Process Control - Project Task 3 & 4
Control of a Multivariable Process

Submitted by
Md. Yasin Arafat

Newell and Lee Evaporator

Evaporator Equation's Matlab Code

```
function dxdt = evapmod(t, x, u)
% Inputs
F1 = u(1);          % [kg/sec];
F2 = u(2);          % [kg/sec]
P100 = u(3);         % [kPa]
F200 = u(4);         % [kg/sec]
T1 = u(5);          % [?C]
XF1 = u(6);          % [%]
F3 = u(7);          % [kg/sec]
T200 = u(8);         % [?C]
% Outputs
X2 = x(1);
P2 = x(2);
L2 = x(3);
% Parameters
C = 4; Cp = 0.07;
lam = 38.5; lams = 36.6;  rhoA = 20;
M = 20; UA2 = 6.84;
% Algebraic equations
% Evaporator and steam jacket
T2 = 0.5616*P2 + 0.3126*X2 + 48.43;
```

```
T3 = 0.507*P2 + 55;
T100 = 0.1538*P100 + 90;
Q100 = 0.16*(F1 + F3)*(T100 -T2);
F100 = Q100/lams;
F4 = (Q100 - F1*Cp*(T2 - T1))/lam;
%Condenser
Q200 = UA2*(T3-T200)/(1+UA2/(2*Cp*F200));
T201 = T200 + Q200/(F200*Cp);
F5 = Q200/lam;
%Differential equations
dX2dt = (F1*XF1 - F2*X2)/M;
dP2dt = (F4 - F5)/C;
dL2dt = (F1 - F4 - F2)/rhoA;
%Output
dxdt = [dX2dt dP2dt dL2dt]';
end
```

Matlab Code for S-Function:

```
function [sys,x0] = evapmods(t, x, u, flag)
%          Simulink interface to evapmods.m
%
%          Inputs:    t    - time in [sec].
%          x    - 3 states, The level of the liquid in the separator tank L2.
%                    Concentration of the liquid obtained as the product X2.
%                    The operating pressure P2 in the evaporator.
%          F1=   u(1),            inlet feed flowrate
%          F2 =   u(2);            % [kg/sec]
%          P100 = u(3);            % [kPa]
%          F200 = u(4);            % [kg/sec]
%           T1 = u(5);           % [?C]
```

```
%           XF1 = u(6);           % [%]
%           F3 = u(7);            % [kg/sec]
%          T200 = u(8);
%           Outputs:   sys and x0 as described in the SIMULINK manual.
%          when flag is 0 sys contains sizes and x0 contains
%          initial condition.
%          when flag is 1, sys contains the derivatives,
%          and when flag is 3 sys contains outputs;
%          y(1)    - Concentration of the liquid obtained as the product X2.
%          y(2)    - The operating pressure P2 in the evaporator.
%          y(3)    - The level of the liquid in the separator tank L2.
%           Initialize :  define initial conditions,  X0
%           define system in terms of number of states, inputs etc.
%           e.g. sys = [3, 0, 3, 3, 0, 0];
%           1st array :  number of continuous states
%           2nd array :  number of discrete states
%           3rd array :  number of outputs
%          4th array :  number of inputs
%          5th array :  flag for direct feedthrough
%          6th array :  the number of sample times
 if abs(flag) == 1
  % Return state derivatives.
  sys = evapmod(t,x,u);
elseif abs(flag) == 3
  % Return system outputs.
  sys(1,1,1) = x(1);       %  X2
  sys(2,1,1) = x(2);       %  P2
  sys(3,1,1) = x(3);       %  L2
elseif flag == 0
  % Initialize the system
```

```
  load init_ss.mat
  x0=X0ss ;
  sys = [3, 0, 3, 8 , 0, 0];
else
  sys = [];
end
```

Simulink Model:

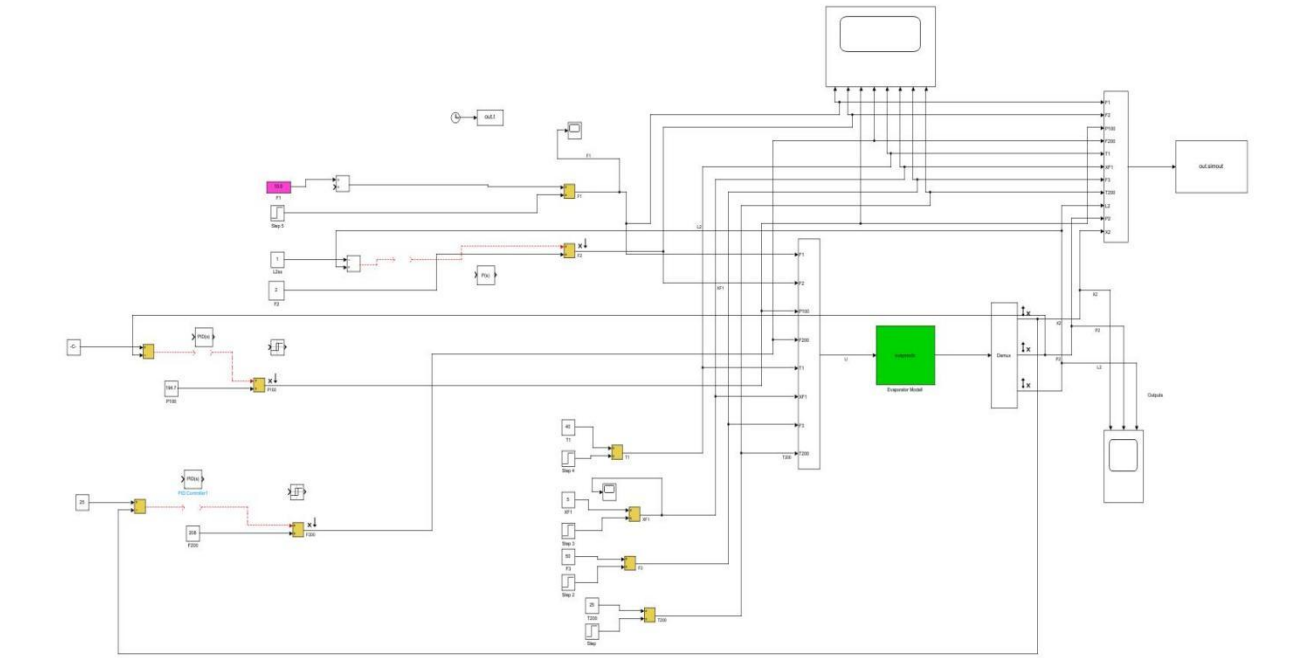

Fig 01: Simulink Model for linearizer (3x3)

RGA Analysis Computation:

STEP 1: Linear system matrices

```
>> [A,B,C,D] = ssdata(linsys1)
A =
   -0.1000         0         0
   -0.0209   -0.0558         0
    0.0042    0.0075         0
```

B =

-1.2500	0	0
0	0.0096	-0.0018
-0.0500	-0.0019	0

C =

1.0000	0	0
0	1.0000	0
0	0	1.0000

D =

0	0	0
0	0	0
0	0	0

System's Eigen Value:

lambda=eig(A)

lambda =

0

-0.0558

-0.1000

STEP 2: G0 for 3 by 3 MIMO system

>>G0= -C*inv(A)*B+D

Warning: Matrix is singular to working precision.

G0 =

NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN

```
>> G0= -C*pinv(A)*B+D
G0 =
   -12.4910    0.0001    0.0000
    4.8395    0.1733   -0.0322
         0         0         0
```

Comment:

```
>> det(A)
ans =
     0
```

Since the determinant is zero, hence the matrix 'A' is singular. The matrix is also not square matrix. So, we have get an error while using command "inv", as 'inv' can be used only for non-singular matrix. Therefore, we use "pinv" as it computes by using singular value decomposition technique.

STEP 3: RGA0 based on G0

```
>> RGA0= G0.*(pinv(G0'))

RGA0 =
    0.9998    0.0002   -0.0000
    0.0002    0.9665    0.0334
         0         0         0
```

Comment: Computation of RGA by using 'pinv(G0)' is not successful. General property of RGA is not fulfilled.

Question: How should one make progress at this stage?

Answer:

As we can see from the Eigen values is that the system is marginally stable. From Step change analysis we see that L2 has previously shown integrating behavior. And from RGA analysis we see that it does not satisfy the general property. Hence by putting a proportional controller to control the uncontrolled behaviour of L2, one can use decentralize control design procedure, and then again recalculate the RGA analysis for remaining two outputs and inputs [1].

Simulink Model for 2x2 Linearzation:

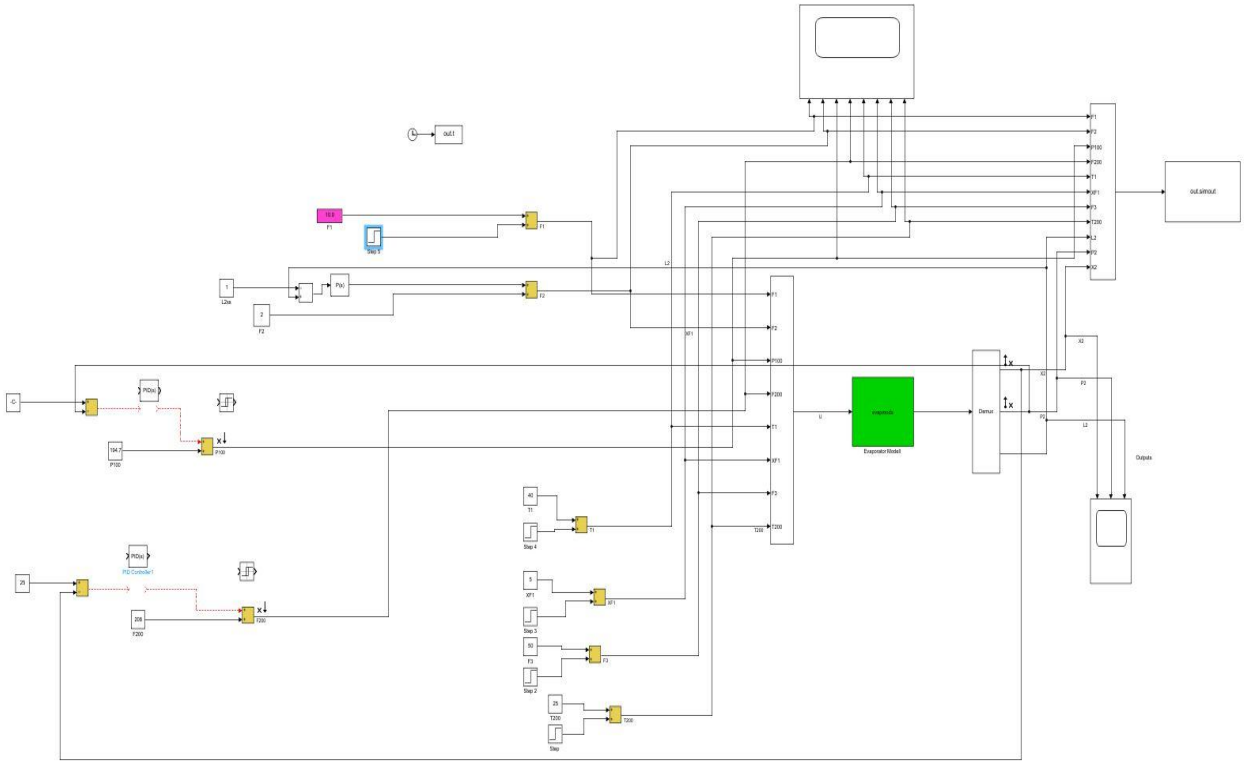

Fig 02: Simulink model for 2x2 model linearization

RGA Analysis after adding proportional controller at input F2:

```
>> [A,B,C,D]=ssdata(linsys1)

A =

   -0.1000         0         -1.2500
   -0.0209    -0.0558         0
    0.0042     0.0075     -0.0500

B =

         0         0
    0.0096    -0.0018
   -0.0019         0

C =

    1.0000         0         0
         0    1.0000         0
```

```
D =
     0     0
     0     0
>> lambda=eig(A)
lambda =
   -0.0893 + 0.0759i
   -0.0893 - 0.0759i
   -0.0273 + 0.0000i
>> G0= -C*inv(A)*B+D
G0 =
    0.1168    0.0459
    0.1281   -0.0500
>> G0= -C*pinv(A)*B+D
G0 =
    0.1168    0.0459
    0.1281   -0.0500
>> det(A)
ans =
  -3.7430e-04
>> RGA0= G0.*(pinv(G0'))
RGA0 =
        0.4982    0.5018
        0.5018    0.4982
```

From the Model Linearizer step response we can see that F200 to X2 and P100 to P2 can be best I/O pairs.

```
>> G0jj=diag(G0)
G0jj =
        0.1168
       -0.0500
```

```
>> NI=det(G0)/(G0jj(1)*G0jj(2))
   NI =
        2.0071
```

Method 1: PID controller incorporating and tuning (Ziegler-Nichols open loop method)

Process Transfer Functions:

```
>> G = tf(linsys1)
G =

 From input "P100" to output...
             0.002397 s + 4.371e-05
  X2:  ---------------------------------------
       s^3 + 0.2058 s^2 + 0.0186 s + 0.0003743

        0.009588 s^2 + 0.001438 s + 4.794e-05
  P2:  ---------------------------------------
       s^3 + 0.2058 s^2 + 0.0186 s + 0.0003743

 From input "F200" to output...
                   1.717e-05
  X2:  ---------------------------------------
       s^3 + 0.2058 s^2 + 0.0186 s + 0.0003743

       -0.001829 s^2 - 0.0002743 s - 1.87e-05
  P2:  ---------------------------------------
       s^3 + 0.2058 s^2 + 0.0186 s + 0.0003743

Name: Linearization at model initial condition
Continuous-time transfer function.
```

Output Step Response Curve:

```
>> figure(1)
```

step(G)

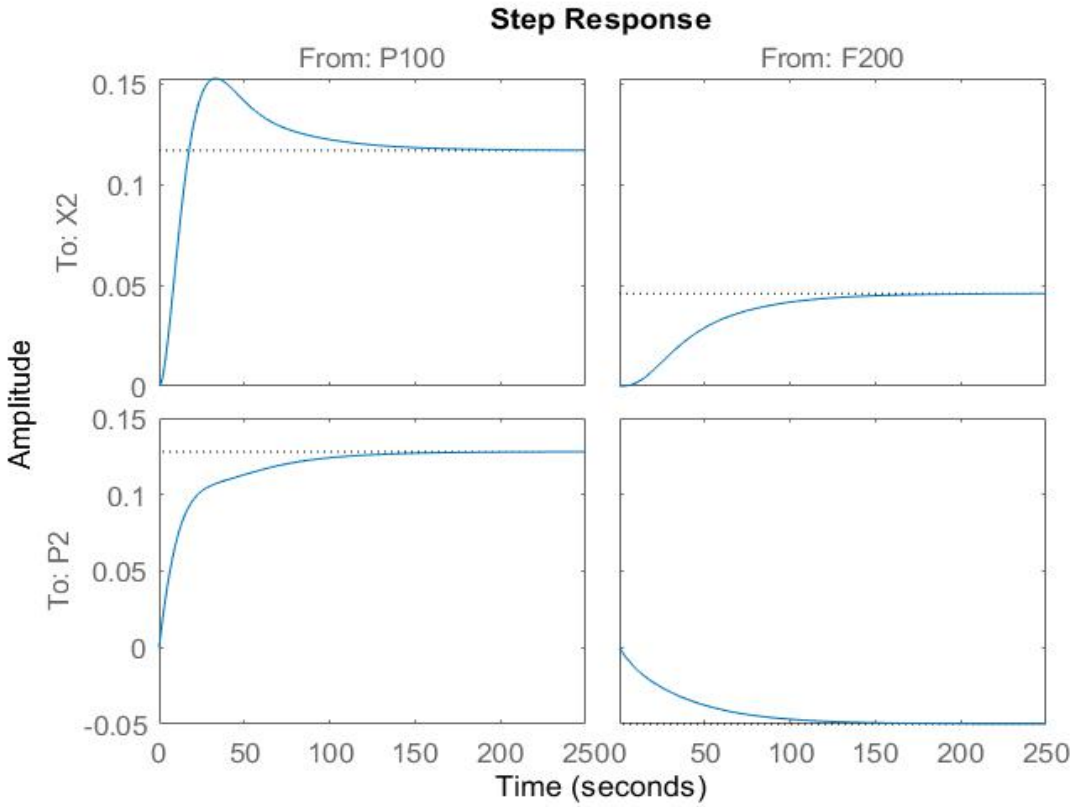

Fig 03: Step response curve

Storing the output data in variable 'y' and time 't' vector:

>> [y,t] = step(G)

Plotting the step output response again:

>> figure(2)

plot(t,y(:,1,2),'r')

legend('X_2')

xlabel('time (min)')

ylabel('X2')

title('Step Change in F200 to Change in X2')

figure(3)

plot(t,y(:,2,1),'g')

legend('P_2')

xlabel('time (min)')

ylabel('P2')

title('Step Change in P100 to Change in P2')

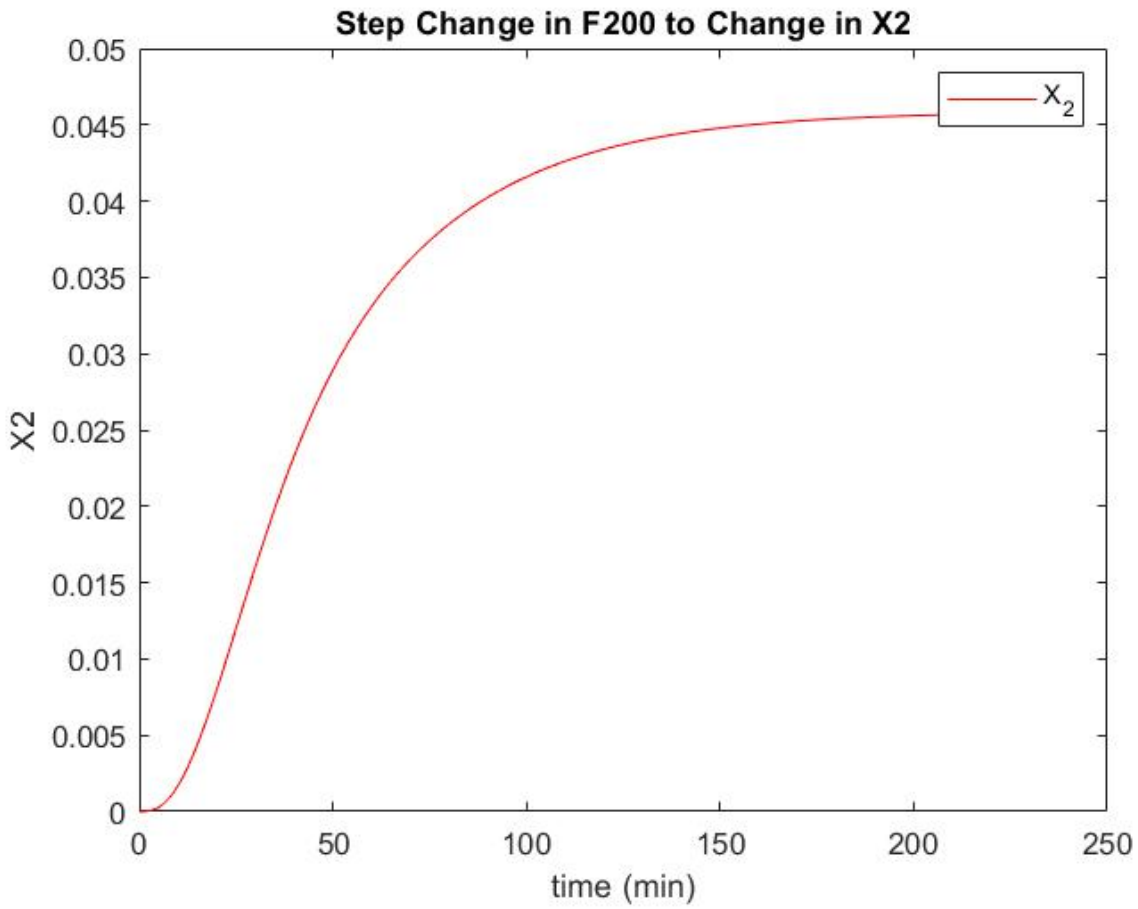

Fig 02: Step change of X2 due to F200

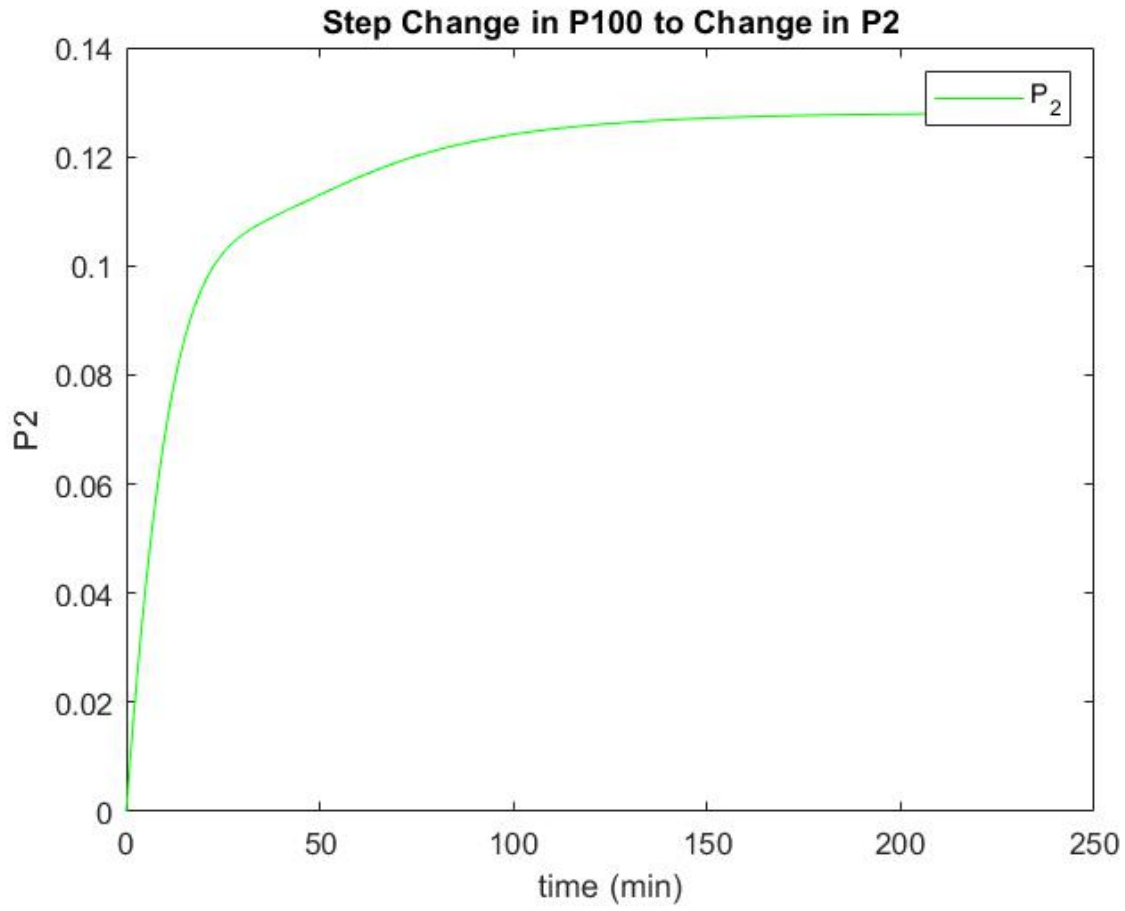

Fig 03: Step change of P2 due to P100

Slope at each point:

>> slope1 = gradient(y(:,1,2),t)

>> slope2 = gradient(y(:,2,1),t)

XY values at the point of inflection:

>> [tslope1,idx1] = max(slope1)

tIP1 = t(idx1)

yIP1 = y(idx1,1,2)

[tslope2,idx2] = max(slope2)

tIP2 = t(idx2)

yIP2 = y(idx2,2,1)

tslope1 = 8.1683e-04

idx1 = 26

tIP1 = 25.7944

yIP1 = 0.0126

tslope2 = 0.0093

idx2 = 1

tIP2 = 0

yIP2 = 0

Tangent lines at the inflection point:

```
>> yTangentLine1 = tslope1*(t-tIP1) + yIP1
yTangentLine2 = tslope2*(t-tIP2) + yIP2
```

Inflection points and tangent lines at the output response curve:

```
>>  figure(2)
plot(t,y(:,1,2),'r')
xlabel('time (min)')
ylabel('X2')
title('Step Change in F200 to Change in X2')
hold on
plot(tIP1,yIP1,'r*'),grid on
plot(t,yTangentLine1,'b')
legend('X_2','Inflation Point','Tangent')
hold off
figure(3)
plot(t,y(:,2,1),'g')
xlabel('time (min)')
ylabel('P2')
title('Step Change in P100 to Change in P2')
hold on
plot(tIP2,yIP2,'g*'),grid on
```

```
plot(t,yTangentLine2,'b')
legend('P_2','Inflation Point','Tangent')
hold off
```

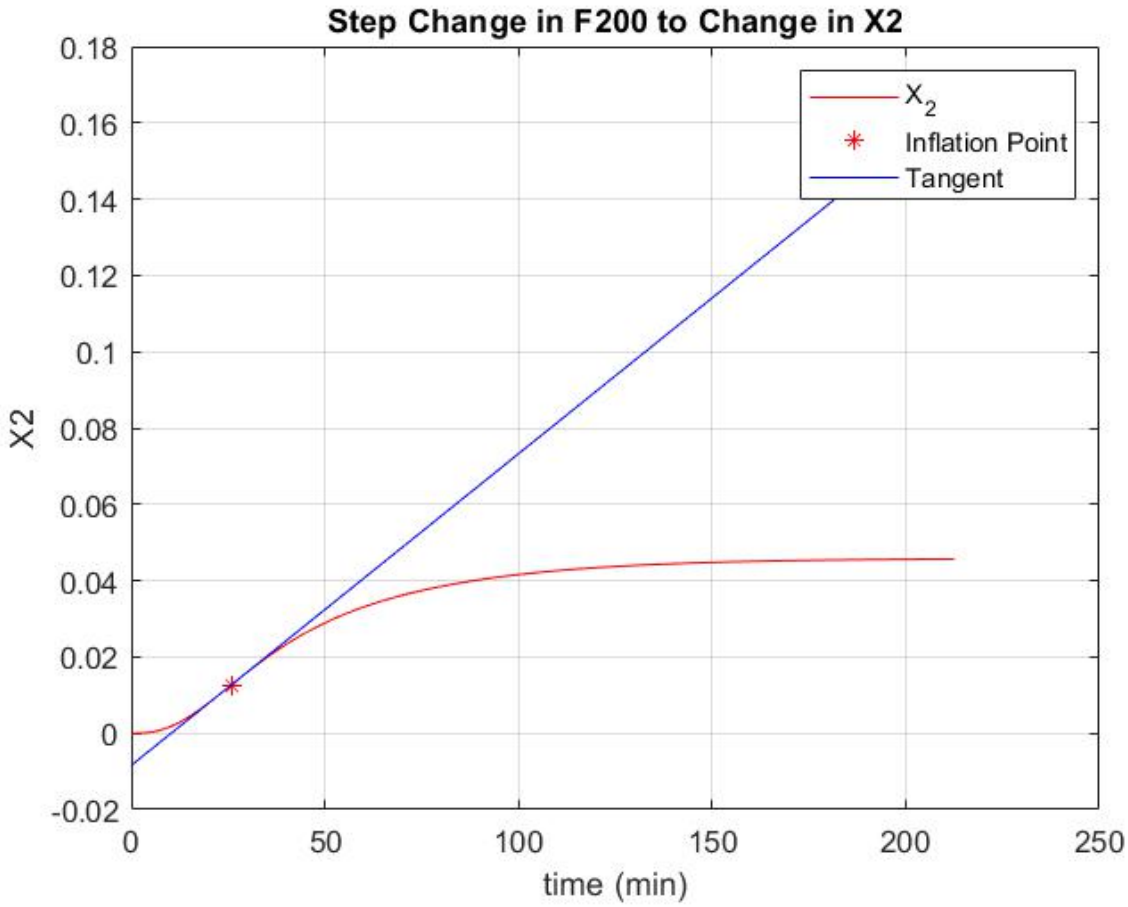

Fig 04: Inflation point at the step response of X2

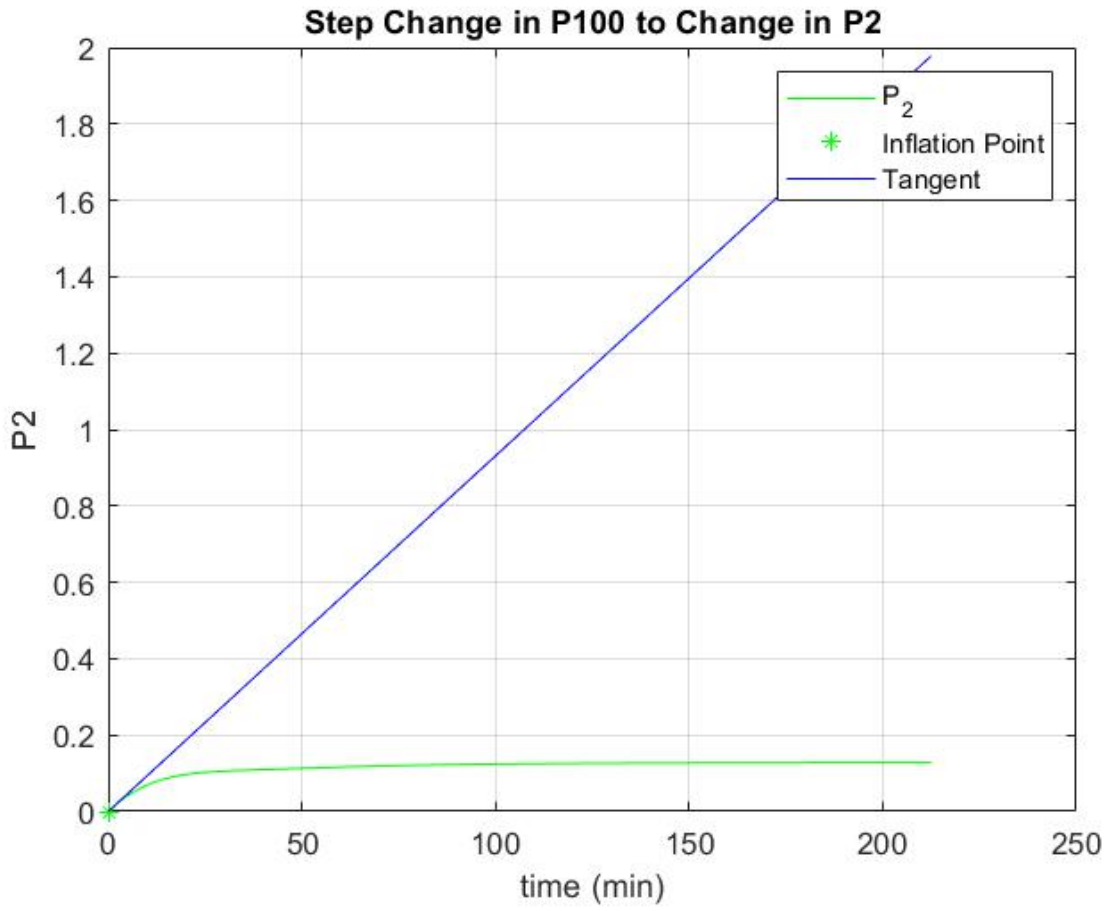

Fig 05: Inflation point at the step response of P2

Estimated parameters of the transfer function (FOPTD) model:

>> Td1 = tIP1-(yIP1/tslope1)

Td2 = tIP2-(yIP2/tslope2)

Td1 = 10.3182

Td2 = 0

Here, Td2 is zero, as output response curve of P100 to P2 does not have time delay.

That will lead transfer function P100 to P2 to a static value. As per the Zieglar-Nichols open loop controller assumption, a measurement time delay should be taken.

Assume:

>> Td2 = 1

Calculating time constant, Tau:

```
>> Tau1 = y(end,1,2)/tslope1
Tau2 = y(end,2,1)/tslope2

Tau1 = 55.9295
Tau2 = 13.7390
```

Process gain, K:

```
>> K1 = y(end,1,2)/1
K2 = y(end,2,1)/1

K1 =  0.0457
K2 =  0.1279
```

Showing time delay on the graph:

```
>> figure(2)
plot(t,y(:,1,2),'r')
xlabel('time (min)')
ylabel('X2')
title('Step Change in F200 to Change in X2')
hold on
plot(Td1,0,'g.','MarkerSize',15)
legend('X_2','Time Delay')
hold off
figure(3)
plot(t,y(:,2,1),'g')
xlabel('time (min)')
ylabel('P2')
title('Step Change in P100 to Change in P2')
hold on
plot(Td2,0,'r.','MarkerSize',15)
```

```
legend('P_2','Time Delay')
hold off
```

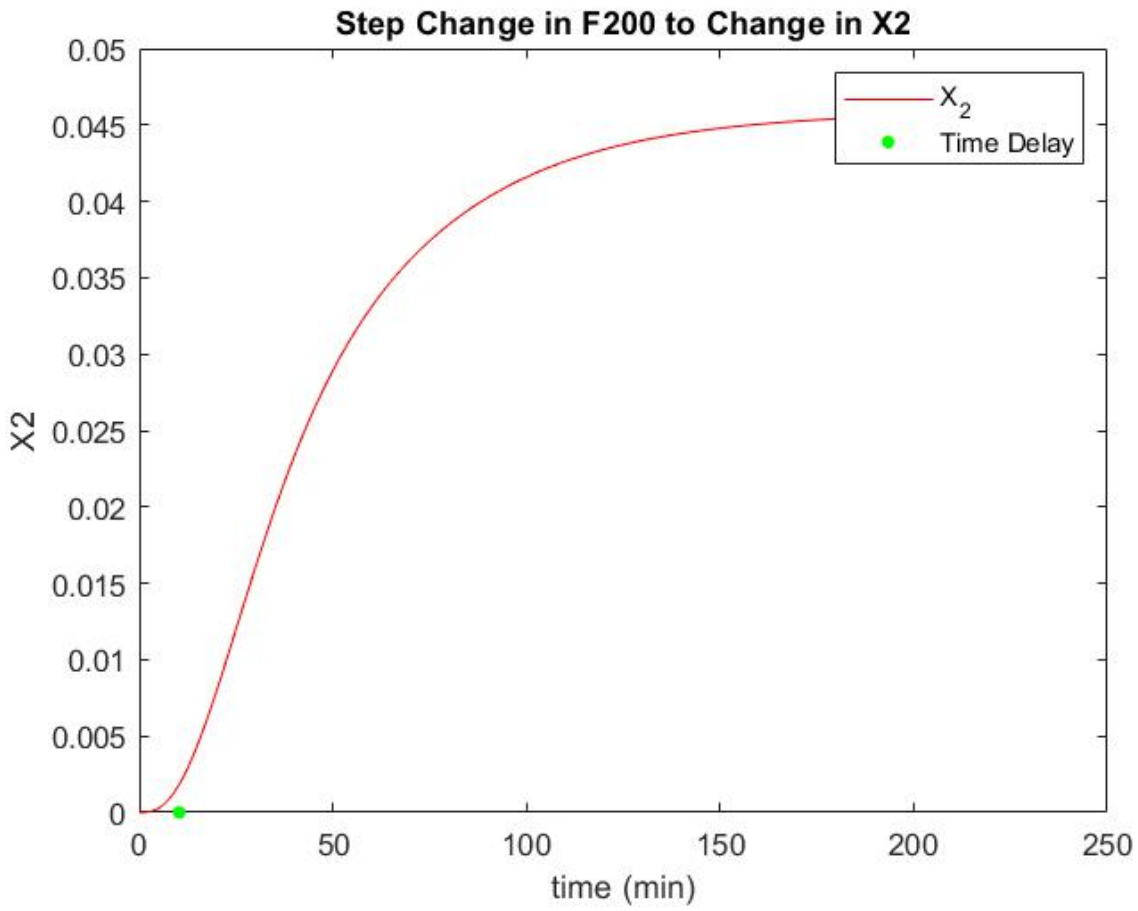

Fig 06: X2 step response with time delay indication

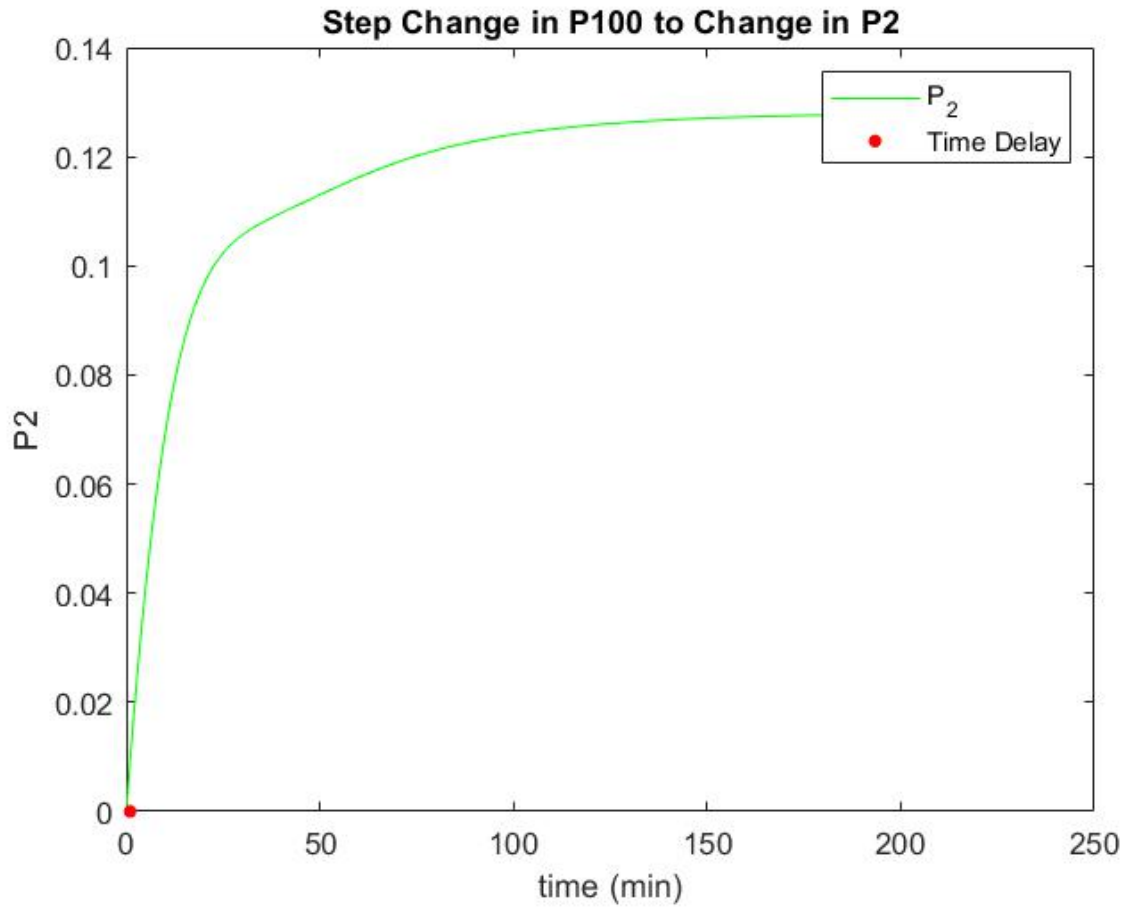

Fig 07: P2 step response with time delay indication

Approximated transfer functions and their step responses:

```
>> G12 = tf(K1,[Tau1],'InputDelay',Td1)
[ynew1,tnew] = step(G12);
figure(4)
plot(t,y(:,1,2),'r')
xlabel('time (min)')
ylabel('X2')
title('Step Change in F200 to Change in X2')
hold on
plot(tIP1,yIP1,'r*'),grid on
plot(t,yTangentLine1,'b')
plot(Td1,0,'g.','MarkerSize',15)
```

```
plot(tnew,ynew1,'r--')
legend('Actual Process','InflectionPoint','Tangent at IP','Dead Time','Approximated FODT')
hold off
G21 = tf(K2,[Tau2],'InputDelay',Td2)
[ynew2,tnew] = step(G21);
figure(5)
plot(t,y(:,2,1),'g')
xlabel('time (min)')
ylabel('P2')
title('Step Change in P100 to Change in P2')
hold on
plot(tIP2,yIP2,'g*'),grid on
plot(t,yTangentLine2,'b')
plot(Td2,0,'r.','MarkerSize',15)
plot(tnew,ynew2,'g--')
legend('Actual Process','InflectionPoint','Tangent at IP','Dead Time','Approximated FODT')
hold off

G12 =
  exp(-10.3*s) * (0.0008168)
Continuous-time transfer function.
G21 =
  exp(-1*s) * (0.009308)
 Continuous-time transfer function.
```

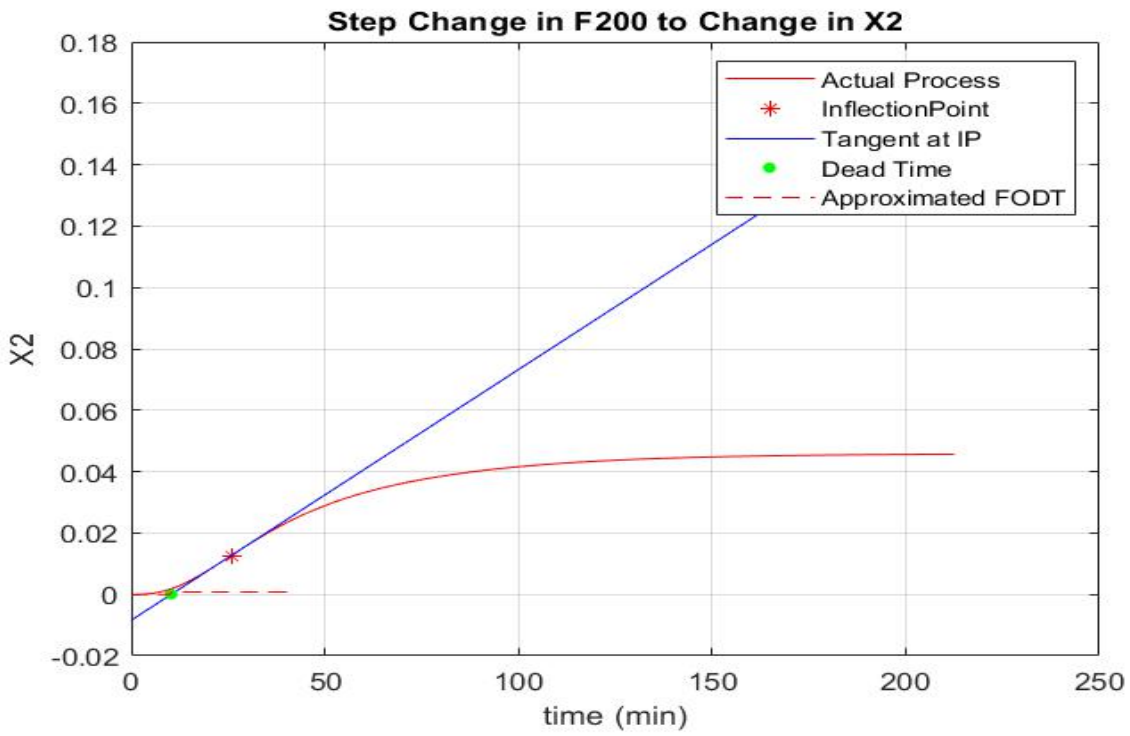

Fig 08: Step response of X2 (approximated)

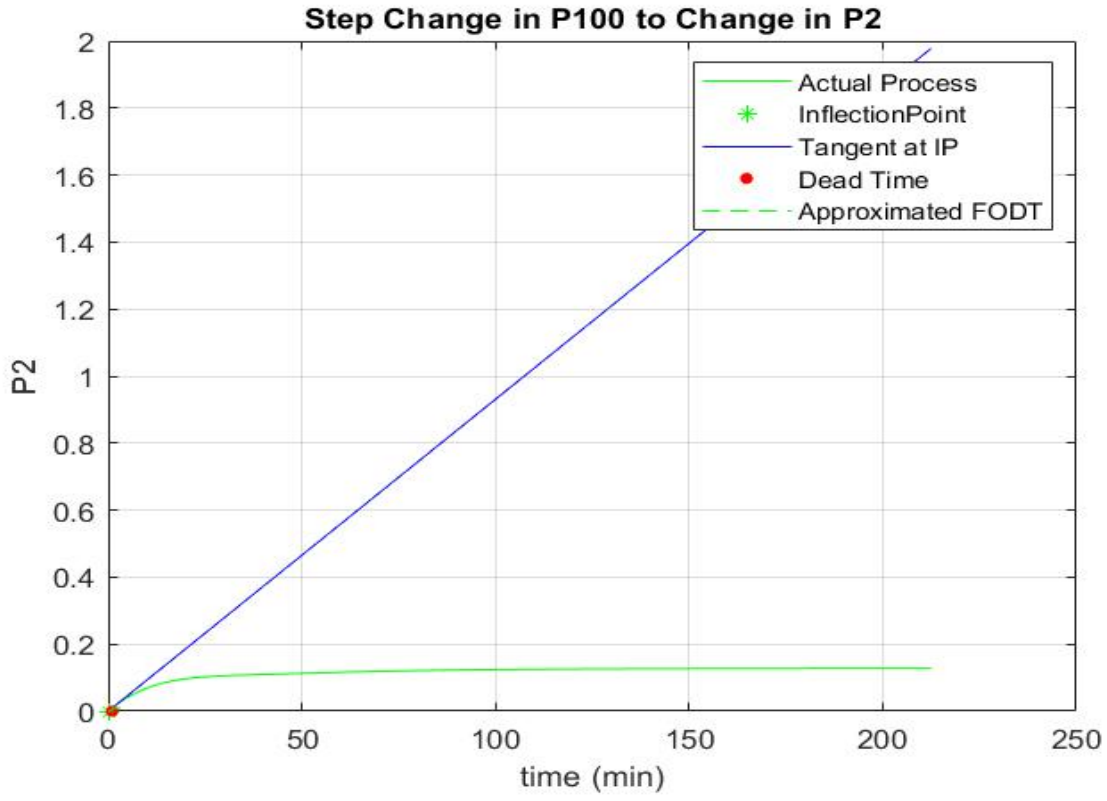

Fig 09: Step response P2 (approximated)

Calculating PID Controller Parameters:

Proportional gain, Kp1 and Kp2

>> Kp1 = 1.2 * Tau1/(K1*Td1)

Kp2 = 1.2 * Tau2/(K2*Td2)

Kp1 = 142.3780

Kp2 = 128.9154

Integral Time TI1 and TI2

>> TI1 = 2*Td1

TI2 = 2*Td2

TI1 = 20.6365

TI2 = 2

Derivative Time TD1 and TD2

>> TD1 = 0.5*Td1

TD2 = 0.5*Td2

TD1 = 5.1591

TD2 = 0.5000

PID controller used in time domain is as follows [2]:

$$
\begin{aligned}
u(t) &= K_P\left(e(t) + \frac{K_I}{K_P}\int e(t)dt + \frac{K_D}{K_P}\frac{de(t)}{dt}\right) \\
&= K_P\left(e(t) + \frac{1}{T_I}\int e(t)dt + T_D\frac{de(t)}{dt}\right) \\
&\text{with: } T_I = K_P/K_I \text{ (integral time constant)}, \\
&\quad\ T_D = K_D/K_P \text{ (derivative time constant)}
\end{aligned}
$$

Hence,

>> KI1=Kp1/TI1

KI2=Kp2/TI2

Kd1=Kp1*TD1

Kd2=Kp2*TD2

KI1 = 6.8993

KI2 = 64.4577

Kd1 = 734.5442

Kd2 = 64.4577

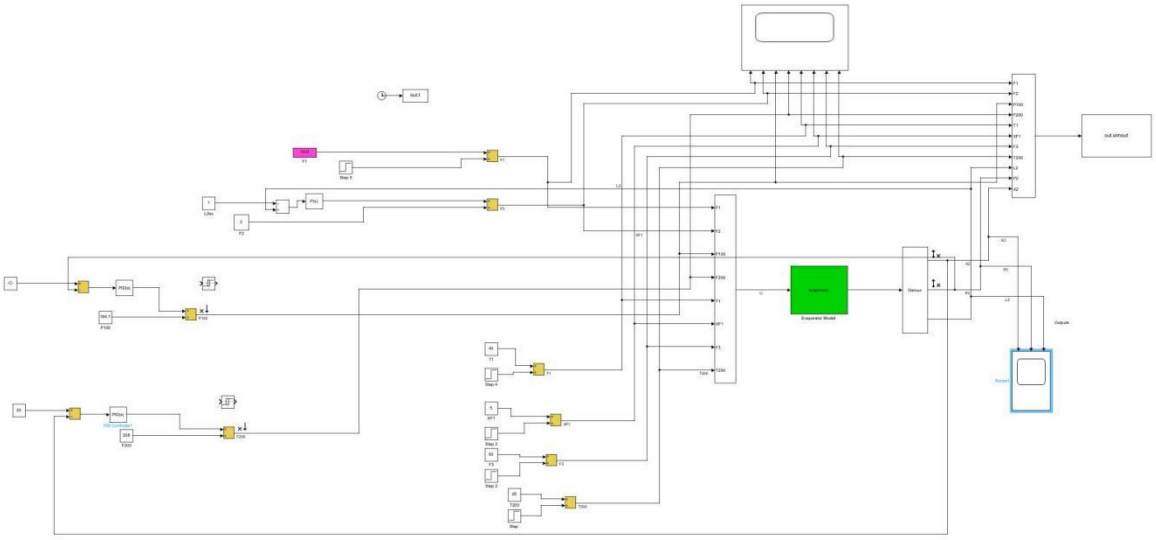

Fig 10: Simulink Model after adding PID controller

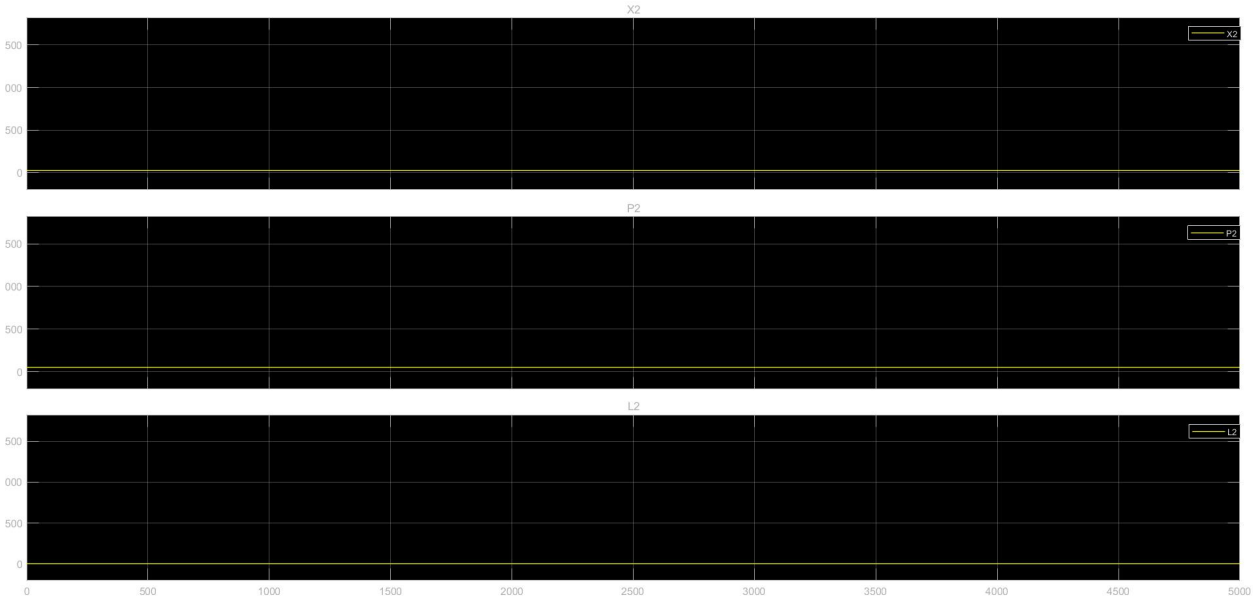

Fig 11: Output step response after adding PID controller without any disturbance or set point tracking

Method 2: Relay Feedback as PID tuning method in Simulink

This method is used in real plant. In general, 'error signal' is larger than the measurement noise. This measurement noise usually larger then 'eps' value [2].

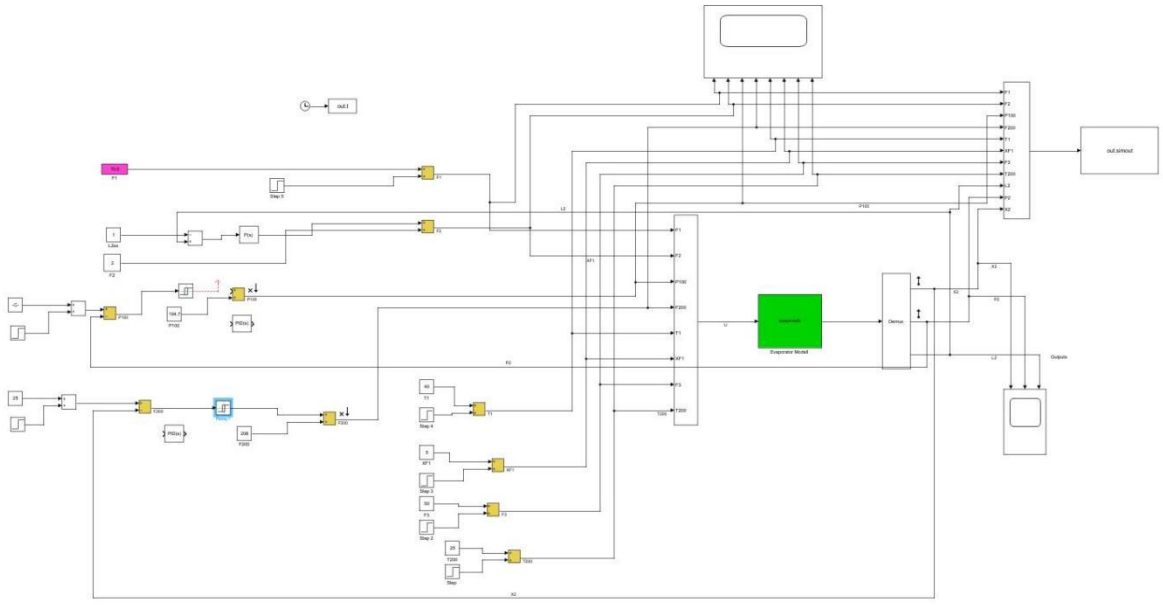

Fig 12: Simulink model using Relay Feedback 1 active

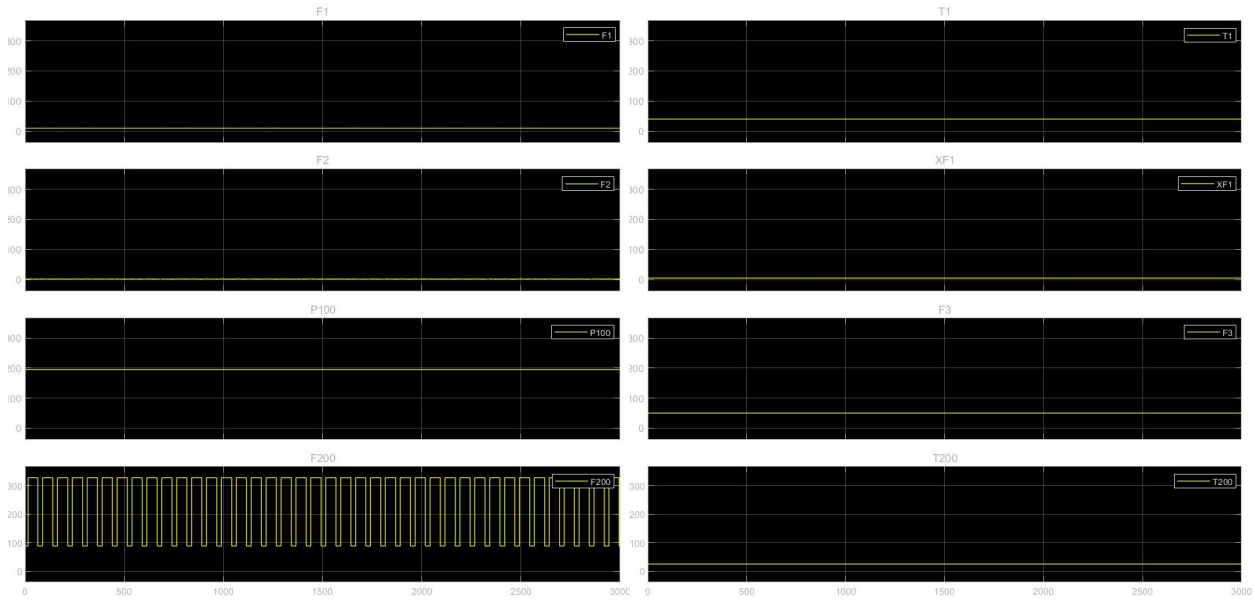

Fig 13: Input response of RF_1 controller connected

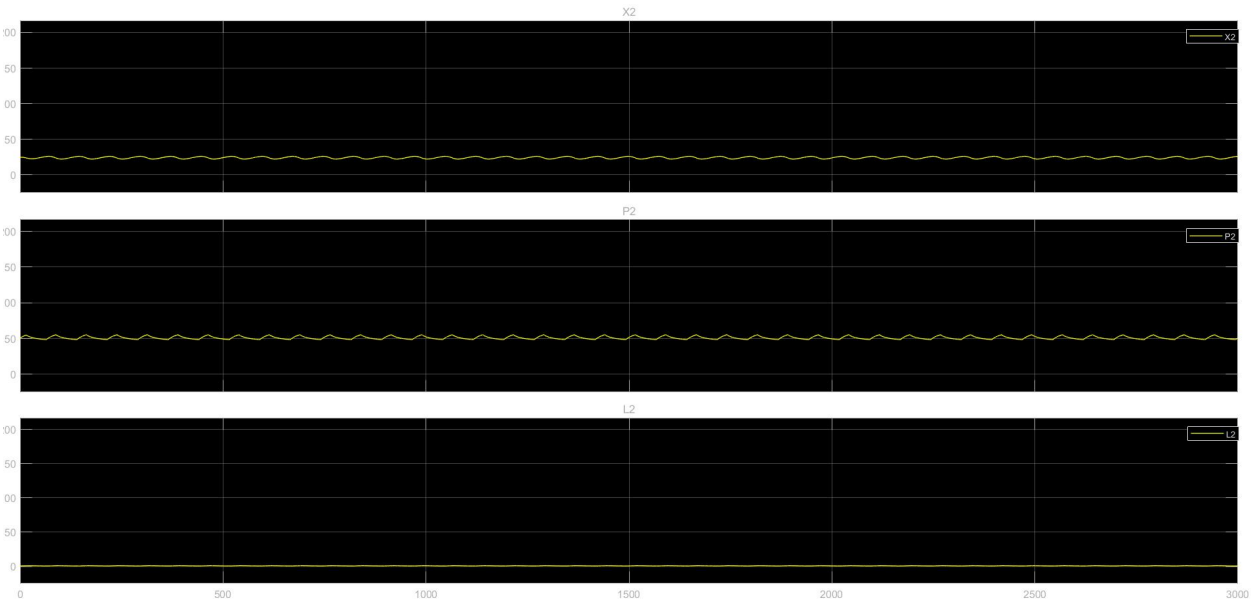

Fig 14: Output response with RF_1 controller connected

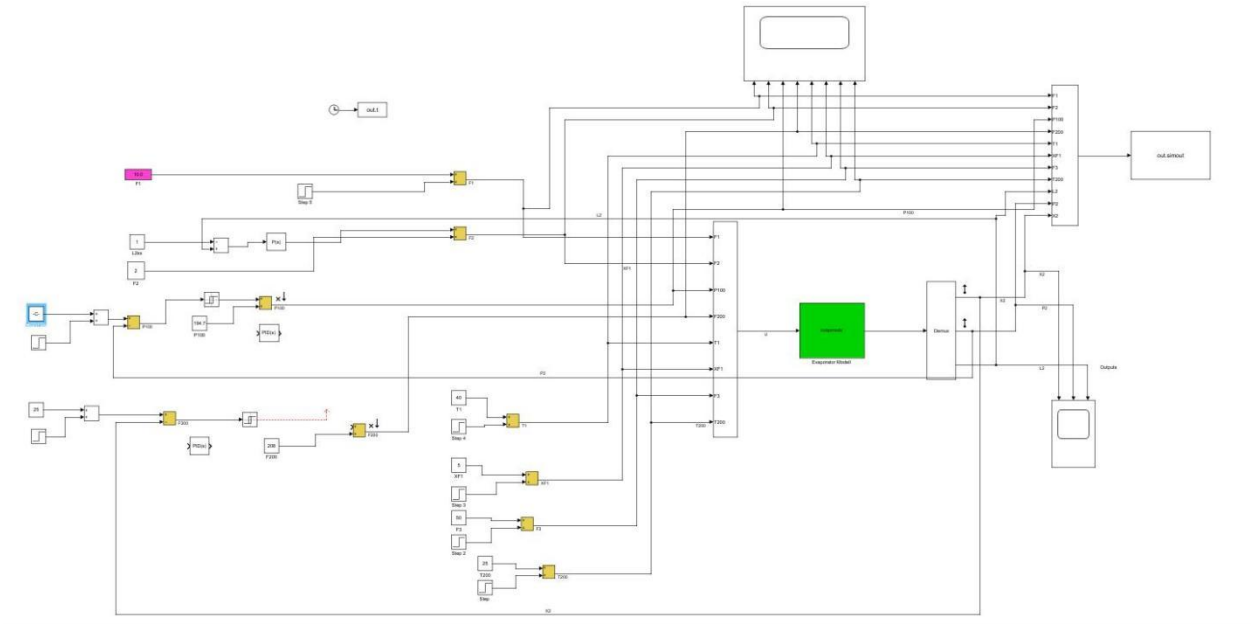

Fig 15: Simulink model of RF 2 controller active

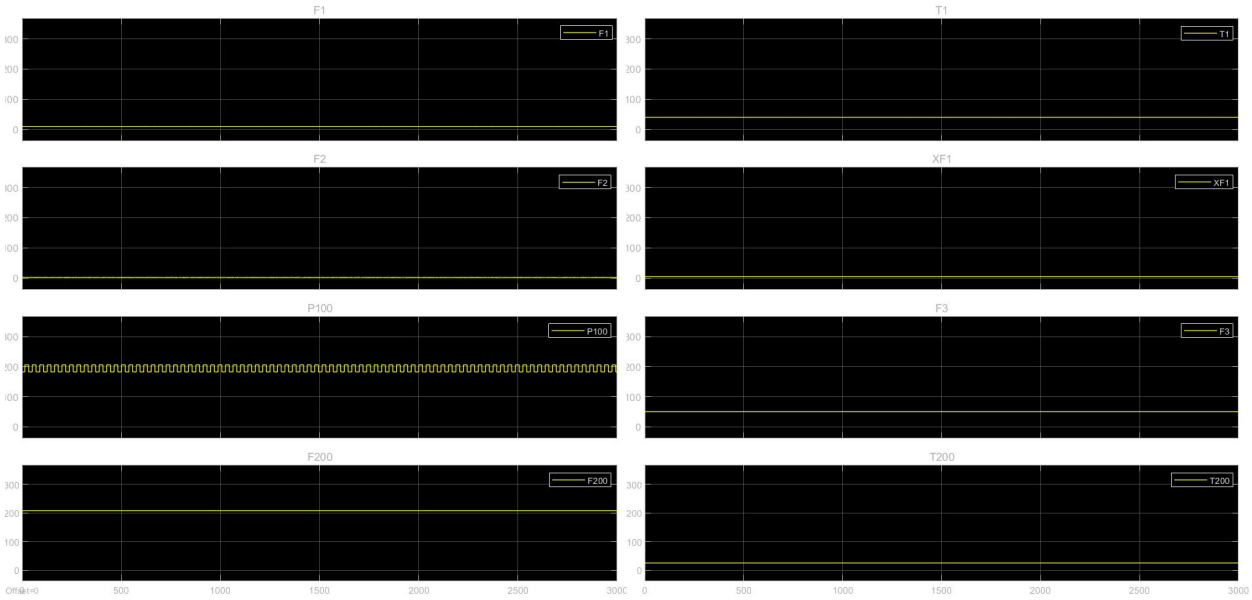

Fig 16: Input while RF_2 controller connected

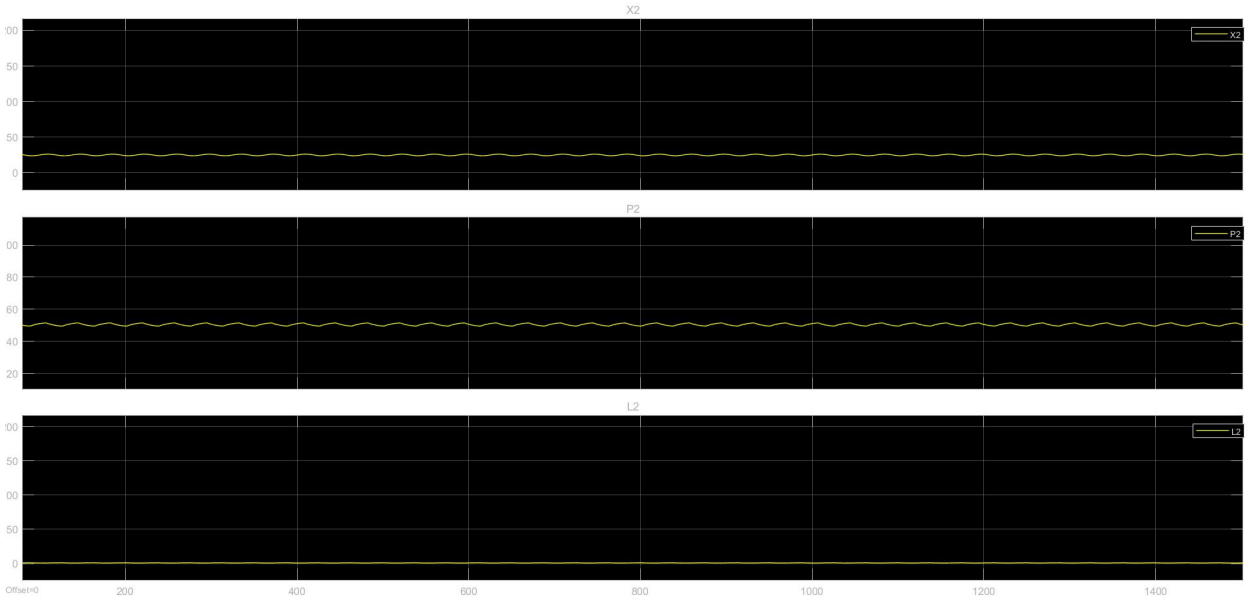

Fig 17: Output while RF_2 controller connected

Pu and Ku calculation for I/O pairs:

F200 to X2

Pu = 58.978 s

a = 3.6447

h = 120

Ku_1 = $\frac{4h}{\pi a}$ = 5.534

PID Controller values:

Kp = 0.6*Ku = 3.3204

TI = 0.5*Pu = 35.573

Td = 0.125*Pu = 8.89325

P100 to P2

Pu = 58.978 s

a = 1.9

h = 10

Ku_2 = $\frac{4h}{\pi a}$ = 12.739

PID Controller values:

Kp = 0.6*Ku = 7.643

TI = 0.5*Pu = 32.617

Td = 0.125*Pu = 8.154

Evaluation the performance of designed controller:

Disturbance rejection scenarios:

I) +10% step change in the feed flowrate F1 to be applied at t=10 min

Step time= 10

Initial value=0

Final value= 10*0.1

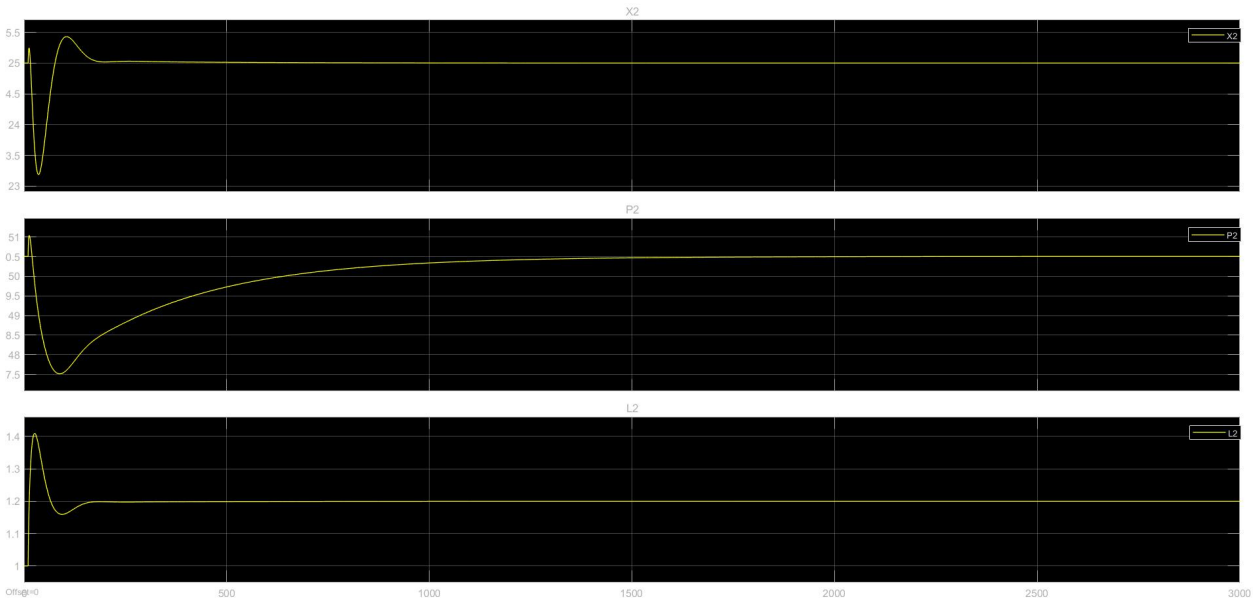

Fig 18: Disturbance rejection with PID controller

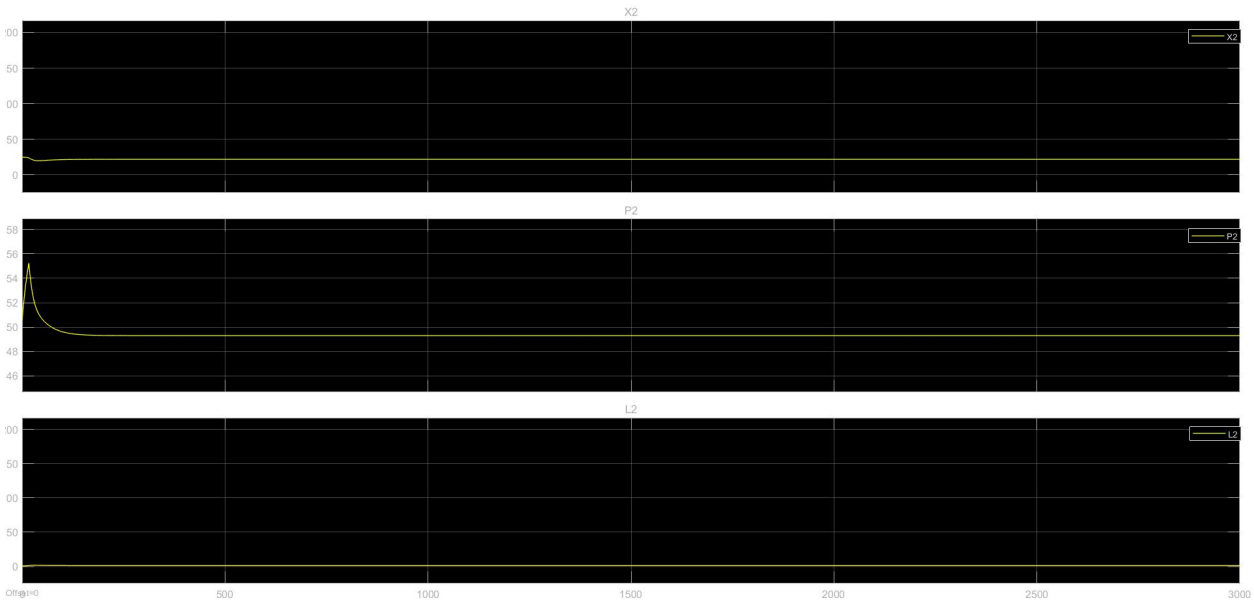

Fig 19: Disturbance rejection while RF_1 controller connected

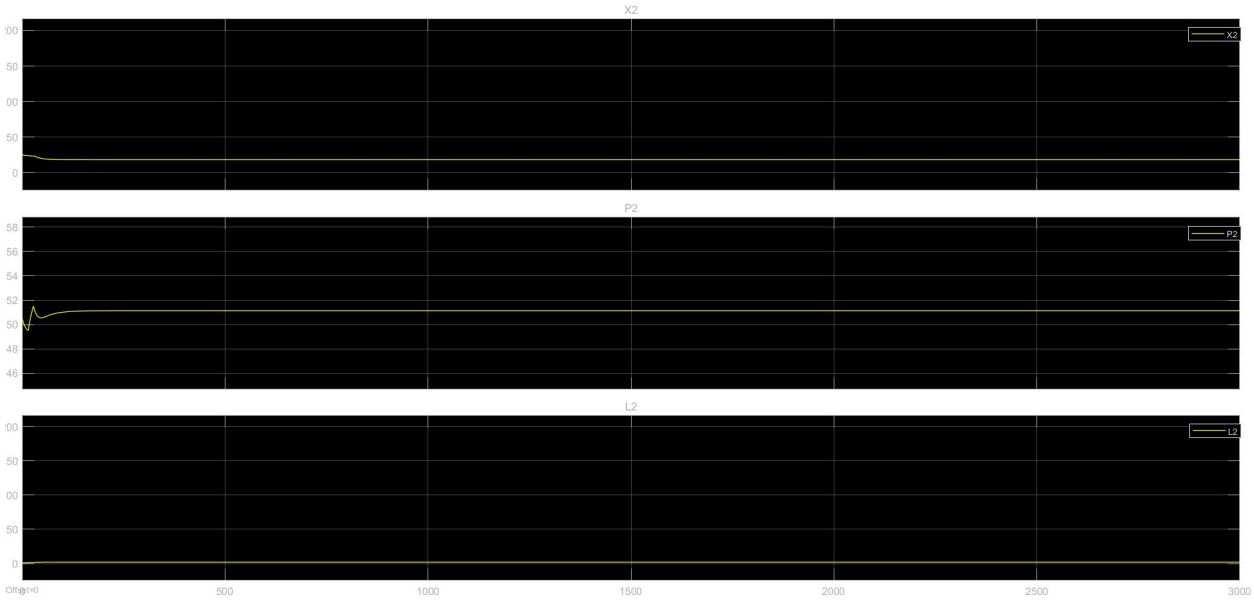

Fig 20: Disturbance rejection while RF_2 controller connected

II) +5% step change in the feed flowrate F1 to be applied at t=10 min. And -5% step change in the feed composition XF1 at t=300 min

For F1:	For XF1:
Step time= 10	Step time= 300
Initial value=0	Initial value=0
Final value= 0.5	Final value= -0.25

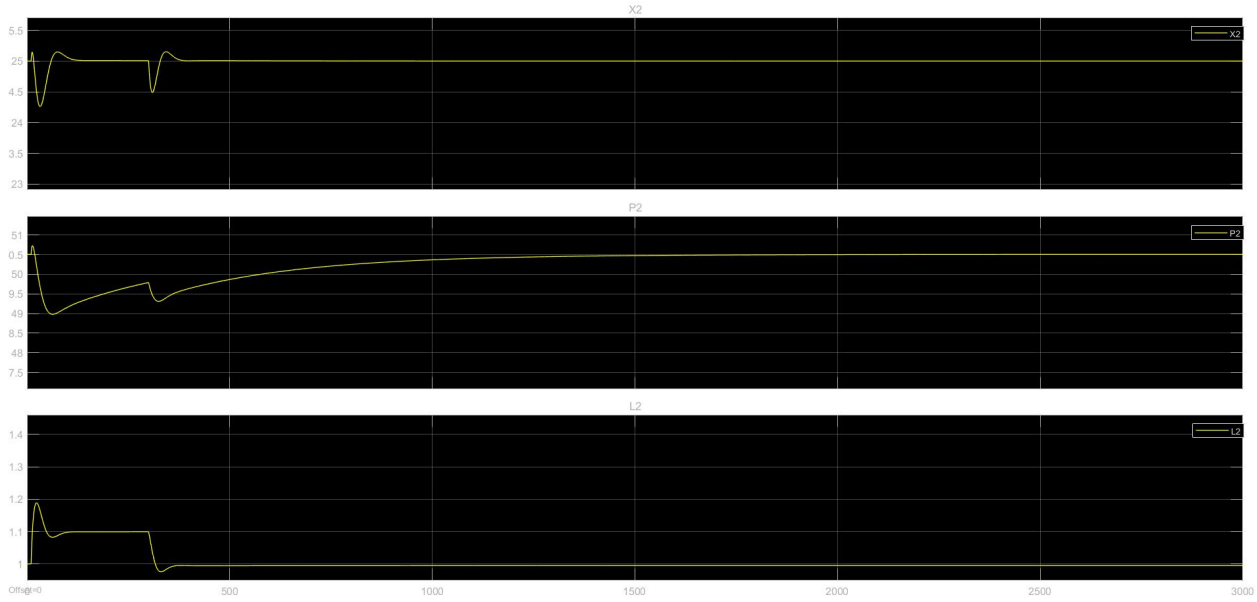

Fig 21: Disturbance rejection with PID controller

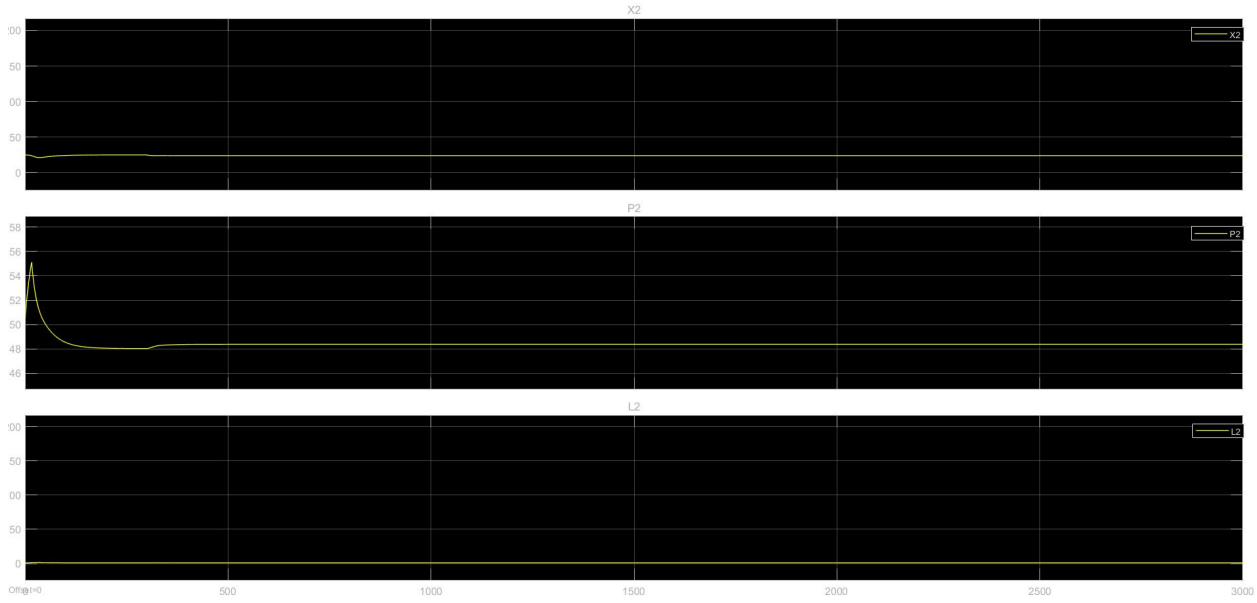

Fig 22: Disturbance rejection with RF_1 controller connected

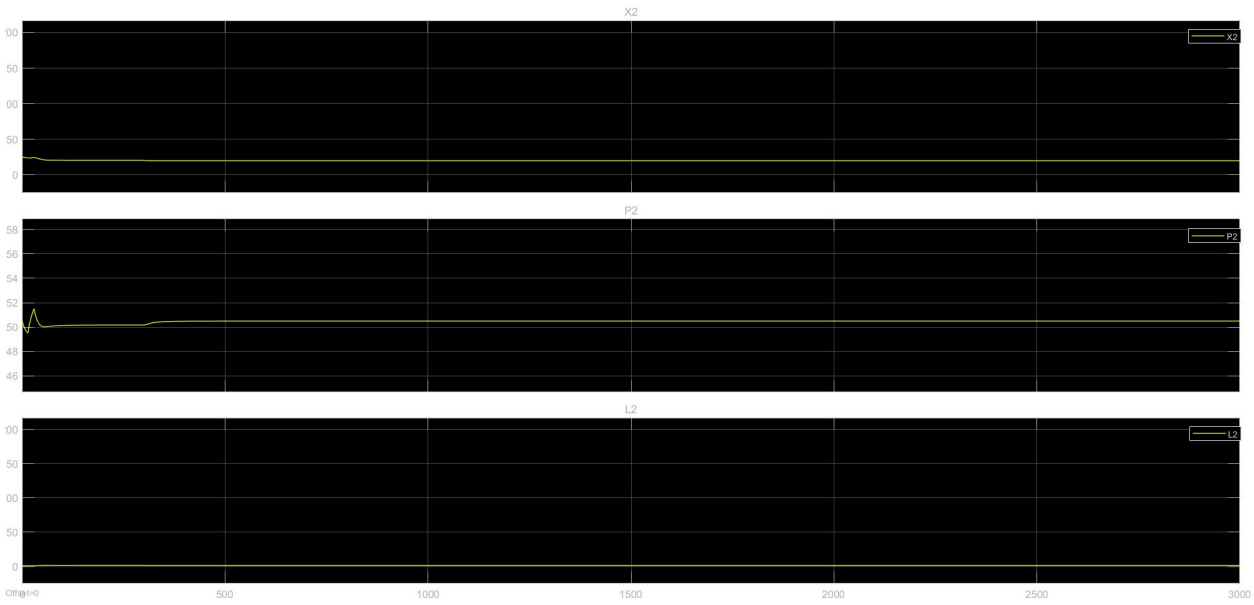

Fig 23: Disturbance rejection while RF_2 controller connected

Set point tracking scenario:

III) +10% step change in the set point of X2 at t=10 min

For X2:

Step time= 10

Initial value=0

Final value= 2.5

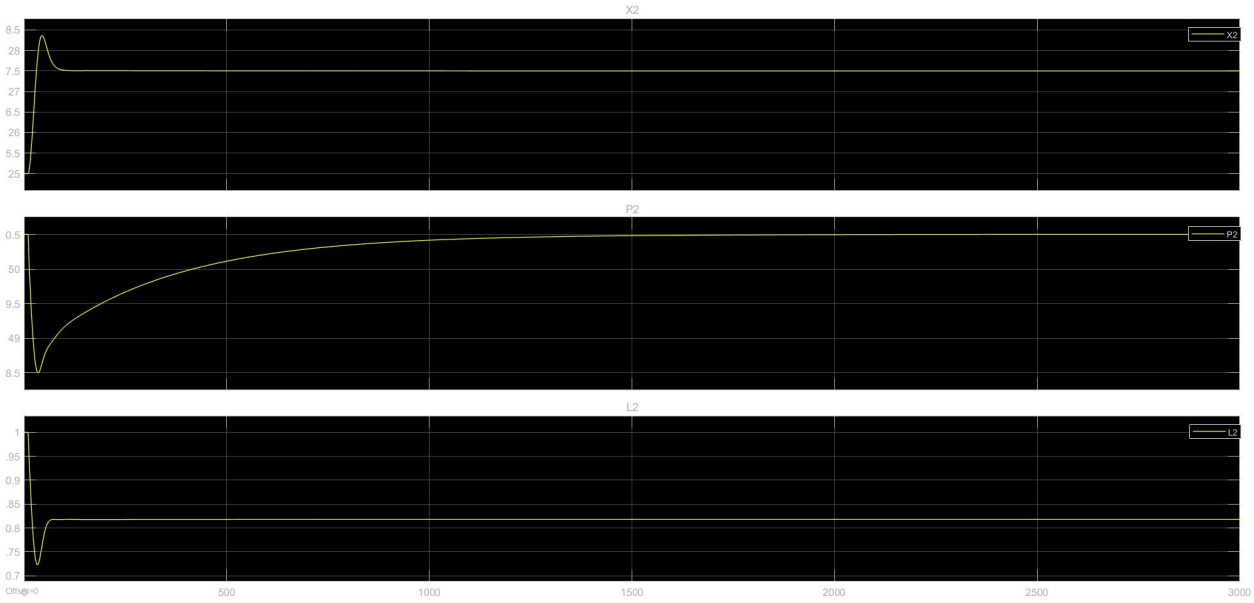

Fig 24: Set point tracking with PID controller

Fig 25: Set point tracking while RF_1 controller connected

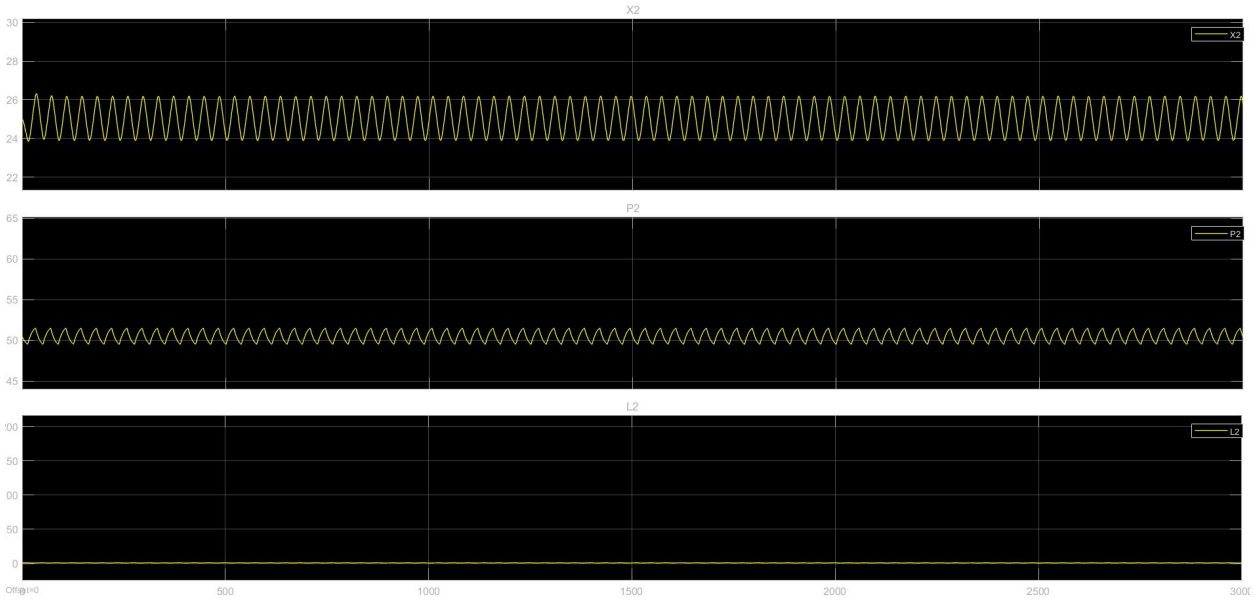

Fig 26: Set point tracking while RF_2 controller connected

IV) +5% step change in the set point of X2 at t = 10 min and +5% step change in the set point of P2 at t = 300 min

For X2:

Step time= 10

Initial value=0

Final value= 1.25

For P2:

Step time= 300

Initial value=0

Final value= 1 / 2.675265

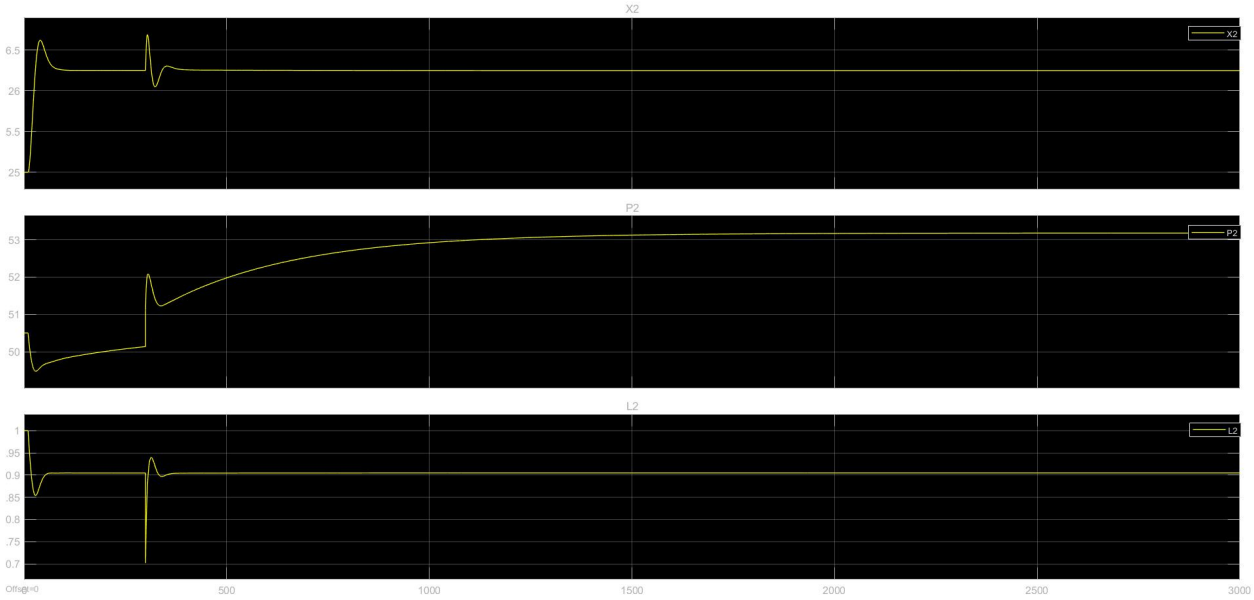

Fig 27: Set point tracking with PID controller

Fig 28: Set point tracking while RF_1 controller connected

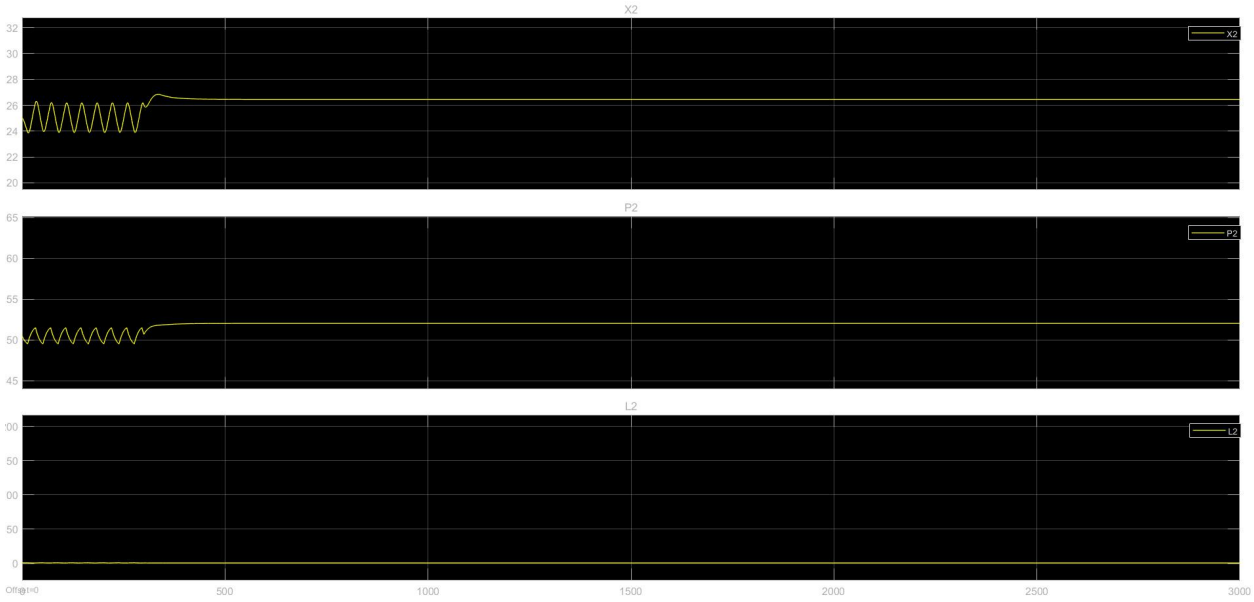

Fig 29: Set point tracking while RF_2 controller connected

Conclusion:

From the above difference case scenario, we observe that RF controller is following its nominal point without drifting away as we studied in Lecture 6 [3]. On the other hand, nominal operating point of X2 using PID controller is drifting away. It also has overshoot. That is absent in RF controller. In PID controller settling time for X2 is larger. In RF controller, though its remain within a fixed oscillation range, but it does not reach to its nominal point as smooth as PID controller does. Rise time is also high for PID controller than the RF controller in every cases. RF controller is also efficient than the PID controller in terms of calculations.

Hence, we can say that the performance of RF controller is better than PID controller.

References

1. Feedback provided by professor
2. Lecture 6, page 4, equation 6 & 7
3. Lecture 6
4. Guidelines provided by professor at e-learning portal