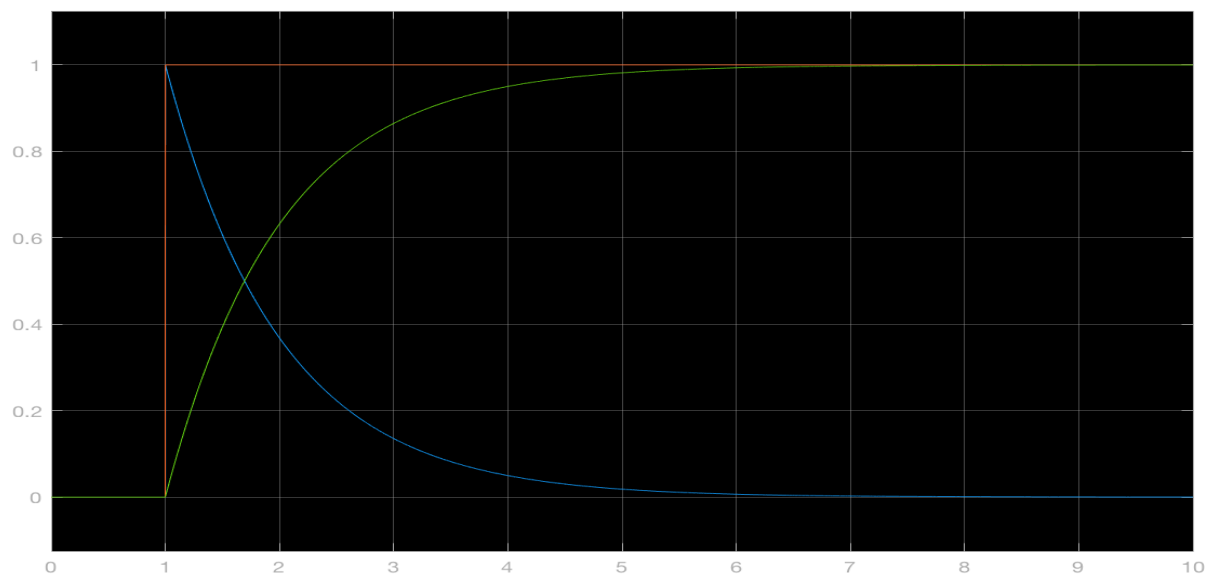
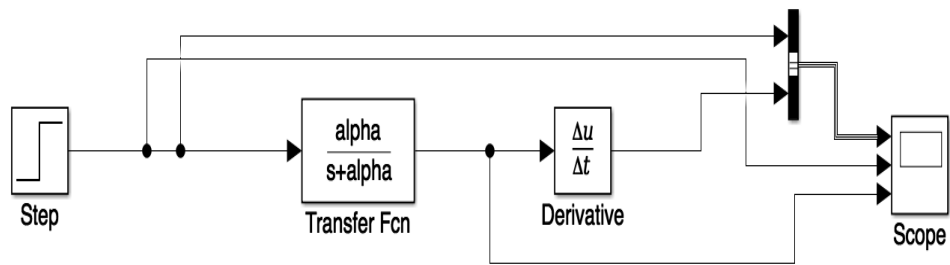
Where α will be assigned from MATLAB

- **Visualize the step input and the output of the transfer function in one Scope block.** Fixed Step = 0.001; Step up time = 1 sec; $x_0 = 0$, $x_f = 1$; $\alpha = 1$



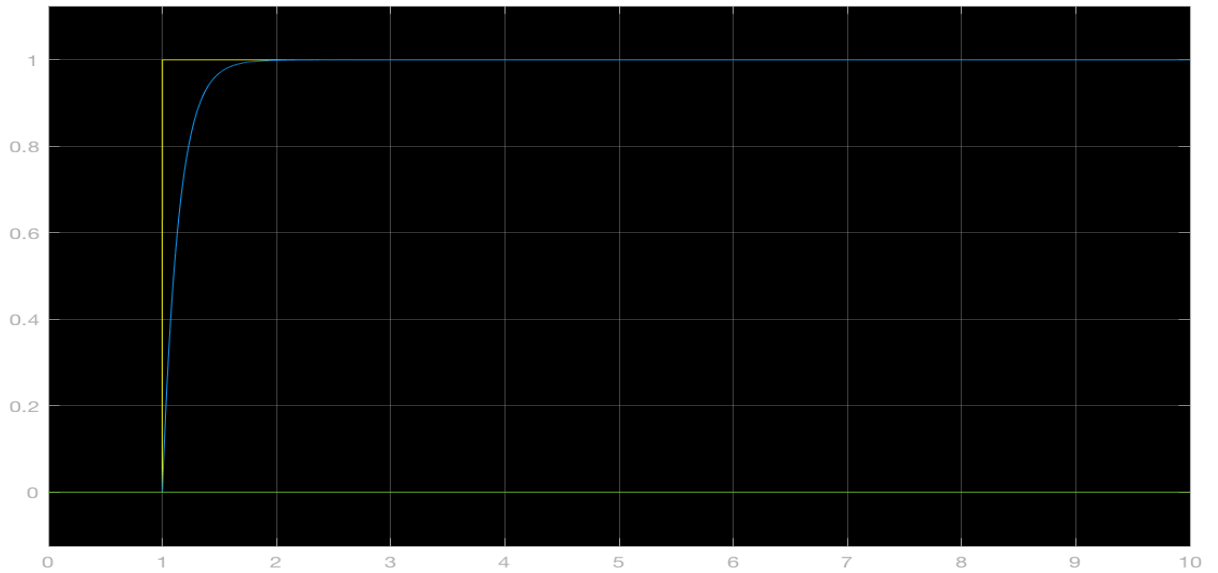


- **Scope output that visualizes \dot{x} .** Fixed Step = 0.001; Step up time = 1 sec;
 $x_0 = 0$, $x_f = 1$; $\alpha = 1$



Problem 1. (b)

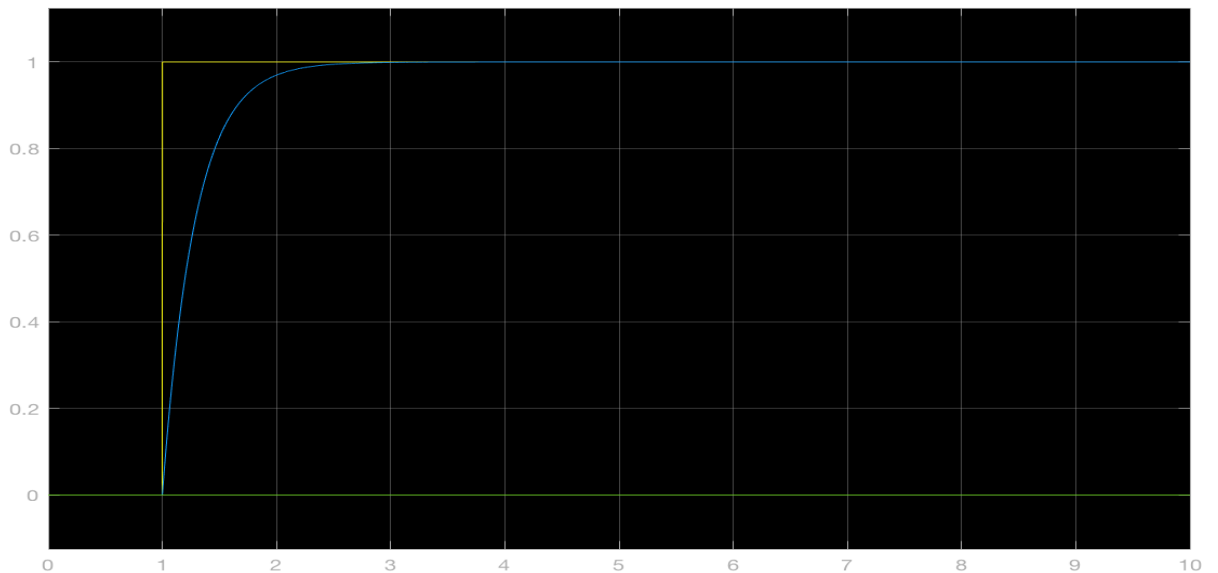
- For $\tau = 1; \alpha = 7$



α is the gain for this low pass filter. Since in this question we want the frequency to be particularly high or shoot up as soon as possible therefore gain of this transfer function is increased so as to reach the goal within limits.

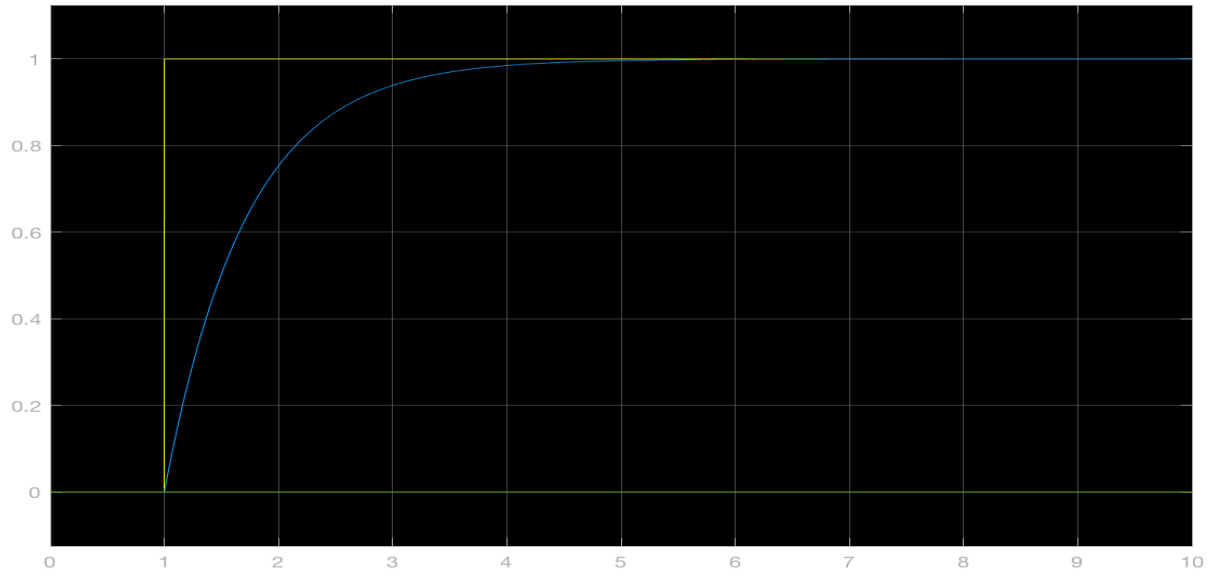
- For $\tau = 2; \alpha = 7$

Effective transfer function = $\frac{\alpha}{2s + \alpha}$

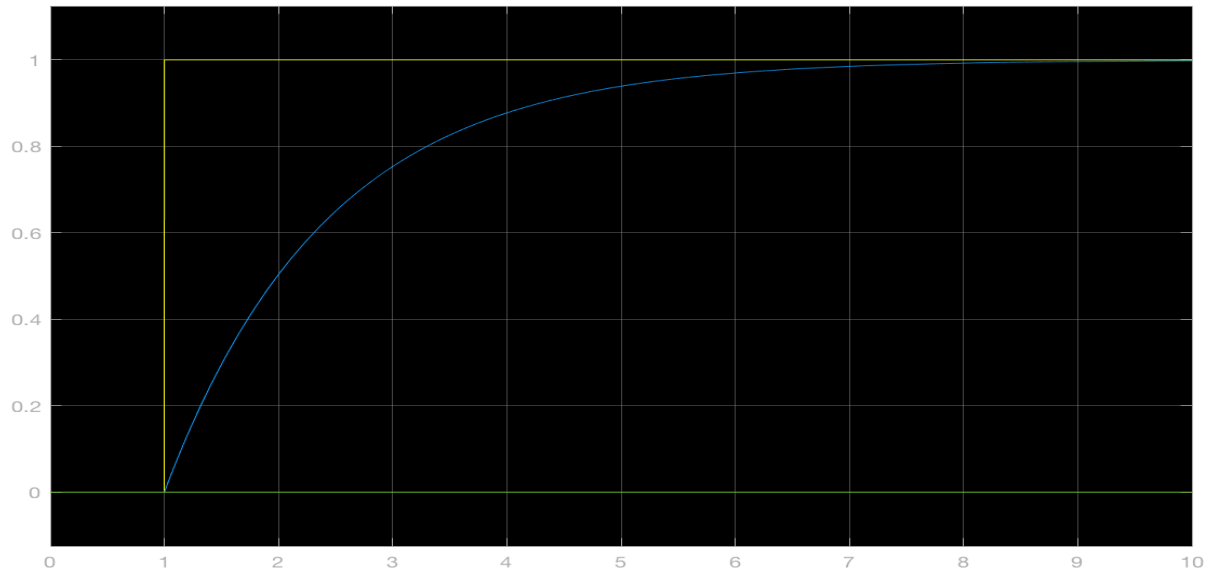


- For $\tau = 5; \alpha = 7$

Effective transfer function = $\frac{\alpha}{5s + \alpha}$



- For $\tau = 10; \alpha = 7$
Effective transfer function = $\frac{\alpha}{10s + \alpha}$



Problem 1. (c)

Pros:

- Immediate action towards the goal is taken.
- Computationally less expensive.

Cons:

- The robots initially tries to immediately reach the goal and slows down when it is about to reach the goal which can not be replicated in real-life situations.

- The goal is not actually reached. The movement is such that it just tends to be close to the goal

Problem 1. (d)

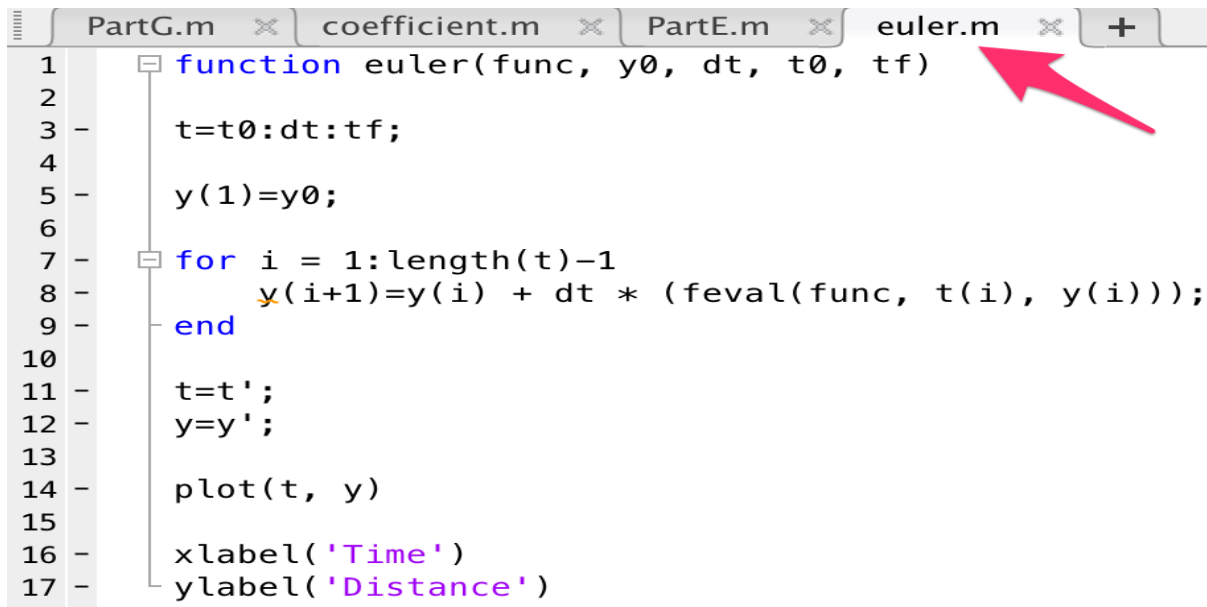
MATLAB Program : euler.m and funct.m

Euler integration methods uses the following equation

$$y_{i+1} = y_i + \Delta t * f(t_i, y_i) \quad (7)$$

In this question

$$f(t, y) = \frac{\alpha}{\tau} * (y_f - y) \quad (8)$$



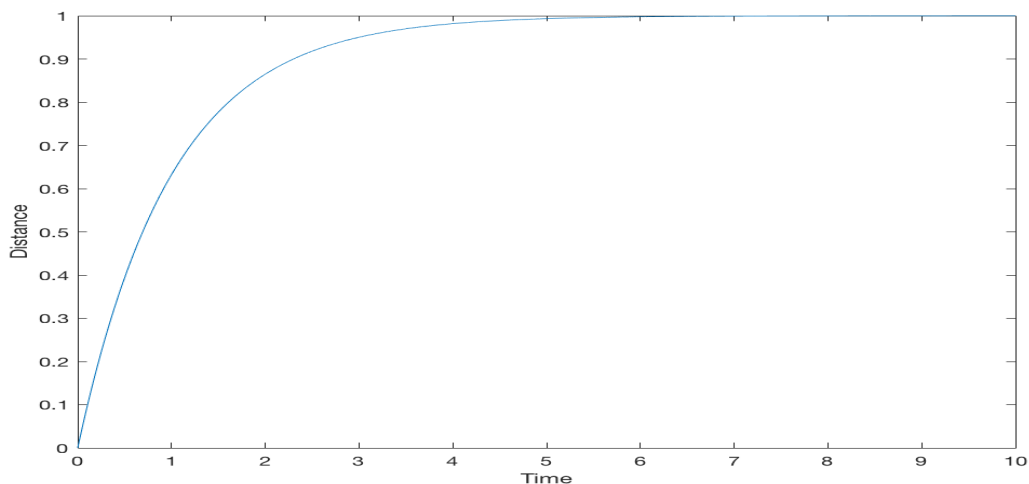
```

1 function euler(func, y0, dt, t0, tf)
2
3     t=t0:dt:tf;
4
5     y(1)=y0;
6
7     for i = 1:length(t)-1
8         y(i+1)=y(i) + dt * (feval(func, t(i), y(i)));
9     end
10
11     t=t';
12     y=y';
13
14     plot(t, y)
15
16     xlabel('Time')
17     ylabel('Distance')

```

To run the code use command : `euler(@funct, 0, 0.001, 0, 10)`

Output:



Comparison with part (a) The transfer function used in part (a) is an example of analog controller whereas Euler's method is discrete method which finds next position using incremental integration at discrete time steps. Euler reaches the target earlier than Transfer function and Euler's graph is smoother than the one in part(a)

Problem 1. (e)

MATLAB CODE: *PartE.m* and *PartEfunct.m*

$$\ddot{x} = \frac{\alpha}{\tau} * (\beta(x_f - x) - \dot{x}) \quad (9)$$

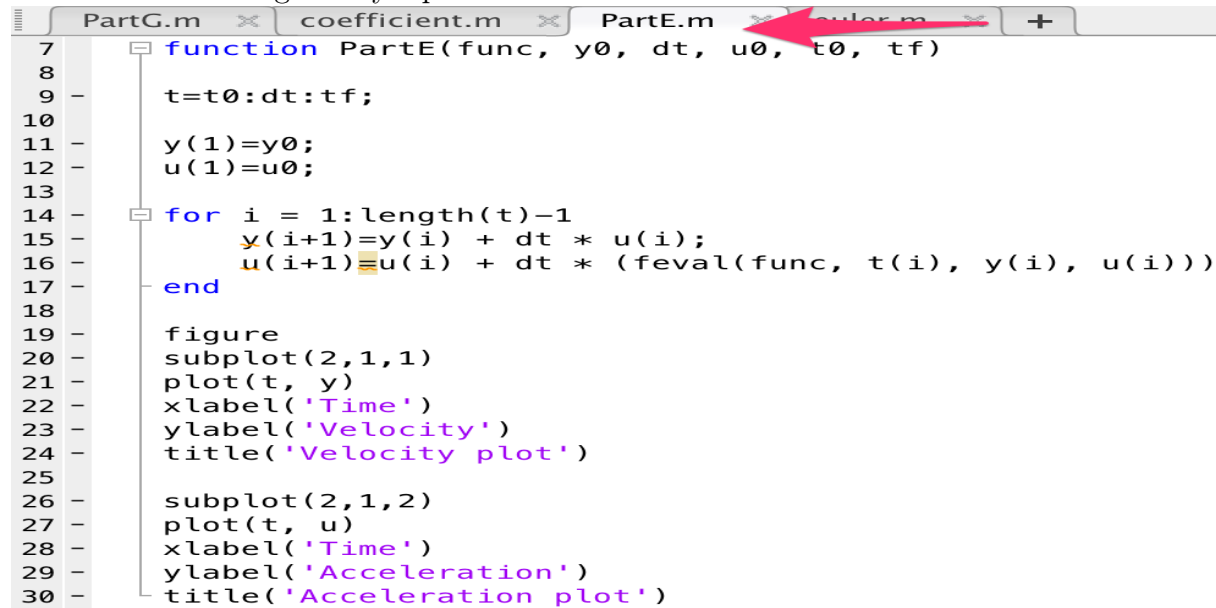
where $\alpha = 24$, $\beta = 6$, $x_f = 1$ Euler's equation used are:

$$t_{i+1} = t_i + \Delta t \quad (10)$$

$$x_{i+1} = x_i + \Delta t * u_i \quad (11)$$

$$u_{i+1} = u_i + \Delta t * f(t_i, x_i, u_i) \quad (12)$$

where function f is given by equation 9

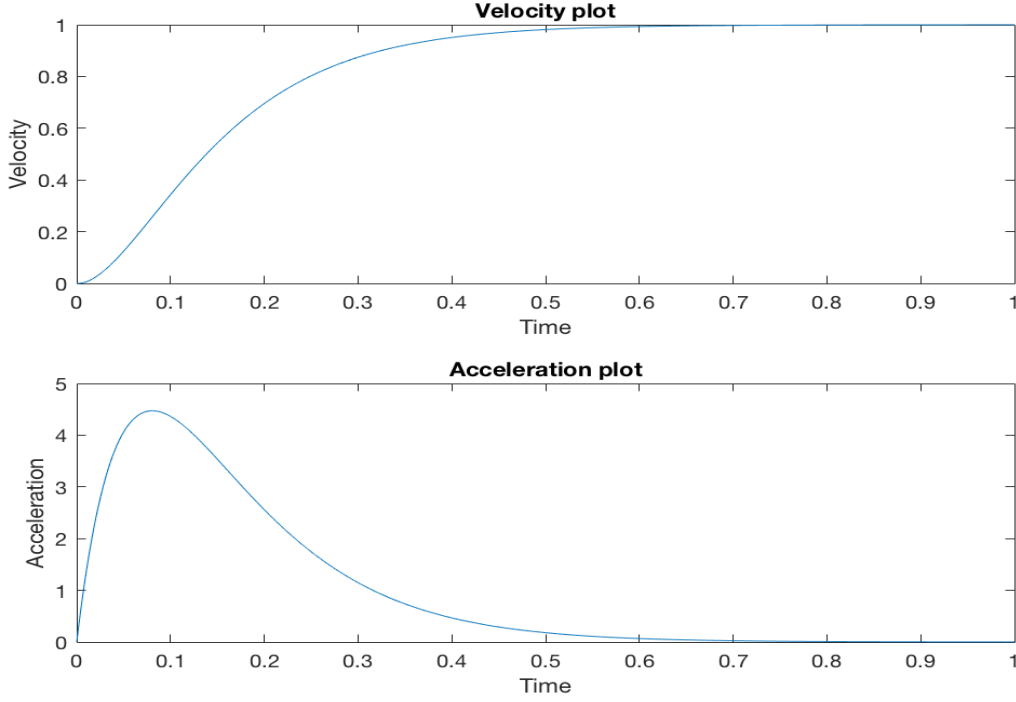


```

7 function PartE(func, y0, dt, u0, t0, tf)
8
9     t=t0:dt:tf;
10
11     y(1)=y0;
12     u(1)=u0;
13
14     for i = 1:length(t)-1
15         y(i+1)=y(i) + dt * u(i);
16         u(i+1)=u(i) + dt * (feval(func, t(i), y(i), u(i)))
17     end
18
19     figure
20     subplot(2,1,1)
21     plot(t, y)
22     xlabel('Time')
23     ylabel('Velocity')
24     title('Velocity plot')
25
26     subplot(2,1,2)
27     plot(t, u)
28     xlabel('Time')
29     ylabel('Acceleration')
30     title('Acceleration plot')

```

To run use the command : **PartE(@PartEfunct, 0, 0.001, 0, 0, 1**
OUTPUT:



This dynamic system is second order whereas the one used in part (a) was based on first order equations. The target here is reached approx 10 times faster. Although there is initial velocity gain as in part (a) and then it remains when the robot is about to reach the target.

Problem 1. (f)

Coefficients c_0 , c_1 and c_2 were found using manual calculations whereas for c_3, c_4 and c_5 I used MATLAB. Given

$$x(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \quad (13)$$

$$x'(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4 \quad (14)$$

$$x''(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3 \quad (15)$$

Equating $x(t) = 0$ at $t = 0$, we get $c_0 = x_0$ because given that start position is x_0 . Hence, $c_0 = x_0$

Equating $x'(t)$ to \dot{x} at $t = 0$, since velocity at $t = 0$ is given as \dot{x}
Therefore, $x'(0) = c_1 = \dot{x}$

Equating $x''(t)$ to \ddot{x} at $t = 0$. We get, $c_2 = \ddot{x}/2$

To compute c_3, c_4, c_5 I used MATLAB. The values received in terms of $\mathbf{x}_0, \dot{\mathbf{x}}_0, \ddot{\mathbf{x}}_0, \mathbf{x}_f, \dot{\mathbf{x}}_f, \ddot{\mathbf{x}}_f$ and t are

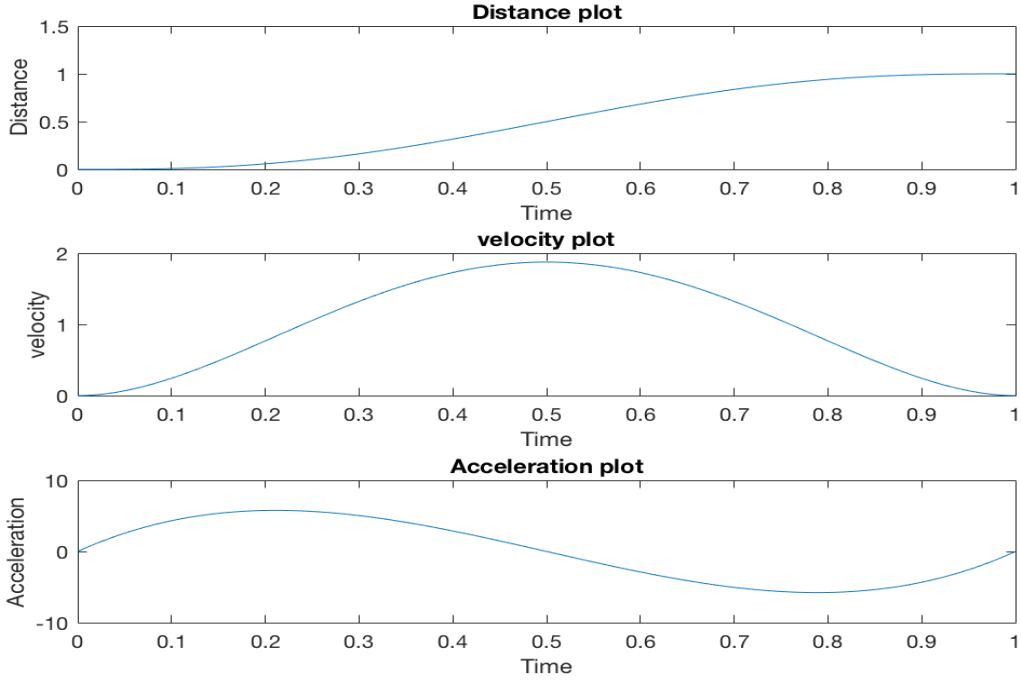
$$c_3 = -(20 * \mathbf{x}_0 - 20 * \mathbf{x}_f + 12 * t * \dot{\mathbf{x}}_0 + 8 * t * \dot{\mathbf{x}}_f + 3 * \ddot{\mathbf{x}}_0 * t^2 - \ddot{\mathbf{x}}_f * t^2) / (2 * t^3) \quad (16)$$

$$c_4 = (30 * \mathbf{x}_0 - 30 * \mathbf{x}_f + 16 * t * \dot{\mathbf{x}}_0 + 14 * t * \dot{\mathbf{x}}_f + 3 * \ddot{\mathbf{x}}_0 * t^2 - 2 * \ddot{\mathbf{x}}_f * t^2) / (2 * t^4) \quad (17)$$

$$c_5 = -(12 * x_0 - 12 * x_f + 6 * t * \dot{x}_0 + 6 * t * \dot{x}_f + \ddot{x}_0 * t^2 - \ddot{x}_f * t^2)/(2 * t^5) \quad (18)$$

Problem 1. (g)

MATLAB Code : PartG.m and coefficients.m



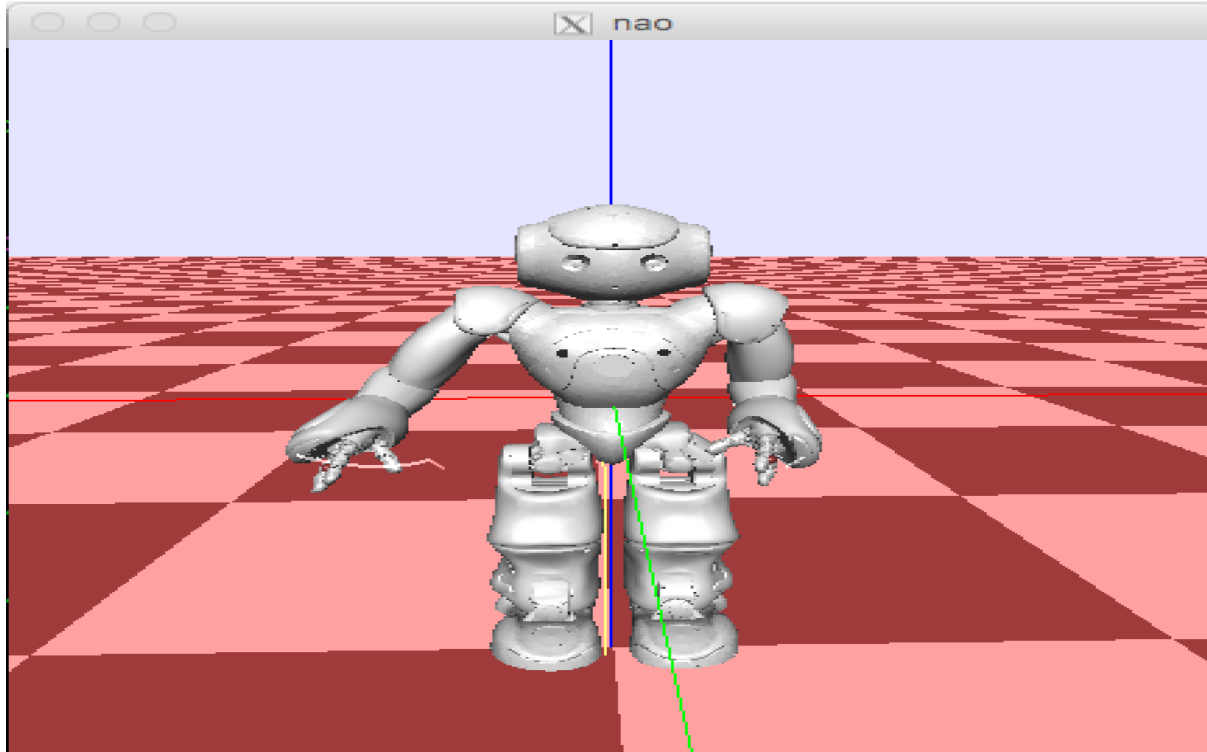
On

comparing the outputs with the ones achieved in part (e) we can see velocity here increases gradually and then gradually slows down when almost reached the goal value. Whereas in part (e) there is a jerk and sudden velocity rise.

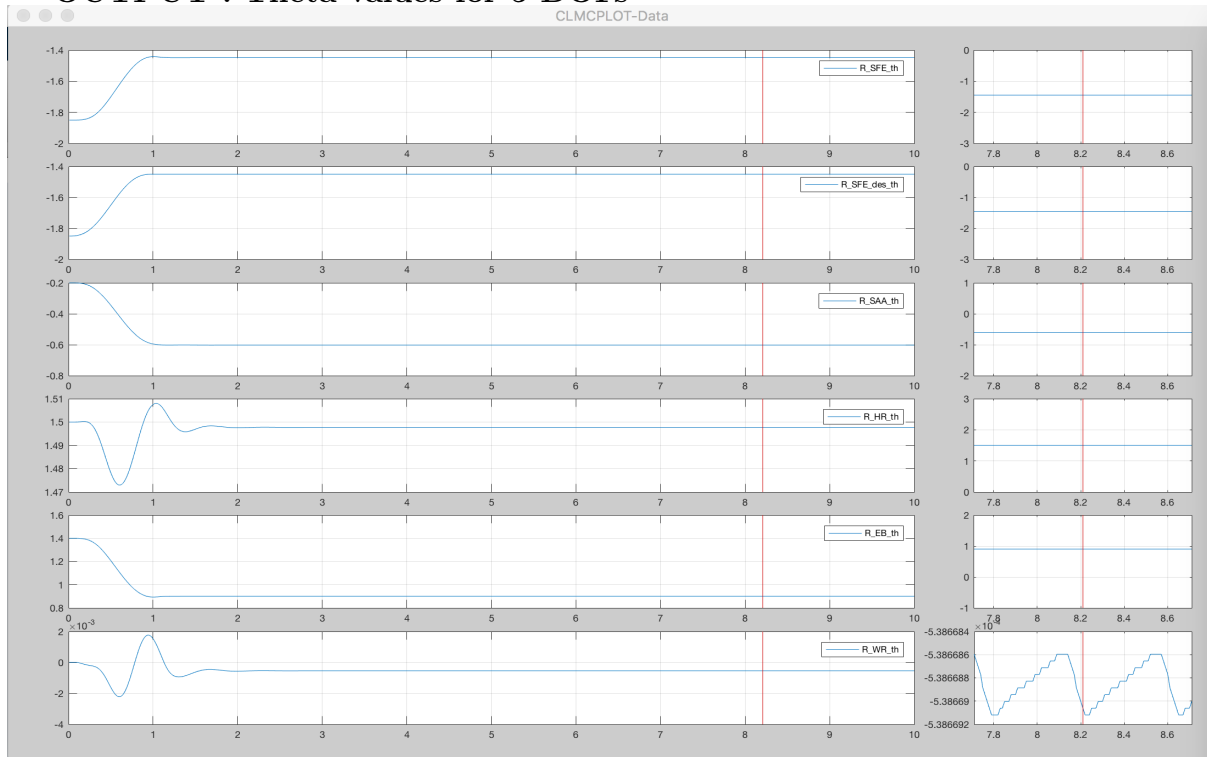
Problem 1. (h)

C++ Code : min jerk task.cpp

```
254 static int
255 min_jerk_next_step (double x, double xd, double xdd, double t, double td, double tdd,
256                    double t_togo, double dt,
257                    double *x_next, double *xd_next, double *xdd_next)
258 {
259     double c0, c1, c2, c3, c4, c5;
260     c0 = c1 = c2 = c3 = c4 = c5 = 0;
261
262     findCoefficients(x, xd, xdd, t, td, tdd, t_togo, &c0, &c1, &c2, &c3, &c4, &c5);
263
264     *x_next = c0 + c1 * dt + c2 * pow(dt, 2) + c3 * pow(dt, 3) + c4 * pow(dt, 4) + c5 * pow(dt, 5);
265     *xd_next = c1 + 2 * c2 * dt + 3 * c3 * pow(dt, 2) + 4 * c4 * pow(dt, 3) + 5 * c5 * pow(dt, 4);
266     *xdd_next = 2 * c2 + 6 * c3 * dt + 12 * c4 * pow(dt, 2) + 20 * c5 * pow(dt, 3);
267     return TRUE;
268 }
269
270
271 static int
272 findCoefficients(double x, double xd, double xdd, double tf, double tf_d, double tf_dd,
273                double tau, double *c0, double *c1, double *c2, double *c3, double *c4, double *c5 )
274 {
275     *c0 = x;
276     *c1 = xd;
277     *c2 = xdd/2;
278
279     *c3 = -(20*x - 20*tf + 12*tau*xd + 8*tau*tf_d + 3*xdd*pow(tau,2) - tf_dd*pow(tau,2))/(2*pow(tau,3));
280
281     *c4 = (30*x - 30*tf + 16*tau*xd + 14*tau*tf_d + 3*xdd*pow(tau,2) - 2*tf_dd*pow(tau,2))/(2*pow(tau,4));
282
283     *c5 = -(12*x - 12*tf + 6*tau*xd + 6*tau*tf_d + xdd*pow(tau,2) - tf_dd*pow(tau,2))/(2*pow(tau,5));
284     return TRUE;
285 }
```

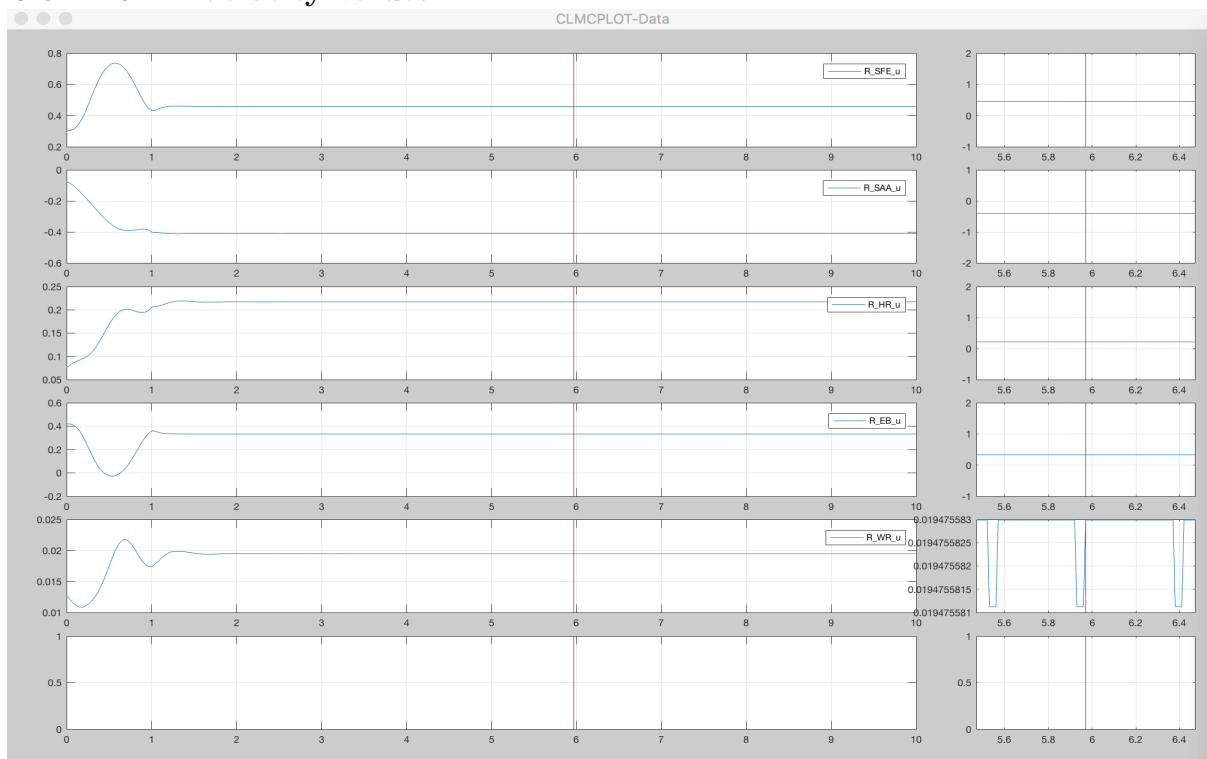


OUTPUT : Theta values for 5 DOFs

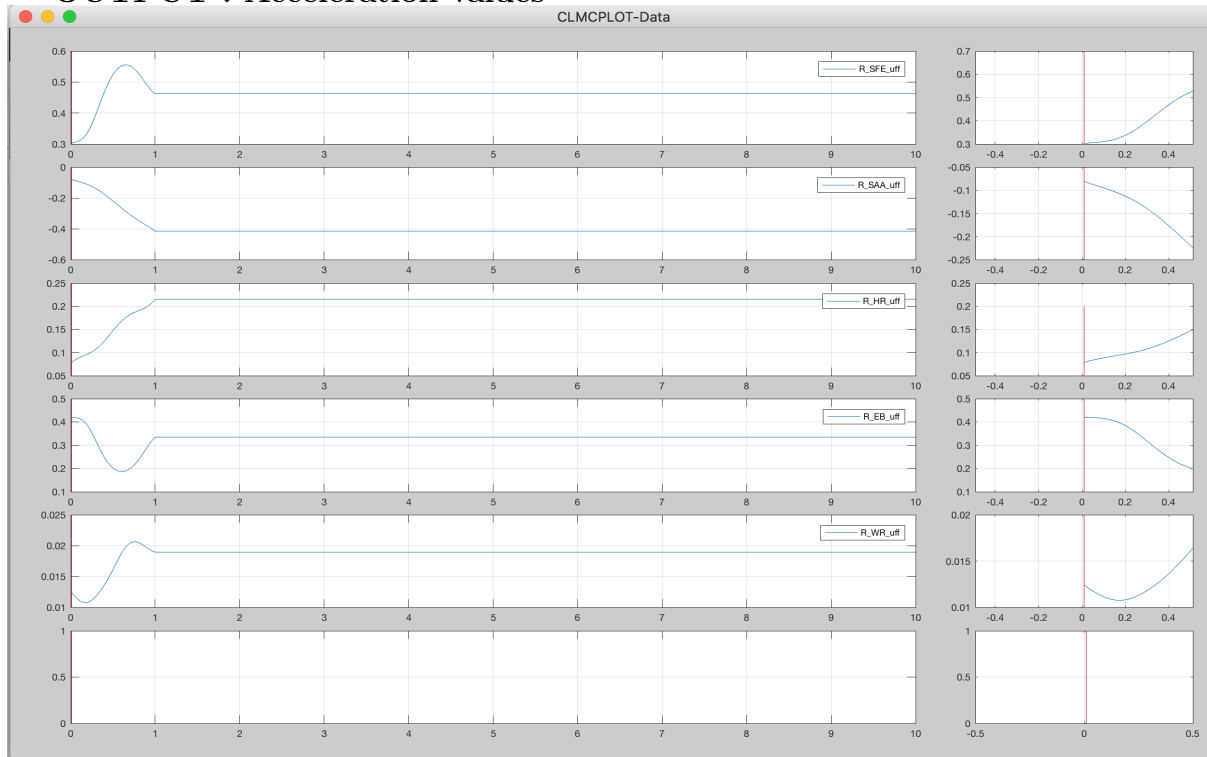


The values are captured accurately.

OUTPUT : Velocity values



OUTPUT : Acceleration values



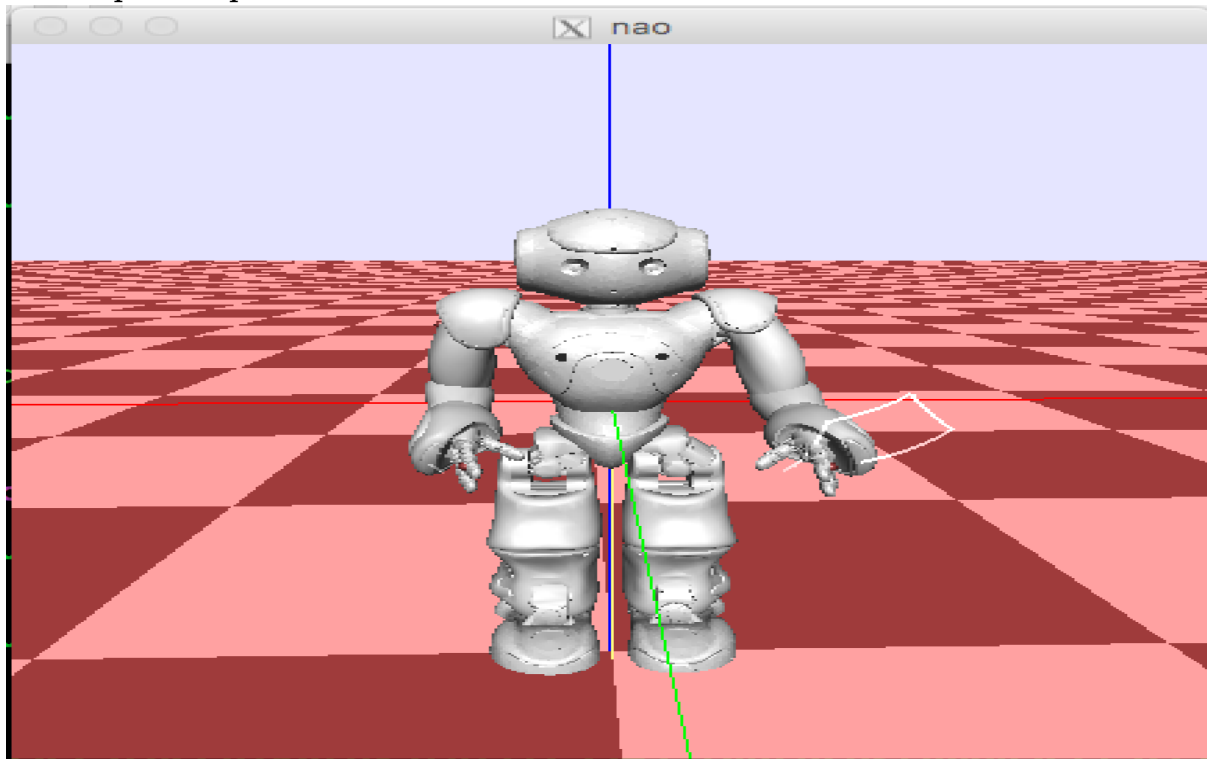
Problem 1. (i)

I have used two controls to decide the motion of the robot. To switch between part (h) and part (i) use the control variable. As shown

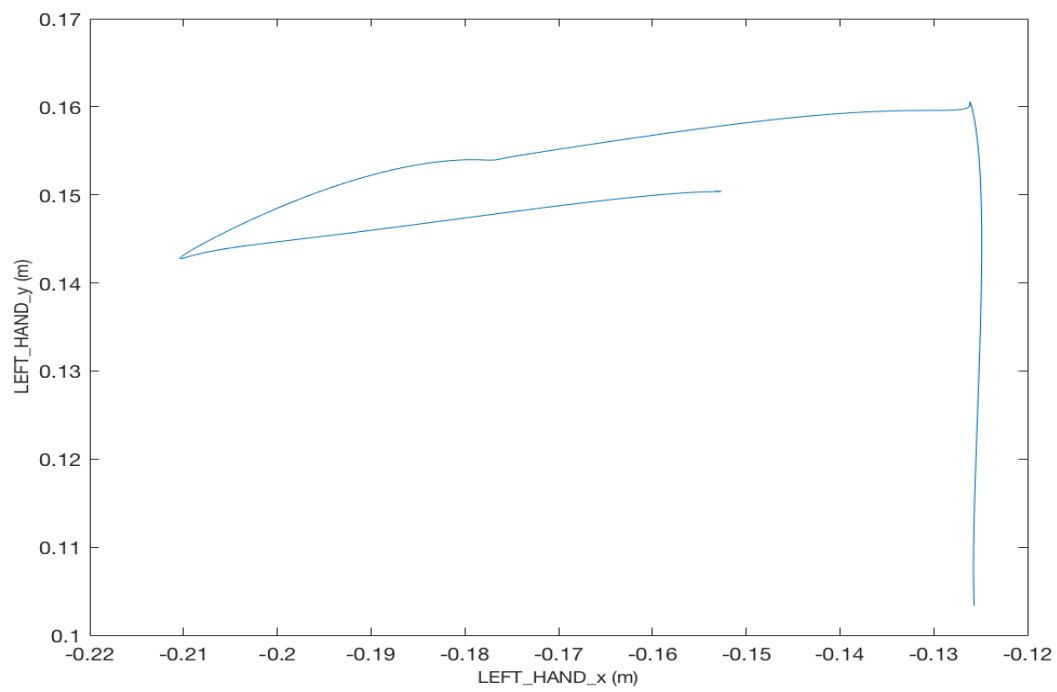
```
static int control = 2;
```

```
static bool changePlanning = false;
```

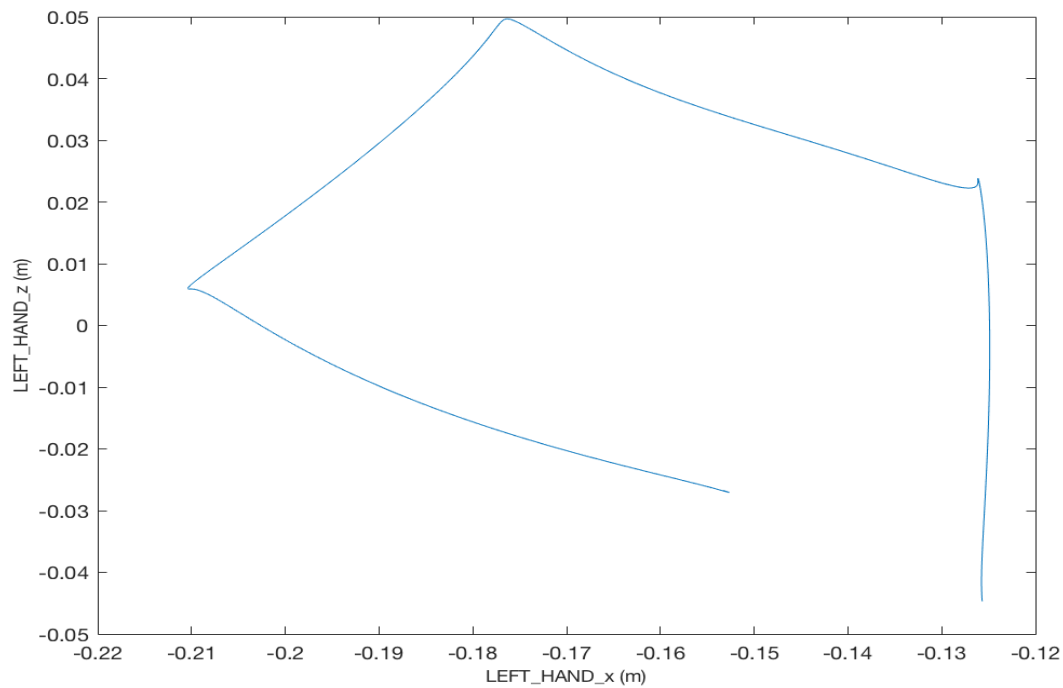
Output : square



Output : LEFT HAND x and LEFT HAND y plot



Output : LEFT HAND x and LEFT HAND z plot



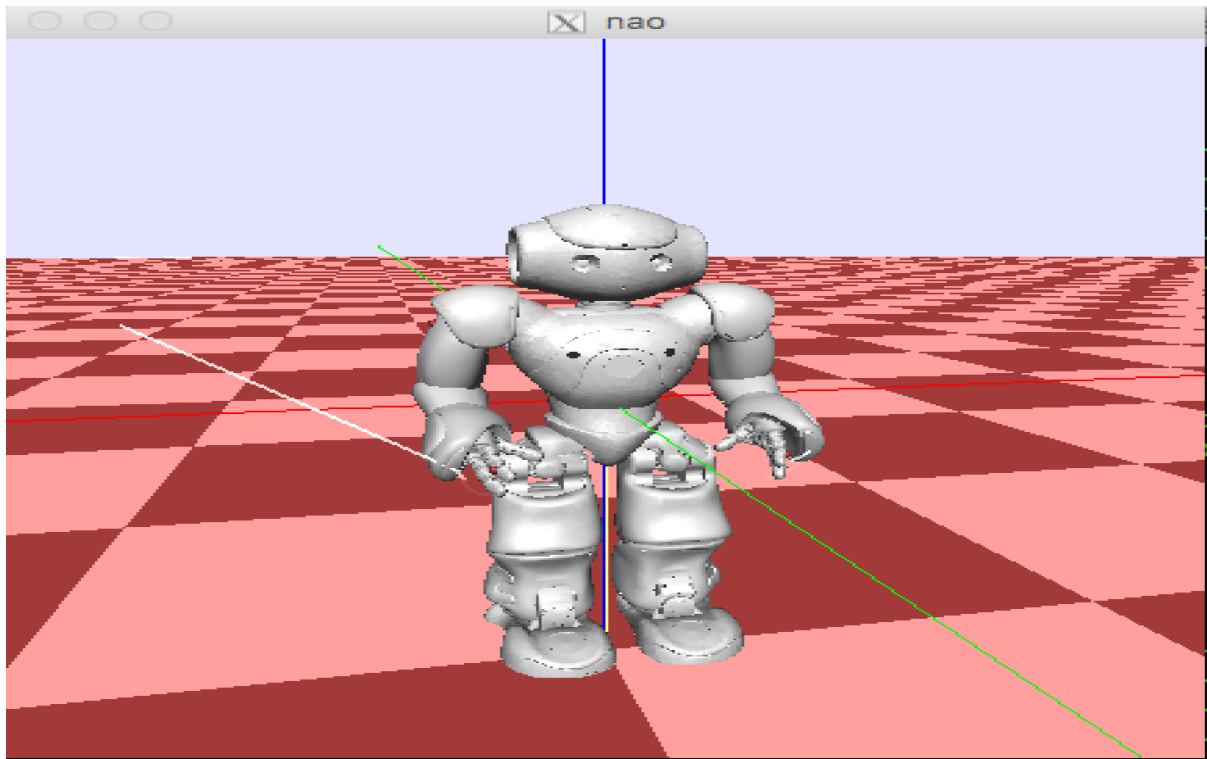
Although the plot is not perfect but it is near perfect.

Problem 1. (j)

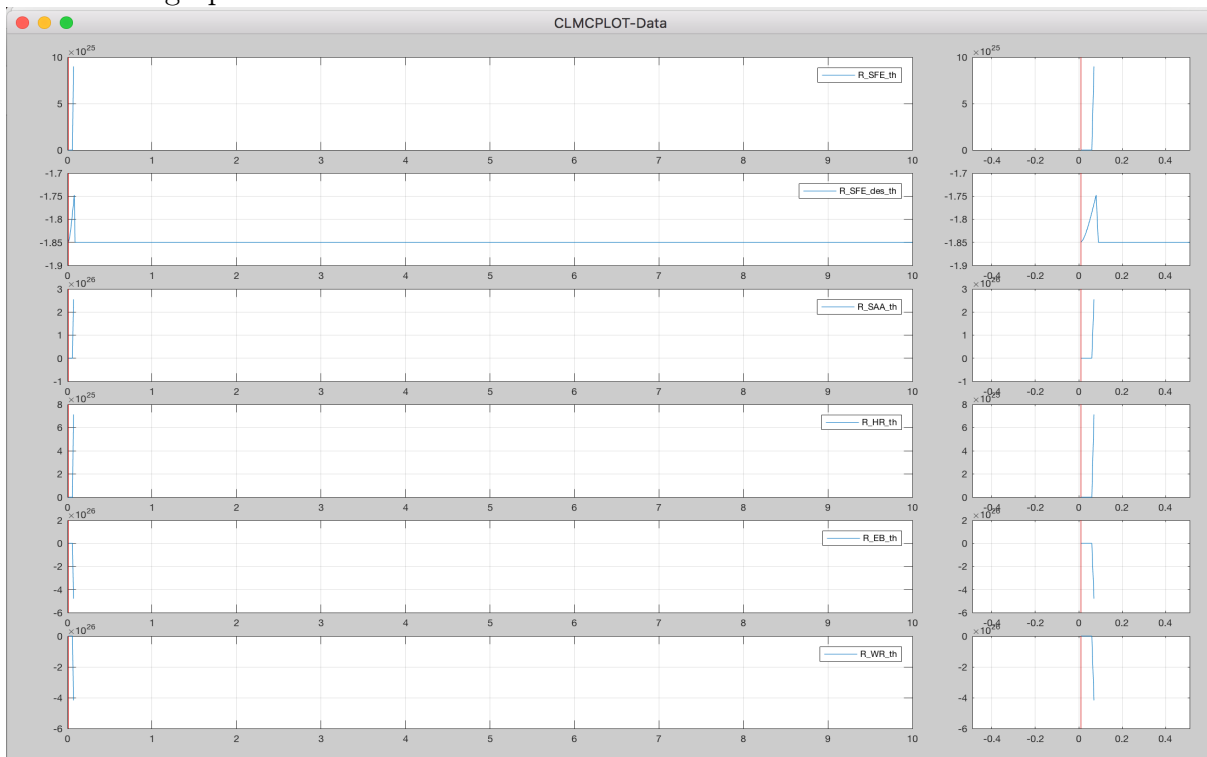
Part(h) using the dynamic system as defined in part(e) To implement this part I have used two controls. One to control drawing a square or an arc. If you want robot to draw square use control = 1 and changePlanning as true to use dynamic system as discussed in part (h). As shown

```
static int control = 1;
static bool changePlanning = true;
```

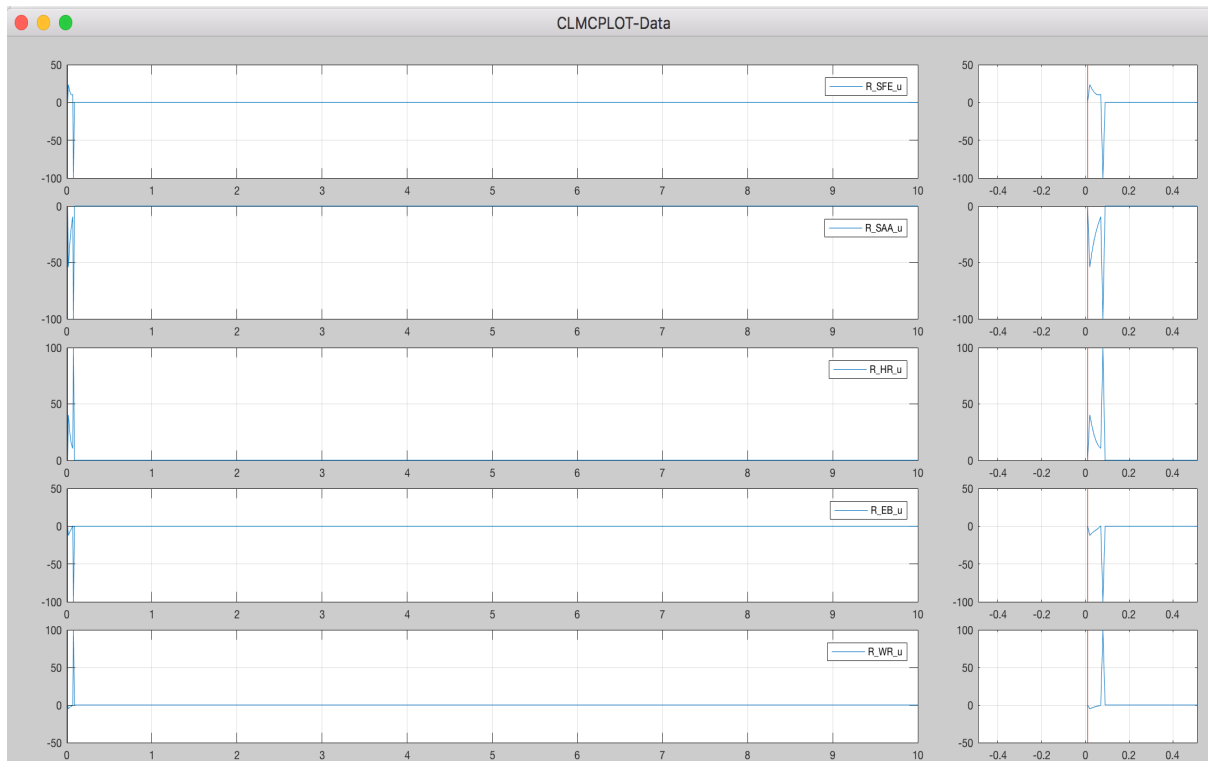
Here is the result of arc. It can be clearly seen the spline in this case is straighter than the previous case rather than curvy as it was in case of using coefficients. Whereas in Euler algorithm some details are lost and curve is rather straight.



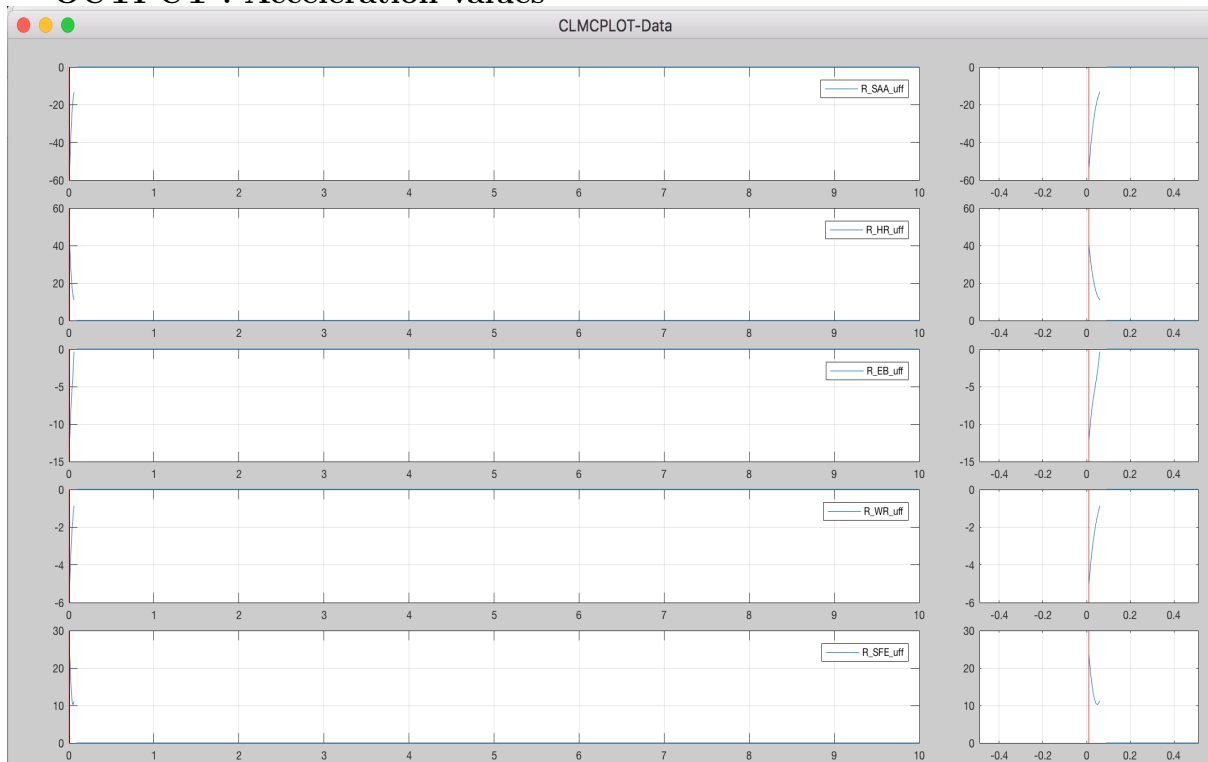
Here is the graph for various theta values



OUTPUT : Velocity values



OUTPUT : Acceleration values

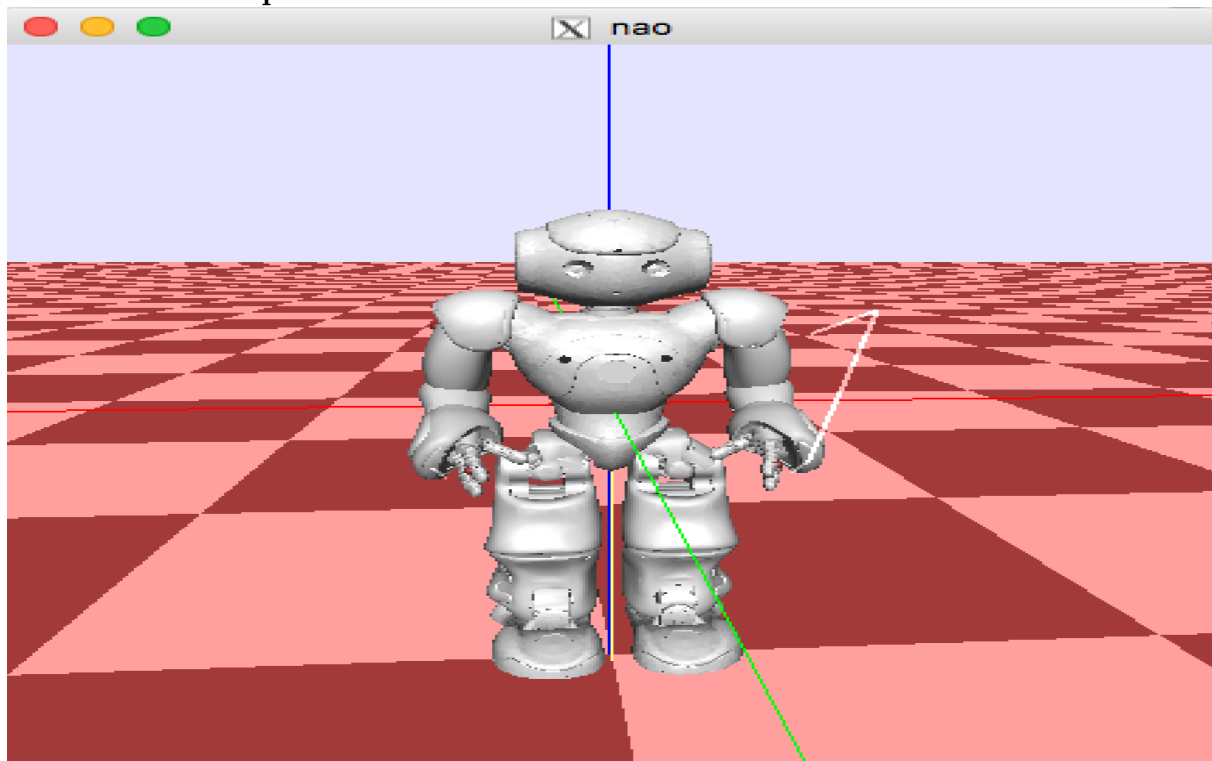


Part(i) using the dynamic system as defined in part(e)
 To implement this part I have used two controls. One to control drawing a square or an

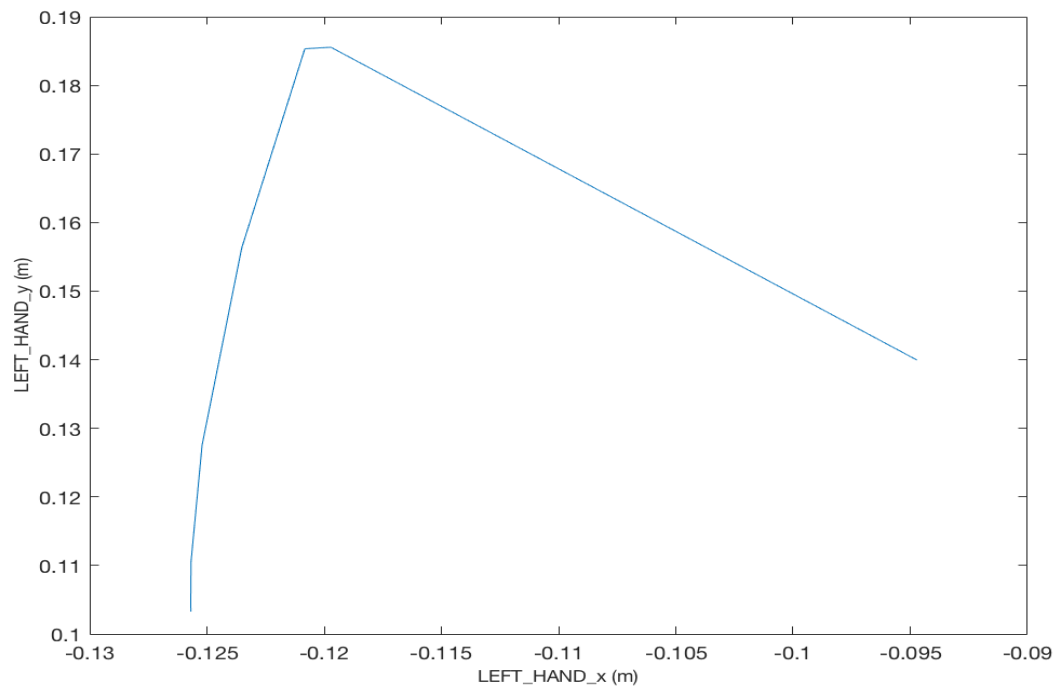
arc. If you want robot to draw square use control = 2 and changePlanning as true to use dynamic system as discussed in part (h)

```
static int control = 2;  
static bool changePlanning = true;
```

OUTPUT: Square



OUTPUT: LEFT HAND x and LEFT HAND y plot



OUTPUT: LEFT HAND x and LEFT HAND y plot

