ZHANPENG HE

1. a

1. $x(t) = C_0 + C_1 t + C_2 t^2 + C_3 t^3$

$\dot{x}(t) = C_1 + 2C_2 t + 3C_3 t^2$

$$\Rightarrow \begin{cases} x(0) = C_0 = x_0 \\ x(\tau) = C_0 + C_1\tau + C_2\tau^2 + C_3\tau^3 = x_f \\ \dot{x}(0) = C_1 = \dot{x}_0 \\ \dot{x}(\tau) = C_1 + 2C_2\tau + 3C_3\tau^2 = \dot{x}_f \end{cases}$$

$$3x_0 + 3\tau\cdot\dot{x}_0 + 3\tau^2\cdot C_2 + 3\tau^3\cdot C_3 = 3x_f$$

$$\dot{x}_0\tau + 2\tau^2 C_2 + 3\tau^3\cdot C_3 = \dot{x}_f\tau$$

$$2x_0 + \tau\dot{x}_0 - \tau^3 C_3 = 2x_f - \dot{x}_f\tau$$

$$C_3 = \frac{2x_0 + \tau\dot{x}_0 - 2x_f + \dot{x}_f\tau}{\tau^3}$$

$$3x_0 + 2\tau\dot{x}_0 + \tau^2 C_2 = 3x_f - \dot{x}_f\tau$$

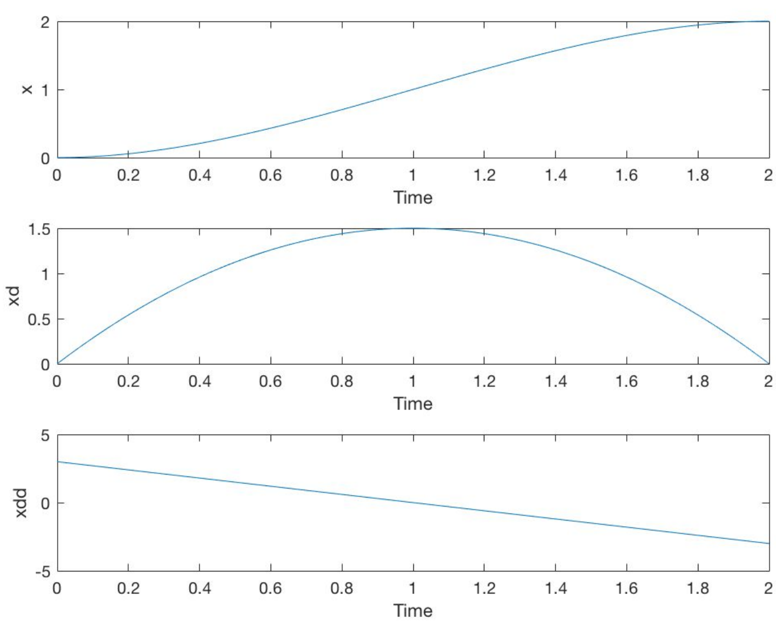
$$C_2 = \frac{3x_f - \tau\cdot\dot{x}_f - 3x_0 - 2\tau\dot{x}_0}{\tau^2}$$

$$\Rightarrow \begin{cases} C_0 = x_0 \\ C_1 = \dot{x}_0 \\ C_2 = \dfrac{3x_f - \tau\cdot\dot{x}_f - 3x_0 - 2\tau\dot{x}_0}{\tau^2} \\ C_3 = \dfrac{2x_0 + \tau\dot{x}_0 - 2x_f + \dot{x}_f\tau}{\tau^3} \end{cases}$$

ZHANPENG HE

1.b


```
x_curr = 0;
x_dot_curr = 0;
x_f = 2.0;
x_dot_f = 0;

remaining_time = 2.0;
delta_t = 0.01;

c_0 = x_curr;
c_1 = x_dot_curr;
c_2 = (3*x_f-remaining_time*x_dot_f-3*x_curr-2*remaining_time*x_dot_curr)↙
/remaining_time/remaining_time;
c_3 = (2*x_curr+remaining_time*x_dot_curr-2*x_f+remaining_time*x_dot_f)↙
/remaining_time/remaining_time/remaining_time;

pos = zeros(1, 201);
vel = zeros(1, 201);
acc = zeros(1, 201);

i = 0;

for t = 0:0.01:remaining_time
    i = i+1;
    pos(i) = c_0+c_1*t+c_2*t*t+c_3*t*t*t;
    vel(i) = c_1+2*c_2*t+3*c_3*t*t;
    acc(i) = 2*c_2+6*c_3*t;
end

x = 0.0:0.01:remaining_time;
subplot(3, 1, 1)
plot(x, pos)
subplot(3, 1, 2)
plot(x, vel)
subplot(3, 1, 3)
plot(x, acc)
```



ZHANPENG HE

1.c

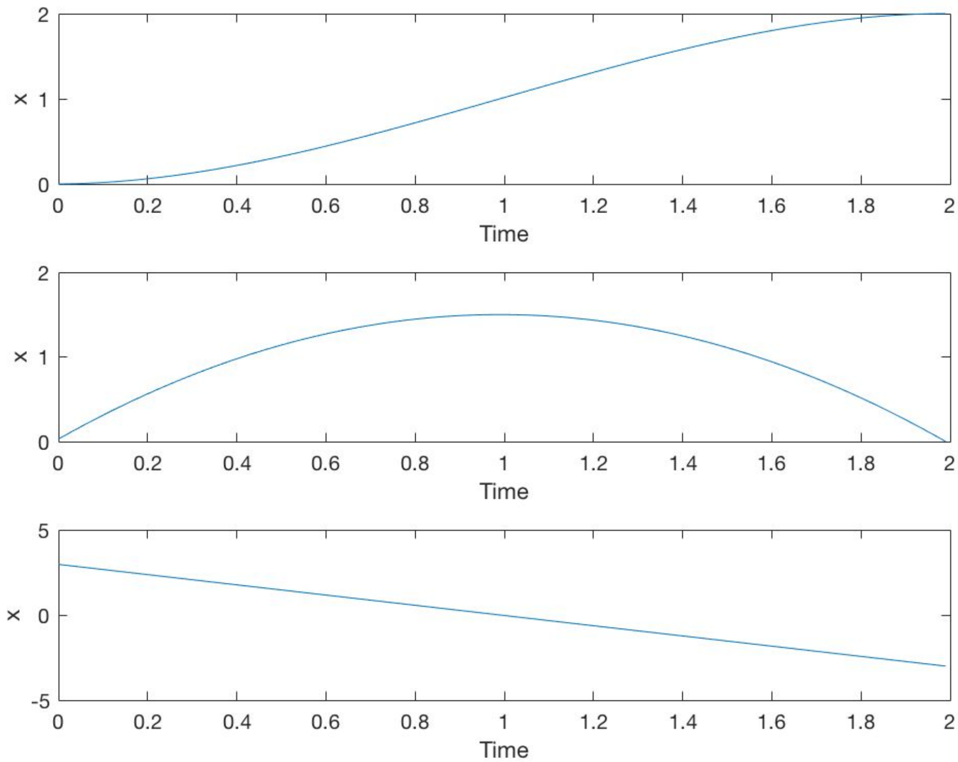
2/23/18 6:43 PM /Users/adamhzp/Documents/MATLAB/planning.m

```
function [x_t, x_dot_t, x_dot_dot_t] = planning(x_curr, x_dot_curr, x_f, x_dot_f,↙
remaining_time, delta_t)

c_0 = x_curr;
c_1 = x_dot_curr;
c_2 = (3*x_f-remaining_time*x_dot_f-3*x_curr-2*remaining_time*x_dot_curr)↙
/remaining_time/remaining_time;
c_3 = (2*x_curr+remaining_time*x_dot_curr-2*x_f+remaining_time*x_dot_f)↙
/remaining_time/remaining_time/remaining_time;

t = delta_t;

x_t = c_0+c_1*t+c_2*t*t+c_3*t*t*t;
x_dot_t = c_1+2*c_2*t+3*c_3*t*t;
x_dot_dot_t = 2*c_2+6*c_3*t;

end
```

2/23/18 6:44 PM /Users/adamhzp/Documents/MATLAB/run.m

```
x_curr = 0;
x_dot_curr = 0;
x_f = 2;
x_dot_f = 0;
tao = 2;
delta_t = 0.01;

pos = zeros(1, 201);
vel = zeros(1, 201);
acc = zeros(1, 201);
re_vec = zeros(1, 4);
i = 0;

for t = 0.0:0.01:tao
    i = i+1;
    remaining_time = tao - t;
    [x_curr,x_dot_curr,x_dot_dot_curr]=planning(x_curr, x_dot_curr, x_f, x_dot_f,↙
remaining_time, delta_t);

    pos(i) = x_curr;
    vel(i) = x_dot_curr;
    acc(i) = x_dot_dot_curr;

end

x = 0.0:0.01:tao;

subplot(3, 1, 1)
plot(x, pos)
xlabel("Time")
ylabel("x")

subplot(3, 1, 2)
plot(x, vel)
xlabel("Time")
ylabel("x")

subplot(3, 1, 3)
plot(x, acc)
xlabel("Time")
ylabel("x")
```

ZHANPENG HE

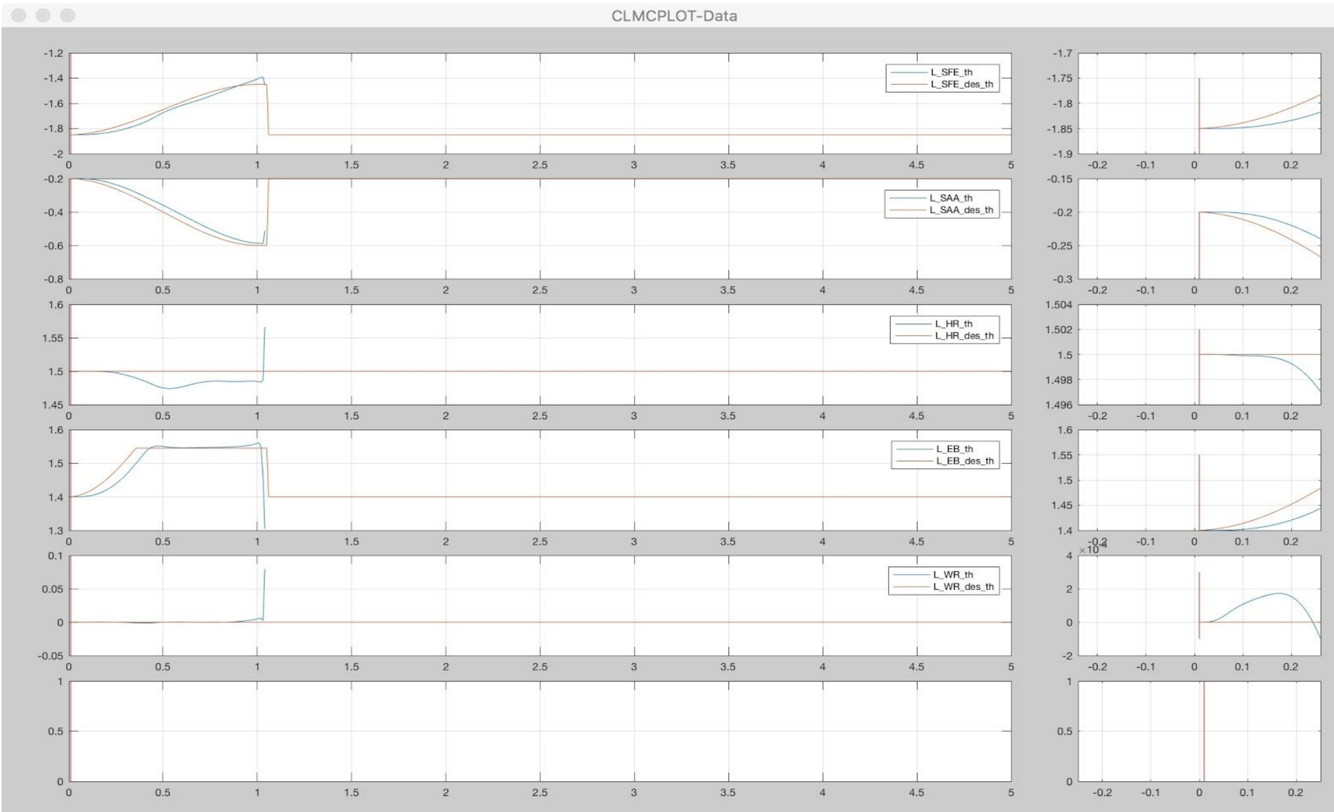


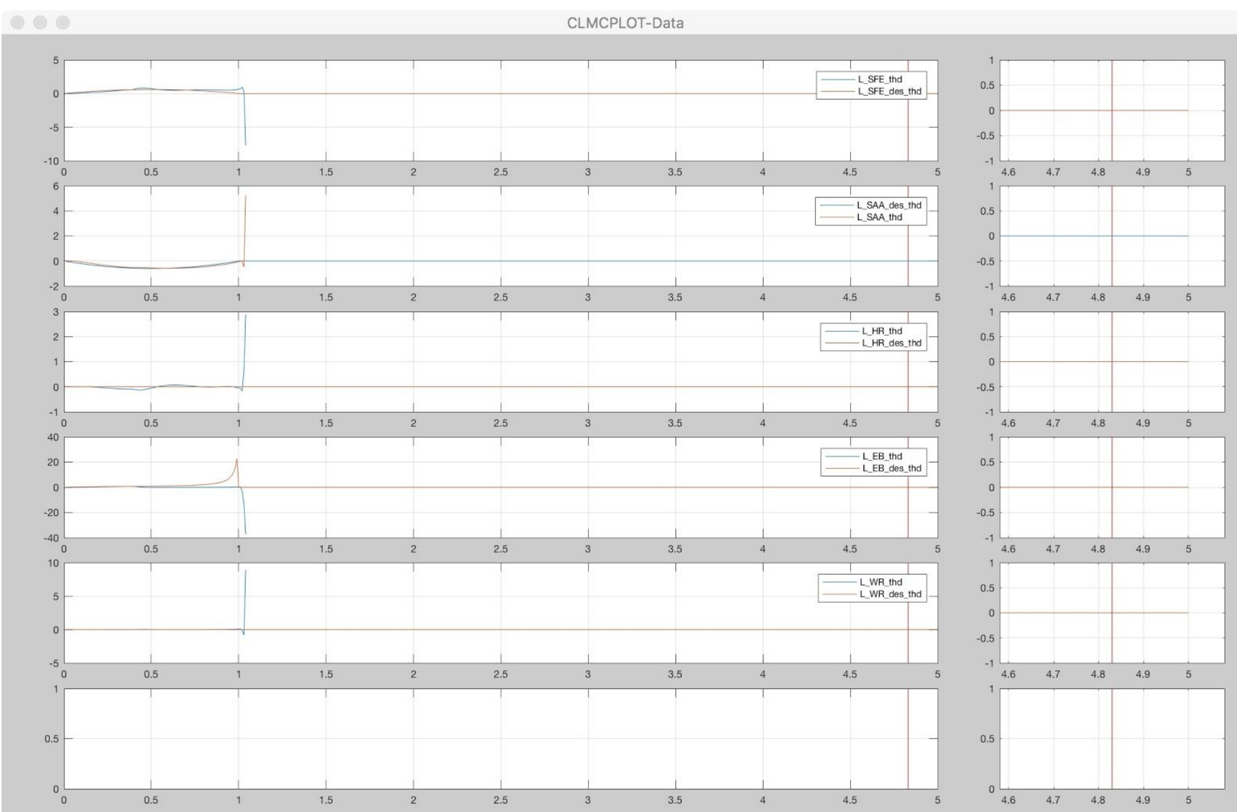
Similarity:
Both method offered similar shape of curved for position, velocity and acceleration(ie. 3rd order, 2nd order, 1st order). Both method give not bad results.

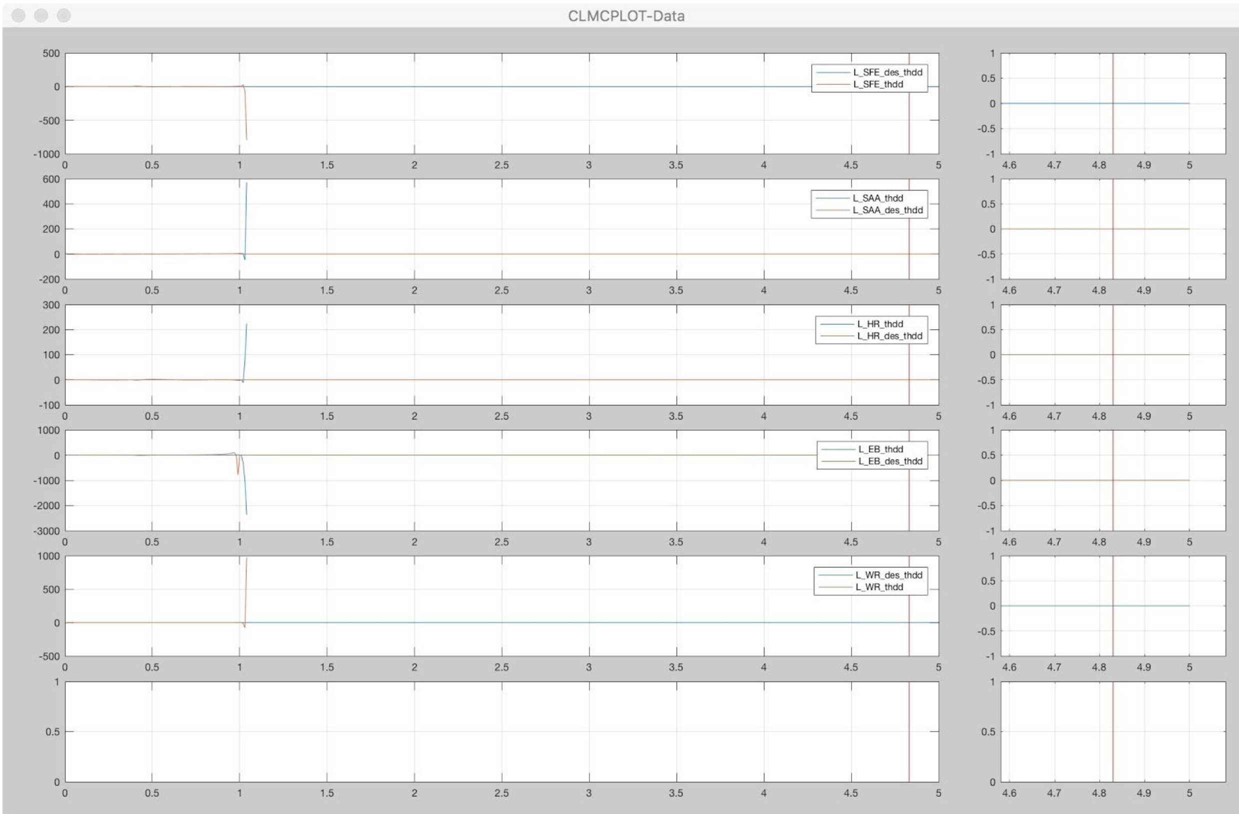
Differences:
Incremental planning could give more accurate and proper control since it recalculate the parameters at each timestep. c) reach NAN at the end because time_to_go reach 0.

ZHANPENG HE

1.d







The position is a little bit delayed, but the curves are in approximately the same shape. Velocity and acceleration have a jump at the end of one second(when time_to_go is 0).

1.e

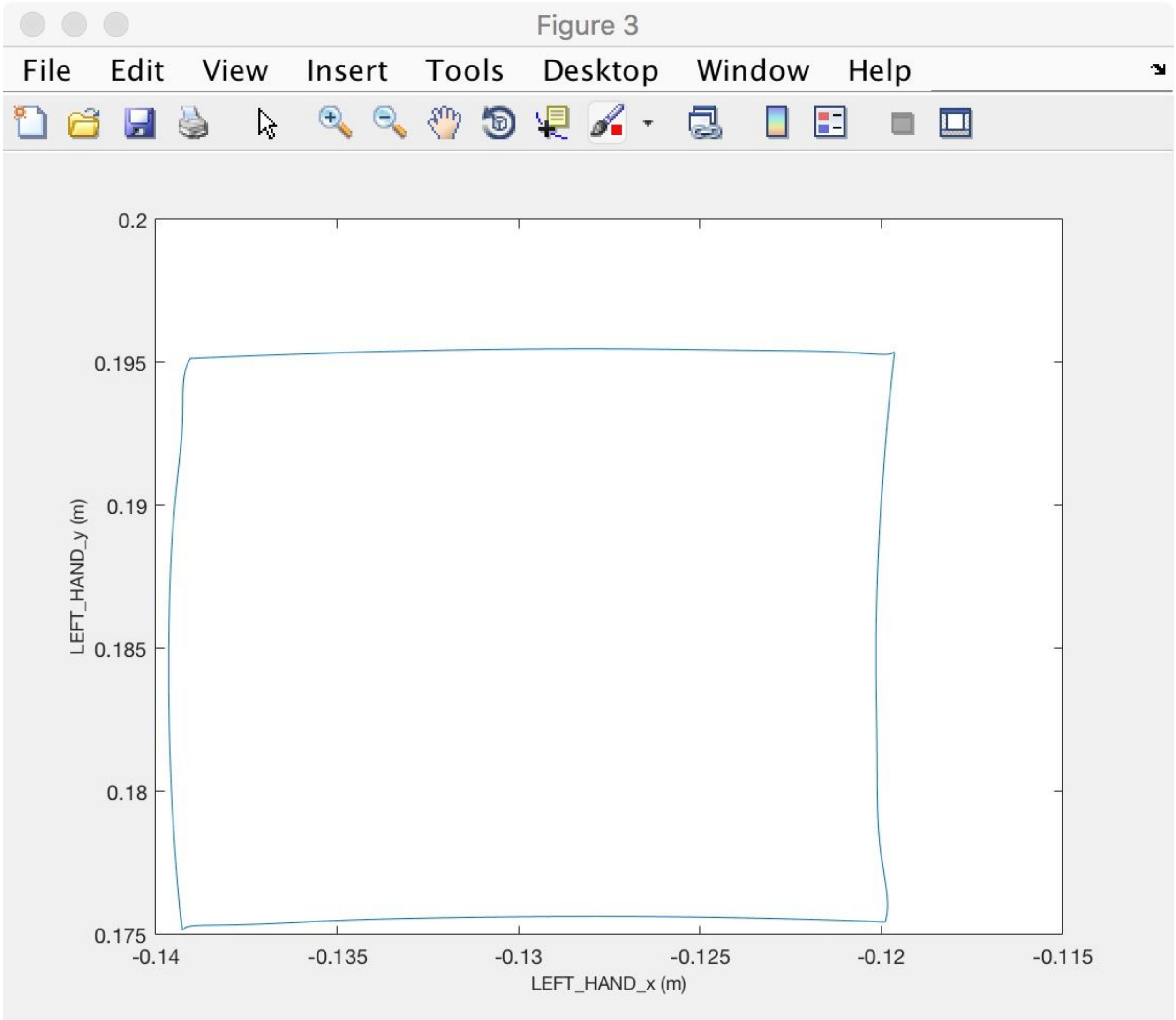
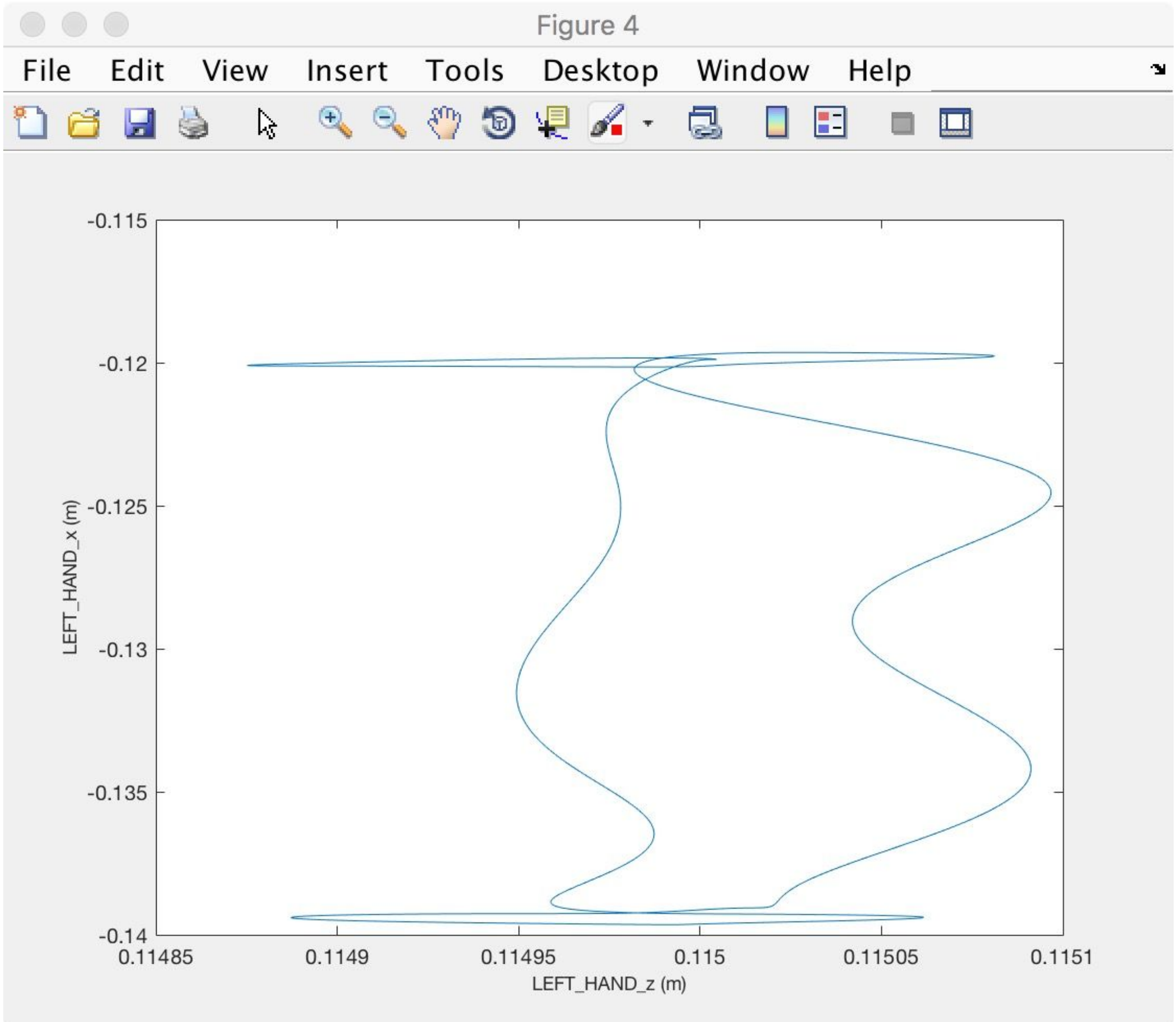


Figure 3
File Edit View Insert Tools Desktop Window Help
0.2
0.195
0.19
0.185
0.18
0.175
LEFT_HAND_y (m)
-0.14
-0.135
-0.13
-0.125
-0.12
-0.115
LEFT_HAND_x (m)



My X-Y plot shows a not bad square. However, the sides are curved.
On the X-Z, since my square is mostly on the X-Y plane, it is very weird shape on this plot.

ZHANPENG HE

2.a, b

2)

a.

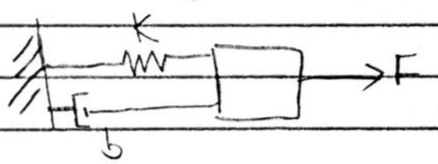
$$\ddot{x}_2 = b\dot{x}_2 + kx_2 + x_3$$

$$x_3 = \frac{\alpha(u-x_2)-\dot{x}_3}{\beta}$$

$$= \frac{\alpha}{\beta}(u-x_2) - \frac{\alpha}{\beta}\dot{x}_3$$

$$\Rightarrow \ddot{x}_2 = b\dot{x}_2 + kx_2 + \frac{\alpha}{\beta}(u-x_2) - \frac{\alpha}{\beta}\dot{x}_3$$

Here is a spring damper system that corrisponding to the equations:



Hence, k is the spring constant and b is the constant for damping

$$\Rightarrow \ddot{x}_2 = \underbrace{b\dot{x}_2}_{\text{Damping}} + \underbrace{kx_2}_{\text{spring}} + \underbrace{\frac{\alpha}{\beta}(u-x_2) - \frac{\alpha}{\beta}\dot{x}_3}_{\text{filter}}$$

$\Rightarrow \alpha, \beta$ are parameter for a filter.

b) $$\begin{cases} s^2 X(s) = b \cdot s X(s) + k \cdot X(s) + X_3(s) \\ s X_3(s) = \alpha(u - X(s)) - \beta x_3(s) \end{cases}$$

$$\Rightarrow X_3(s) = \frac{\alpha(u - X(s))}{s+\beta}$$

ZHANPENG HE

$$X(s) = \frac{1}{s^2 - bs - k} \cdot X_3(s)$$

$\Rightarrow$ Combining them for the whole system:

$$X(s) = \frac{\alpha(u - X(s))}{s+\beta} \cdot \frac{1}{s^2 - b - k}$$

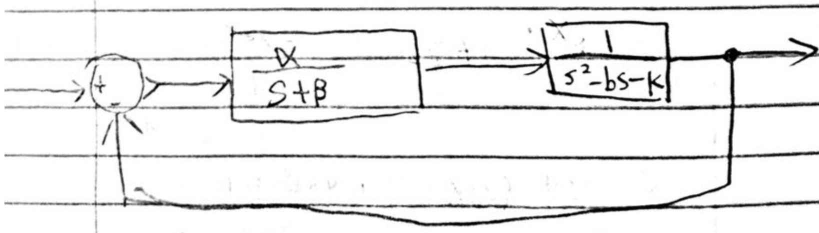
$$= \frac{\alpha u - \alpha X(s)}{(s+\beta)\cdot(s^2 - b - k)}$$

$$\left((s+\beta)(s^2-b-k) + \alpha\right)\cdot X(s) = \alpha u$$

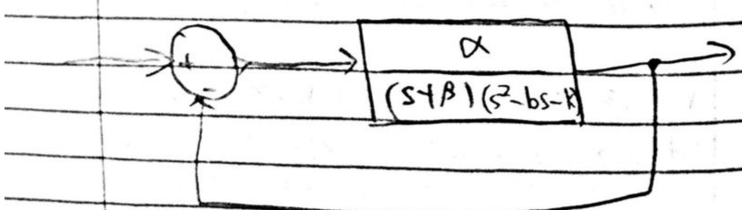
$$X(s) = \frac{\alpha}{((s+\beta)(s^2 - bs - k) + \alpha)} \cdot u$$

Block diagrams:

individually:

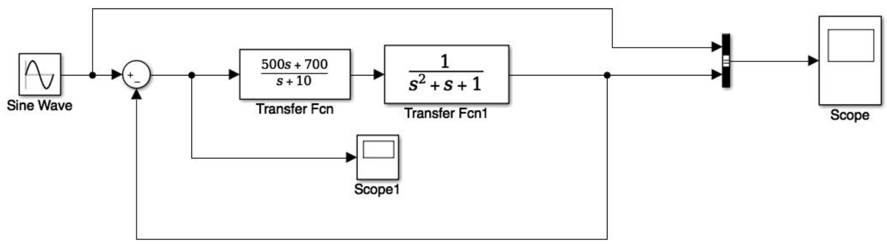


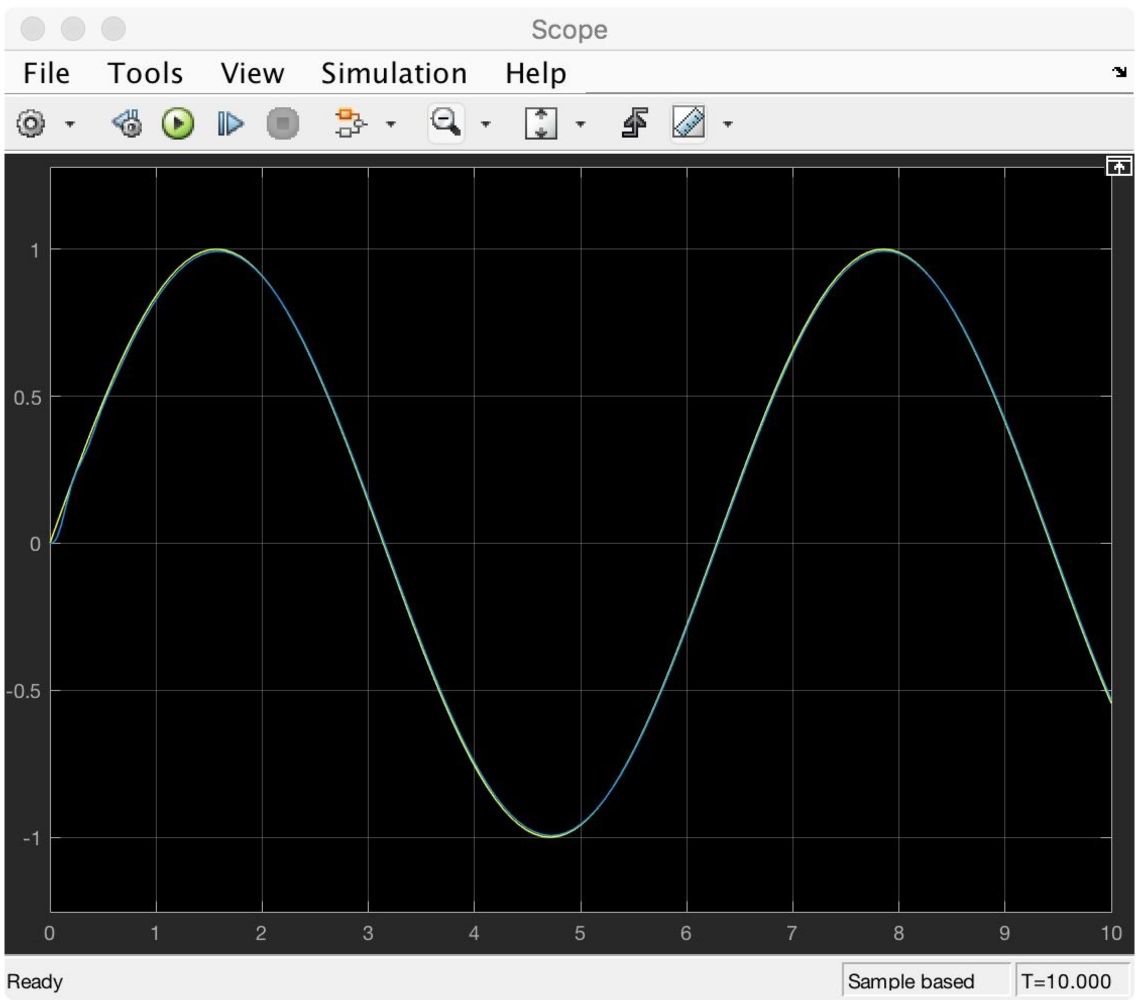
Combined

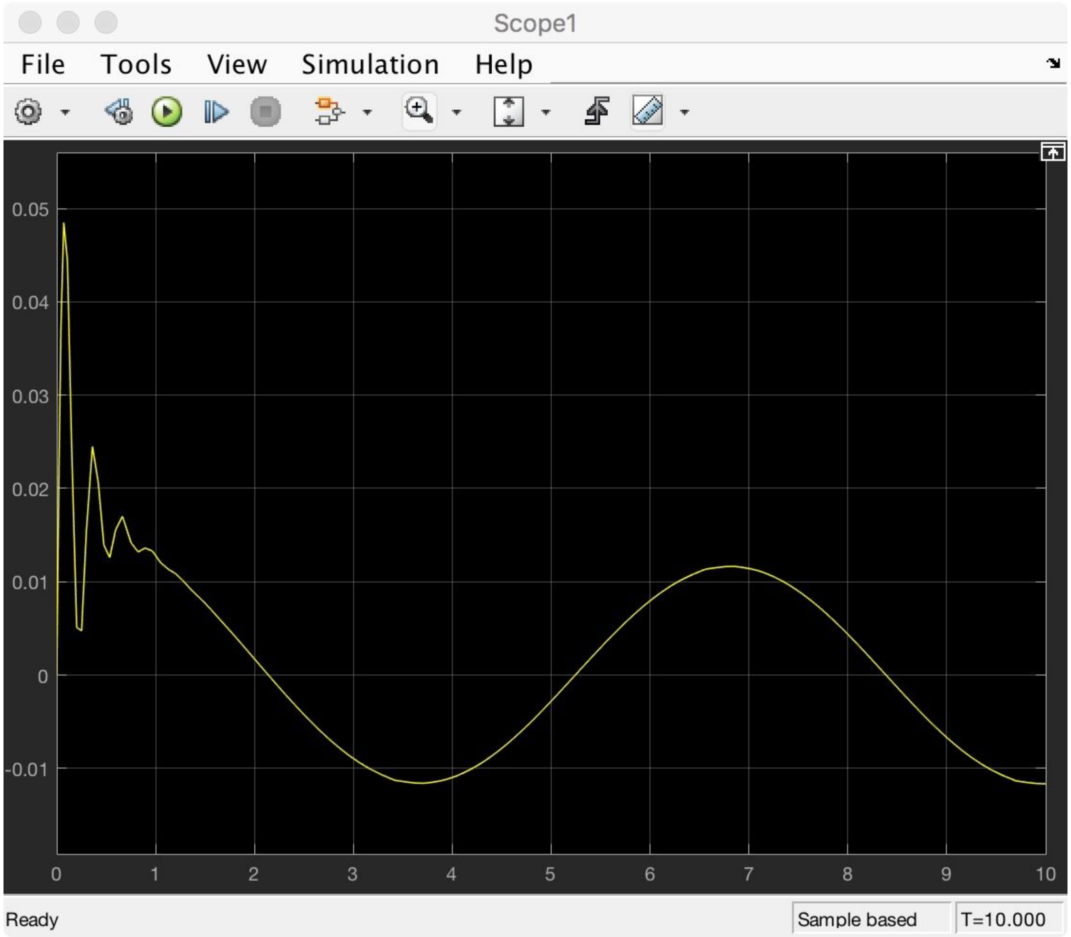


ZHANPENG HE

2.c

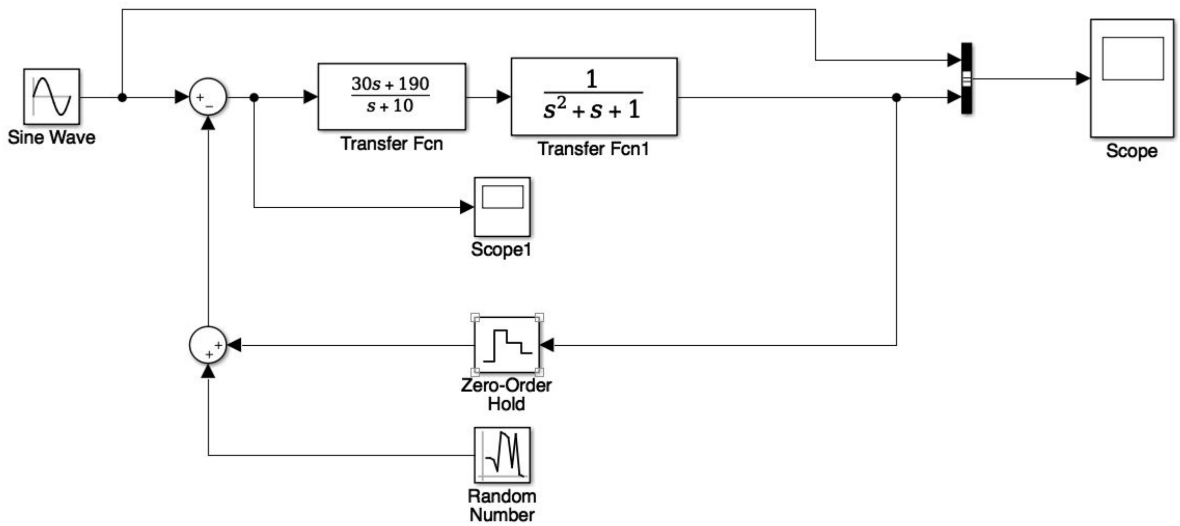




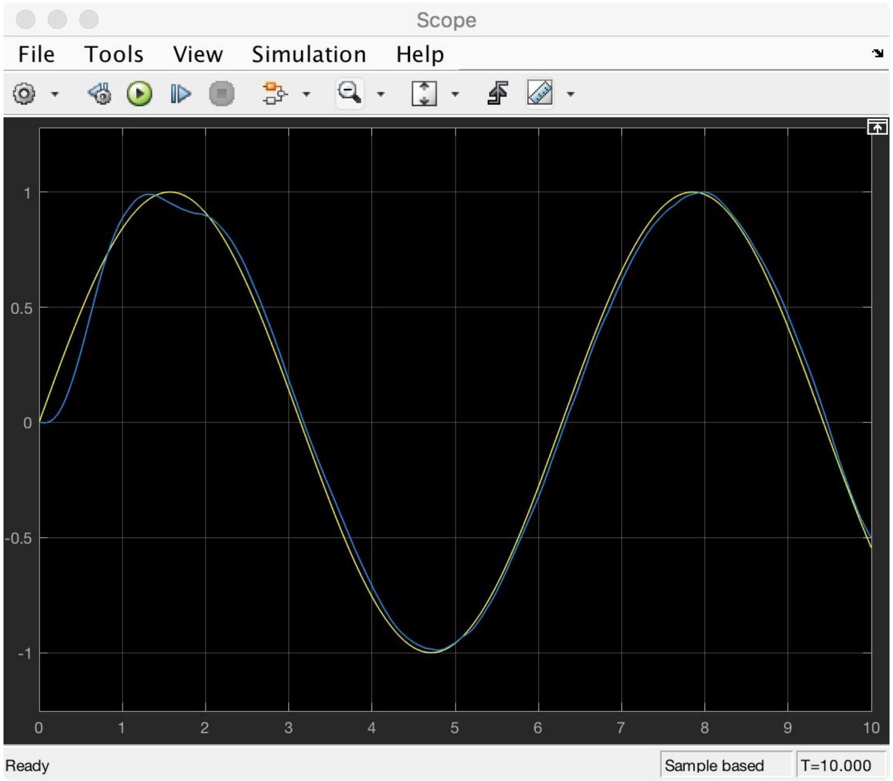

Scope1
File
Tools
View
Simulation
Help
0.05
0.04
0.03
0.02
0.01
0
-0.01
0
1
2
3
4
5
6
7
8
9
10
Ready
Sample based
T=10.000

ZHANPENG HE

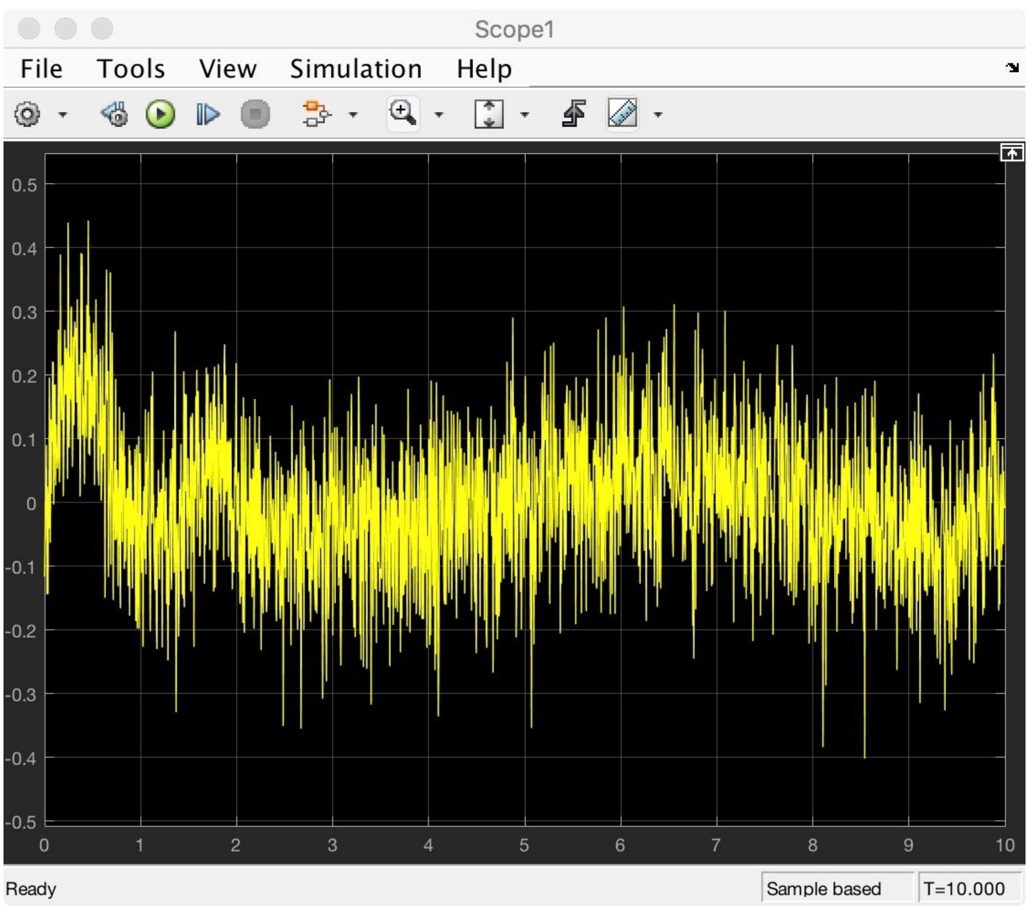
2.d



ZHANPENG HE



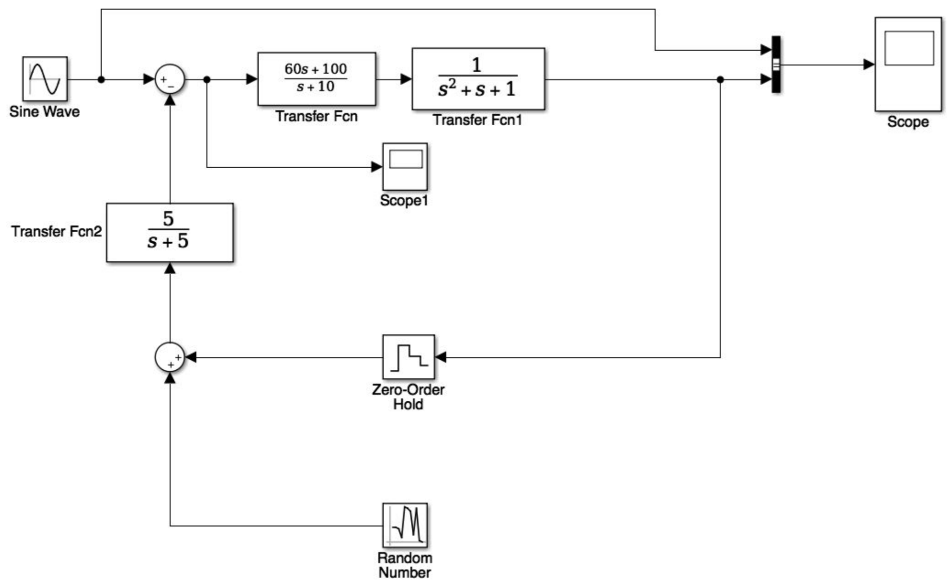
Scope
File Tools View Simulation Help
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
Ready
Sample based
T=10.000



Scope1
File Tools View Simulation Help
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0 1 2 3 4 5 6 7 8 9 10
Ready
Sample based
T=10.000

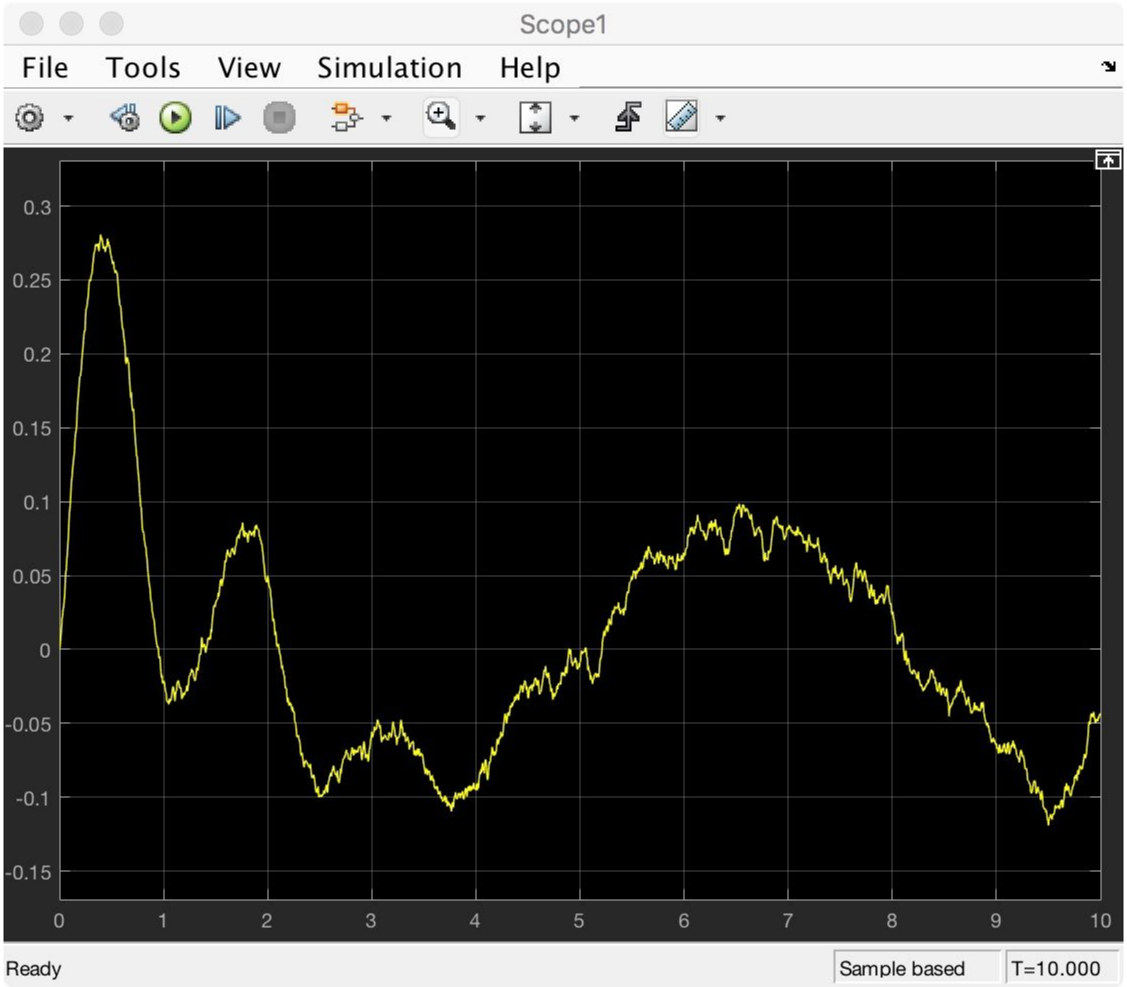
With noise the error is jumping frequently and not smooth. It comes from the random noise on the feedback.

2.




Scope
File Tools View Simulation Help
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
Ready
Sample based
T=10.000

ZHANPENG HE



Success: it smooth out the error to a certain degree while keeping the error small(<-0.15<err<0.3). While setting smaller lambda, i get smoother error but get a larger error.

Transfer function:

xd = lambda*(u-x)
s*x(s) = lambda*(u(s)-x(s))
x(s) =lambda/(s+lambda)*u(s)