Universidade Federal do Ceará
Departament of Teleinformatics
Computer Engeneering
Intelligent Systems in Process Control and Automation

Homework 3

Predictive Control of a Dynamic System

Student: Davi Barreto Façanha

Professor: Prof. Dr. Michela Mulas

Fortaleza
2019

Contents

1 System Identification

“System identification is the art and science of building mathematical models of dynamic systems from observed input-output data.“ - Lennart Ljung

Lennart Ljung is a Swedish Professor in the Chair of Control Theory at Linköping University since 1976. He is considered as the father of System Identification, thanks to his pioneering research in this field.

So, utilising a simple **Simulink** schematic that represents the plant, and choosing the sample time of the input step as 0.1, we can export to *Workspace* the measured inputs-outputs in the discrete-time domain.

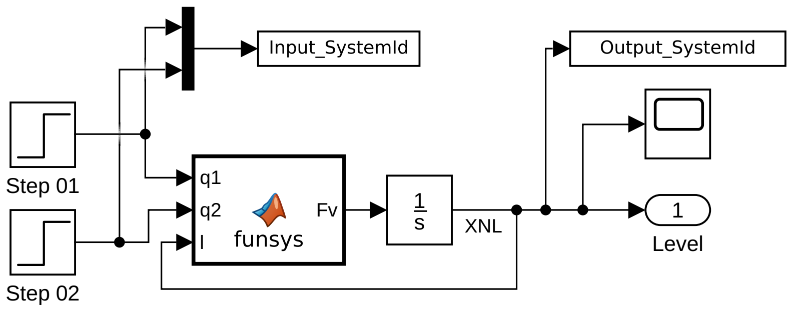

Figure 1: Plant Simulation

Using the *Matlab's System Identification Toolbox*, four guesses were made for the estimation of the Transfer Function Model, one continuous and three discrete.

- tf1 = tfest(mydata, [2 2;2 2;2 2], [1 1;1 1;1 1], Options)
- tf2 = tfest(mydata, [2 2;2 2;2 2], [1 1;1 1;1 1], Options, 'Ts', 0.1)
- tf3 = tfest(mydata, [3 3;3 3;3 3], [2 2;2 2;2 2], Options, 'Ts', 0.1)
- tf4 = tfest(mydata, [3 3;3 3;3 3], [2 0;0 2;1 1], Options, 'Ts', 0.1)

The first and the second guesses had their poles and zeros proposed by the *Matlab*. The difference between then is that *ft1* is continuous and *ft2* is discrete. Also, I tried a guess for a three-order transfer function, and for a transfer function with the same number of poles and zeros of my linearized system.

- **Transfer Functions of the Linearized System**:

  From input 1 to outputs...

$$1 : \frac{64.94s^2 + 2.572s + 0.01905}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

$$2: \frac{0.006428}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

$$3: \frac{0.6465s + 0.01274}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

From input 2 to outputs...

$$1: \frac{0.006428}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

$$2: \frac{64.94s^2 + 1.939s + 0.006428}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

$$3: \frac{0.6456s + 0.006428}{s^3 + 0.04956s^2 + 0.0005885s + 9.667 \cdot 10^{-7}}$$

Checking the *Model Output* of the toolbox, it was given:

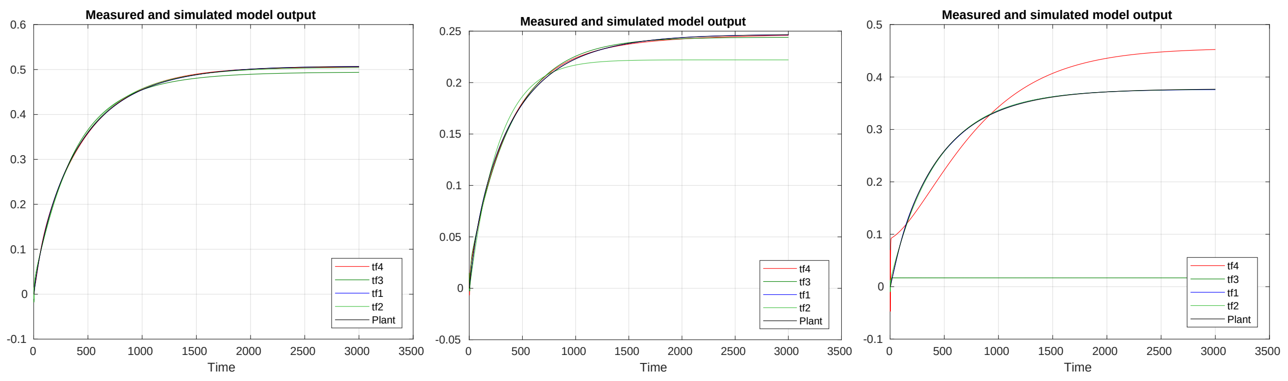

Figure 2: Model Output of System Identification for Y1, Y2 and Y3, respectively

System Identification Fitness			
	Best Fits		
	Y1	Y2	Y3
tf1	99.66	99.73	99.1
tf2	97.56	66.48	98.02
tf3	91.57	95.63	-281.3
tf4	98.57	96.98	37.42

Table 1: Optimal Fitness of System Identification Outputs

The Continuous-Linear Estimation, *ft1*, achieved the best fitness. Checking Figure 3, we can realize that, different of our Linearized Transfer Functions, the fittest model has some zeros at the right-half part of the graph, and complex poles as well.

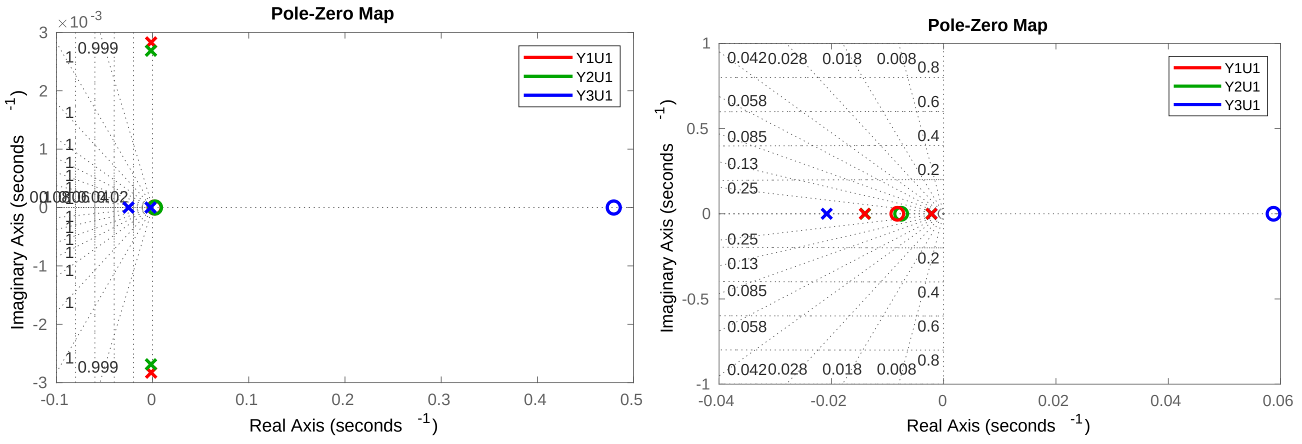

Figure 3: Poles and Zeros plots for the 6 Transfer Functions of tf1

2 Model Predictive Control

2.1 Introduction

After dedicating the semester to understand, model and control our system, we are able to know and apply a technique well known and widely used in industrial processes, the Model Predictive Control.

The MPC consists of several interactions and control strategies, within a prediction and control horizon, respectively, that aims to act on the system based on predictions of how the system should respond, and comparing them with the measured values. Check the Figure 4 for a more intuitive perspective.

Here are some overall objectives of the Model Predictive Control:

1. Prevent violations of input and output constraints,
2. Drive some output variables to their optimal set points, while maintaining other outputs within specified ranges,
3. Prevent excessive movement of the input variables,
4. Control as many process variables as possible when a sensor or actuator is not available.

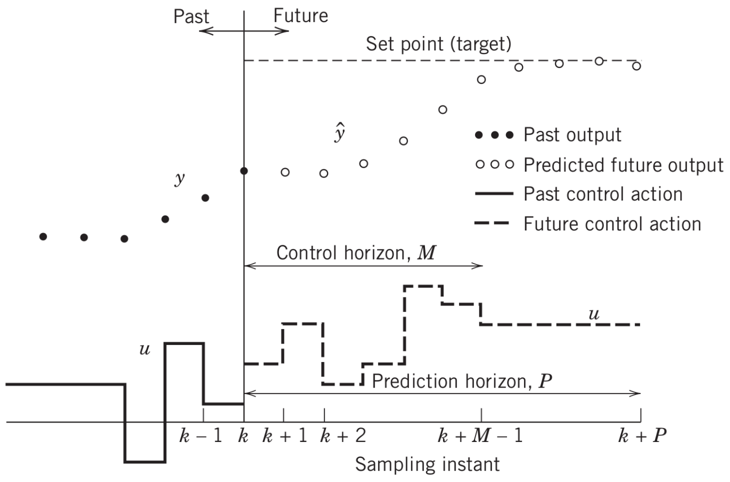

Figure 4: MPC behavior through time

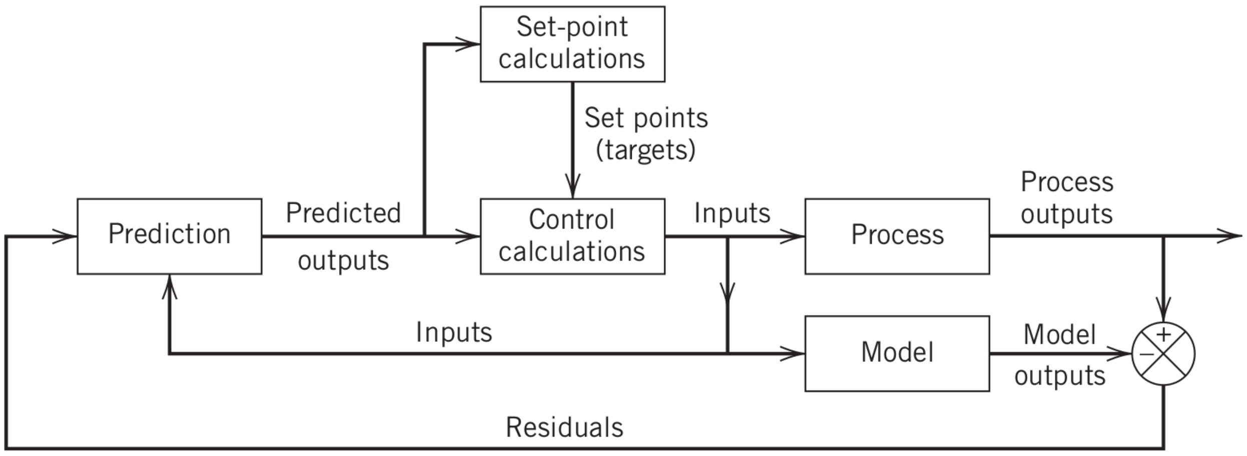

Figure 5: Block diagram for model predictive control.

In Figure 5, is shown the diagram of a MPC system, where the actual output is y, predicted output is $\hat{y}$ and the manipulated input is u. Back to Figure 4, the current sampling instant is denoted by k, and the M parameter, also called **Control Horizon**, consists of a set composed of the current input $u(k)$ and $M-1$ future inputs. That parameter is defined so that a set of P predicted outputs, also called **Predicted Horizon**, reaches the set-point in an optimal way.

Although, there is not only one way to use the Predictive Control. Predictive Control is a category that encompasses several techniques. We are going to explain the Dynamic Matrix Control, as a way to help the understanding of the control, but the Matlab MPC Toolbox, otherwise, uses an optimization problem called **Quadratic Program**, which is solved by interactive methods.

2.2 Dynamic Matrix Control

2.2.1 For SISO Problems

Two classes of discrete-time models are widely used for the MPC design:

- Finite Impulse Response
- Finite Step Response

The equation for a Finite Impulse Response (FIR) is written as,

$$y(k+1) = y(0) + \sum_{i=1}^{N} h_i u(k-i+1) \tag{1}$$

However, there is a relation between the Finite Impulse Response and the Step Response Coefficients, at Figure 6.

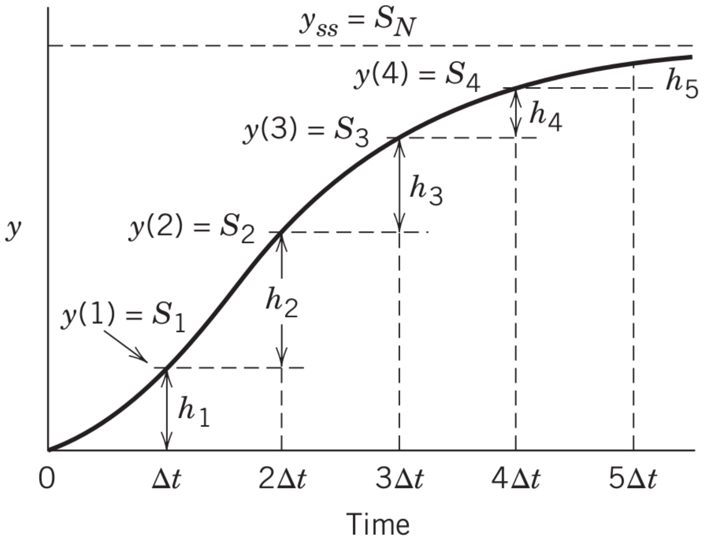

Figure 6: Step Response Coefficients

Note that we can change h_i for $Si - S_{i-1}$. Therefore, substituting in Equation 1 and choosing some value for N, $N = 3$, we find,

$$y(k+1) = y(0) + (S_1 - S0)u(k) + (S_2 - S1)u(k-1) + (S_3 - S2)u(k-2) \tag{2}$$

Defining $\Delta u(k) = u(k) - u(k-1)$, Equation 2 becomes

$$y(k+1) = y(0) + S_1\Delta u(k) + S_2\Delta u(k-1) + S_3 u(k-2) \tag{3}$$

Finally, for the full impulse response, the equation below is defined,

$$y(k+1) = y(0) + \sum_{i=1}^{N-1} S_i \Delta u(k-i+1) + S_N(k-N+1) \tag{4}$$

Understood that, let's initiate the prediction process using the Step Response Method For SISOs systems. But first, it is important to realize that we can separate the Equation 5 into,

- Effect of current control action:
$$S_i \Delta u(k-i+1)$$
- Effect of past control action:
$$\sum_{i=2}^{N-1} S_i \Delta u(k-i+1) + S_N u(k-N+1)$$

Also, up to now, we have calculated the *one-step-ahead* equation. However, substituting $k = \dot{k} + 1$, we would find an equation valid for any positive value of $\dot{k}$. Hence, we can assume, without loss of generality, that $k = \dot{k}$, and next, repeating this derivation over and over, we acquire the *j-step-ahead* prediction equation as

$$\hat{y}(k+j) = \sum_{i=1}^{j} S_i \Delta u(k-i+j) + \sum_{i=j+1}^{N-1} S_i \Delta u(k-i+j) + S_N u(k-N+j) \tag{5}$$

$$= \hat{y}(k+j) = \sum_{i=1}^{j} S_i \Delta u(k-i+j) + \hat{y}^0(k+j) \tag{6}$$

The second term of Equation 6 refers to past control actions and is called **Predicted Unforced Response**.

Yet, for a real application, the MPC does not calculate only one *j-step-ahead* prediction, but in fact a vector of predictions, for each step $(k+j)$ until $j = P$,

$$\hat{Y}(k+1) = [\hat{y}(k+1), \hat{y}(k+2), ..., \hat{y}(k+P)]^T \tag{7}$$

The same goes for the **predicted unforced responses** and the **input deviations**,

$$\hat{Y}^0(k+1) = [\hat{y}^0(k+1), \hat{y}^0(k+2), ..., \hat{y}^0(k+P)]^T \tag{8}$$

$$U(k) = [\Delta u(k), \Delta u(k+1), ..., \Delta u(k+M-1)]^T \tag{9}$$

Rewriting in vector-matrix notation, Equation 5 becomes,

$$\hat{Y}(k+1) = S\Delta U(k) + \hat{Y}^0(k+1) \tag{10}$$

With that, S is the PxM Dynamic Matrix we wanted.

$$S = \begin{bmatrix} S_1 & 0 & ... & 0 \\ S_1 & S_1 & 0 & ... \\ ... & ... & ... & 0 \\ S_M & S_{M-1} & ... & S_1 \\ S_{M+1} & S_M & ... & S_2 \\ ... & ... & ... & ... \\ S_P & S_{P-1} & ... & S_{P-M+1} \end{bmatrix} \tag{11}$$

2.2.2 Derivation for MIMO Problems

Than, by the **Principle of Superposition**, for each Controlled Variable (V) prediction, it is going to have the sum the Step Response of each Manipulated Variable (Z).

$$\hat{y}_V(k+j) = \sum_{i=1}^{N-1} S_{V1}\Delta u_1(k-i+1) + S_{V1}u_1(k+j-N) + ... + \sum_{i=1}^{N-1} S_{VZ}\Delta u_Z(k-i+1) + S_{VZ}u_Z(k+j-N) \tag{12}$$

however, we still have to deal with the **Bias Correction**. So far, we have not yet dealt with the residual generated by the difference between predicted value and the measured value. Because of this, the accuracy of our control may be affected.

To solve that problem, we may add to our prediction a bias correction, $b(k+j) = y(k) - \hat{y}(k)$.

$$\tilde{y}(k+j) = \hat{y}(k+j) + [y(k) - \hat{y}(k)] \tag{13}$$

Converting the Equation 13 to the vector-matrix notation and substituting the first term by the Equation 10, we get

$$\widetilde{Y}(k+1) = S\Delta U(k) + \hat{Y}^0(k+1) + [y(k) - \hat{y}(k)]I \tag{14}$$

Where S is now the Dynamic Matrix of the MIMO system control, with the same structure of the SISO Dynamic Matrix (Equation 11).

But now that we have add the bias term, we have to solve an optimization problem to minimize the residual.

Defining the *reference trajectory* y_r as a convex combination between the trajectory element of the previous k step and the set point of the previous k step, and putting into vector notation as well, we find,

$$y_{i,r}(k+j) = (\alpha_i)^j y_{i,r}(k) + [1 - (\alpha_i)^j] y_{i,sp}(k) \tag{15}$$

$$Y_r = [y_r(K+1), y_r(K+2), y_r(K+3), ..., y_r(K+P)] \tag{16}$$

Lastly, our optimization is defined as the predicted deviations from the *reference trajectory.* Our prediction errors are as follow,

$$\hat{E}(k+1) = Y_r(k+1) - \tilde{Y}(k+1) \tag{17}$$

$$\hat{E}^0(k+1) = Y_r(k+1) - \tilde{Y}^0(k+1) \tag{18}$$

For solving it, several methods can be applied. As we are already used to, the most usual technique of minimization involves calculating the Quadratic Error Sum at each time step. Besides that, to improve our optimization function, some Weights Coefficients were added in order to ponder the influence of both terms,

$$F_{Quadratic} = \hat{E}(k+1)^T Q \hat{E}(k+1) + \Delta E(k)^T R \Delta E(k) \tag{19}$$

Chosen the Q and R values, we should calculate the best control actions so that our condition is satisfied. The optimal set of control actions is defined as below,

$$\Delta U(k) = (\mathbf{S^T Q S} + \mathbf{R})^{-\mathbf{1}} \mathbf{S^T Q} \hat{E}^0(k+1) \tag{20}$$

Ergo, establishing a constant $\mathbf{Kc} = (\mathbf{S^T Q S} + \mathbf{R})^{-\mathbf{1}} \mathbf{S^T Q}$, we conclude that our **Optimization Problem** can be treated as a **Gain Problem** applied to the residue.

2.3 Results

At long last, we used **Matlab's MPC Toolbox** to evaluate the behavior and functionality of a Model Predictive Control.

2.3.1 Algorithm

Here, you can see the implementation of the algorithm and the block schematic in *Simulink*, Figure 7. This time, I integrated the Algorithm to the *Simulink*, setting the step-by-step configuration at the script line.

In order to facilitate the understanding of the algorithm below, I already add that my system was first linearized with the same parameters of Homework 1 and just than passed to the MPC function.

The chosen parameters can be easily viewed at Table 2.

Toolbox Parameters	
Sample Time	0.1
Prediction Horizon	50
Control Horizon	10
Input Weights	[1 1]
Rate Weights	[100 100]
Output Weights	[1 1 1]

Table 2: Table of Main Parameters

```
%% LINEARIZING STATE SPACE MODEL %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Setting the variables with the same values of the first
   Homework
% conditions.
s = 0.0154 ;         % Section Area
sp = 5*10^(-5);      % Pipe Section Area
mi = 0.5;            % Flow Constant
mi2 = 0.675;         % Flow Constant
qmax = 1.2*10^(-4); % Max. Flow Rate
lmax = 0.6;          % Max. High
g = 9.80665;         % Gravity

% The constant values of Input Flow Rate to Find the
   Operating Points
q1 = (1/3)*qmax;
q2 = (2/7)*qmax;

%% Instantiating the EDOs that Describe the System

syms L1 L2 L3

f1 = (1/s)*(q1 - mi*sp*sign(L1 - L3)*sqrt(2*g)*sqrt(abs(L1 -
   L3)));
f2 = (1/s)*(q2 + mi*sp*sign(L3 - L2)*sqrt(2*g)*sqrt(abs(L3 -
   L2)) - mi2*sp*sqrt(2*g)*sqrt(L2) );
f3 = (1/s)*(mi*sp*sign(L1 - L3)*sqrt(2*g)*sqrt(abs(L1 - L3))
   - mi*sp*sign(L3 - L2)*sqrt(2*g)*sqrt(abs(L3 - L2)) );

%% Calculating the Jacobian and Finding Matrix A

Ajacobiano = jacobian([f1 f2 f3],[L1 L2 L3]);

L1 = 0.508003919943708 ;
L2 = 0.246913405476143 ;
L3 = 0.377653414328398 ;

% Substituition into the Jacobian Matrix
AA = subs(Ajacobiano);

% Matrix A
A = double(AA);


%% Finding B Matrix for fixed Height Values.

% Steady Highs
L1 = 0.508003919943708 ;
```

```
L2 = 0.246913405476143 ;
L3 = 0.377653414328398 ;

syms q1 q2

f1 = (1/s)*(q1 - mi*sp*sign(L1 - L3)*sqrt(2*g)*sqrt(abs(L1 -
   L3)));
f2 = (1/s)*(q2 + mi*sp*sign(L3 - L2)*sqrt(2*g)*sqrt(abs(L3 -
   L2)) - mi2*sp*sqrt(2*g)*sqrt(L2) );
f3 = (1/s)*(mi*sp*sign(L1 - L3)*sqrt(2*g)*sqrt(abs(L1 - L3))
   - mi*sp*sign(L3 - L2)*sqrt(2*g)*sqrt(abs(L3 - L2)) );

Bjacobiano = jacobian([f1 f2 f3],[q1 q2]);

% Substituition into the Jacobian Matrix
q1 = (1/3)*qmax;
q2 = (2/7)*qmax;

BB = subs(Bjacobiano);

%Matrix B
B = double(BB);

%% Setting Matrix C and D of the State Space Model

C = [1 0 0 ; 0 1 0 ; 0 0 1];

D = zeros(3,2);

%% MODEL PREDICTIVE CONTROL %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

modelPlant = 'MPC_Disturb';
load_system(modelPlant)
% open_system([modelPlant '/Three Tank Nonlinear System'],'
   force')

mdlMPC = 'MPC_Disturb';
open_system(mdlMPC)

%% Defining the State Space, and Setting the Parameters of
   MPC

% State Space
sys = ss(A,B,C,D);

% Parameters of MPC
sys.InputName = {'q1','q2'};
sys.OutputName = {'L1','L2','L3'};
sys.StateName = {'L1','L2','L3'};
sys.InputGroup.MV = 2;
sys.OutputGroup.MO = [1 2];
sys.OutputGroup.UO = 3;

% Sample Time
Ts = 0.1; % 112.7 > Ts > 56.35

% Creating the MPC Object to import into Simulink
MpcMdl = mpc(sys, Ts);

% Defining Optimal Prediction Horizon and Control Horizon
MpcMdl.PredictionHorizon = 50; % 198 > T_PH > 99
MpcMdl.ControlHorizon = 10; %  0.2*T_PH = 39.36 > CH > 19.8 =
     0.1*T_PH

%% Specify Constraints for MV and MV Rate
MpcMdl.MV(1).Min = 0;
MpcMdl.MV(1).Max = 0.00012;
MpcMdl.MV(2).Min = 0;
MpcMdl.MV(2).Max = 0.00012;

%% Specify Constraints for OV
MpcMdl.OV(1).Min = 0;
MpcMdl.OV(1).Max = 0.6;
MpcMdl.OV(2).Min = 0;
MpcMdl.OV(2).Max = 0.6;
MpcMdl.OV(3).Min = 0;
MpcMdl.OV(3).Max = 0.6;

%% Specify Weights
MpcMdl.Weights.MV = [1 1];
MpcMdl.Weights.MVRate = [100 100];
MpcMdl.Weights.OV = [1 1 1];
MpcMdl.Weights.ECR = 100000;

%% Specify Simulation Options
options = mpcsimopt();
options.RefLookAhead = 'off';
options.MDLookAhead = 'off';
options.Constraints = 'on';
options.OpenLoop = 'off';

%% Setpoint Tracking Parameters
set_param("MPC_Disturb/Setpoint Tracking", 'Time', '150');
set_param("MPC_Disturb/Setpoint Tracking", 'Before', '0.2');
set_param("MPC_Disturb/Setpoint Tracking", 'After', '0.4');

%% Run Simulation
```

```
SimulT = 325 ; % Simulation Time
sim('MPC_Disturb', SimulT , []);

% Get the Simulation Time
tp = SimulMPC.Time ;

% Setpoint Tracking Plot with Noise
figure
plot(tp, SimulMPC.Data(:,1),'LineWidth',2)
hold on
plot(tp ,SimulMPC.Data(:,2),'LineWidth',2)
hold on
plot(tp, SimulMPC.Data(:,3),'LineWidth',2)
hold on
plot(tp, SimulMPC.Data(:,4),'--','LineWidth',2)
hold on
legend({'L1␣with␣Noise','L2␣with␣Noise','L3␣with␣Noise','Set␣
    Point'},'Location','northeast','Orientation','Vertival')
hold off
```

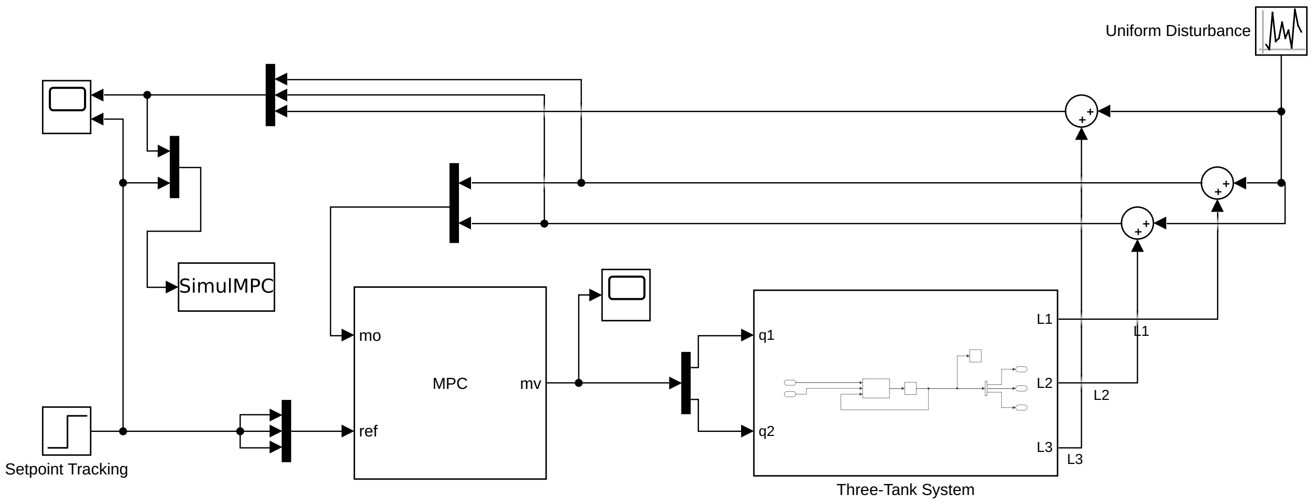

Figure 7: Block Schematic for MPC.

2.3.2 Plots and Graphs

For a visual comprehension, I chose to plot the behaviors of the Set Point Control, the Set Point Tracking Control and the Set Point Tracking Control with Noise, (Figure 8). As seen in Figure 7, the Disturbance was put on the Tank's Level Measurement, symbolizing the noise of physical level measurement sensors, with a variation range of [-0.05 0.05].

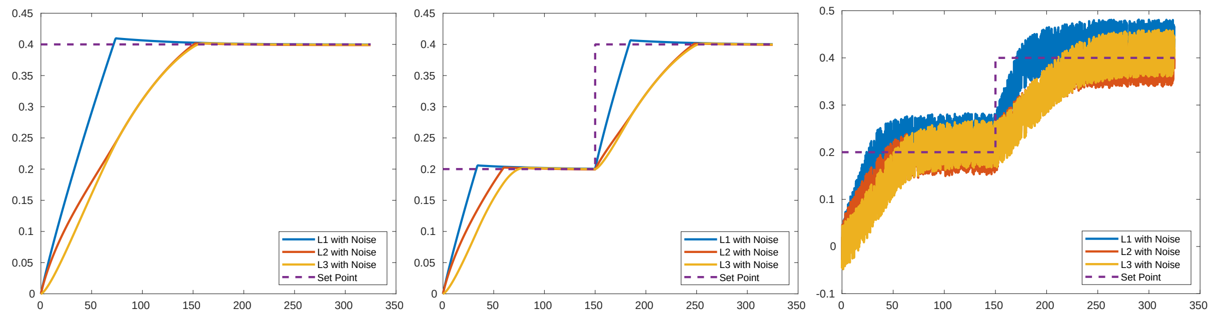

Figure 8: Behavior of a MPC for different conditions

3 Overall Analyse

During the execution of all homeworks, I did the analysis of which values of τ_C were the best and which methods had achieved a better performance only by making a visual and intuitive analysis, because normally the differences were easily recognizable. However, when I ran the *Stepinfo()* function of *matlab* for the same set-point of 0.4 for all methods, I realized that some values of τ_C slightly different than those reported in the past reports resulted in faster outputs, despite causing an overshoot (of a very low percentage compared to the improvement of the Rise Time and Settling Time).

All values and quantifications can be seen in the table 3. The green color represents the best method, the red the worst method and, also, I wanted to highlight the Genetic Algorithm because, looking at the numbers, it seemed to have been a great method. But in my case, this method was very time-consuming and saturated the input flow values into the plant (which could perhaps be solved with a better cost function, which also involved the input deviation).

As well, the Settling Time of Online Method is a bit out of the ordinary, thanks to the PID obtained by the Step Test Method. The coefficients found with that method tunned a bad controller for our system.

Performance Balance															
	1st IMC Approach			2nd IMC Approach			Online Method			Genetic Algorithm			MPC		
	Y1	Y2	Y3	Y1	Y2	Y3	Y1	Y2	Y3	Y1	Y2	Y3	Y1	Y2	Y3
Rise Time	141.1312	164.9027	154.2139	80.5579	136.9235	129.6491	336.4702	780.4805	493.5028	64.45	x	x	57.7574	116.7688	108.0681
Settling Time	558.9241	989.6467	420.9813	750.4890	475.0411	609.5476	2.8845e+03	2.1819e+03	974.3435	78.17	x	x	86.3095	144.4500	146.6552
Overshoot	6.4760	0	3.2125	5.3998	3.7099	4.5816	4.1183	0	0.4400	7.86e-10	x	x	2.4276	0.3970	0.4428
τ_C	80			2000			200			NONE					

Table 3: Overall Performance

4 Conclusion

In fact, the MPC was the controller that obtained the best results. In addition, it was extremely interesting to study and implement it.

Overall, the work was very interesting and challenging. I was able to understand and practice many concepts of Classic Control that went unnoticed by the Introduction to Control chair.

I also want to point out that there were some misunderstandings in the process of doing the homeworks, sometimes because of the time, sometimes because of the understanding, but in the end they were worth it.

I hope that my participation has also contributed in some way to all, and that you have enjoyed the work.