Unconstrained and Constrained MPC Implementation: CH5120 Mini Project

Krthan Musugunthan - AE21B039

December 2024

1 Introduction

Based on the paper *"The Quadruple-Tank Process: A Multivariable Laboratory Process with an Adjustable Zero",* by **Karl Henrik Johansson** we use the setup described to implement a Kalman filter to filter out the process and measurement noise and a linear MIMO (Multi-input, Multi-output) Model Predictive Controller to to take the system to desired set points.

2 Setup

Four tanks are named according to the schematic shown below is arranged, there is a reservoir collecting from the openings of Tank 1 and 2. Two pumps, A (pumps into Tank 1 and 4 through a distribution valve 1) and B (pumps into tank 2 and 3 through a distribution valve 2) circulates liquid collected in the reservoir.
In the paper, the author performs his study using 2 operating points based on the phase characteristics ($P_\ominus$) and ($P_\oplus$). However, this project on focuses on the $P_\ominus$ operating point.

3 System dynamics

The system's dynamical equation has been given by the following non-linear equation:

$$\frac{dh_1}{dt} = -\frac{a_1}{A_1}\sqrt{2gh_1} + \frac{a_3}{A_1}\sqrt{2gh_3} + \frac{\gamma_1 k_1}{A_1}\nu_1$$
$$\frac{dh_2}{dt} = -\frac{a_2}{A_2}\sqrt{2gh_2} + \frac{a_4}{A_4}\sqrt{2gh_4} + \frac{\gamma_2 k_2}{A_2}\nu_2$$
$$\frac{dh_3}{dt} = -\frac{a_3}{A_3}\sqrt{2gh_3} + \frac{(1-\gamma_2)k_2}{A_3}\nu_2$$
$$\frac{dh_4}{dt} = -\frac{a_4}{A_4}\sqrt{2gh_4} + \frac{(1-\gamma_1)k_1}{A_4}\nu_1$$

where, A_i is the area of cross section of tank i; a_i is the area of cross-section of the outlet hole ; h_i is the water level; γ_i is the valve position; ν_i is the voltage applied to pump i;$k_i\nu_i$ is the corresponding flow; g is the acceleration of gravity.

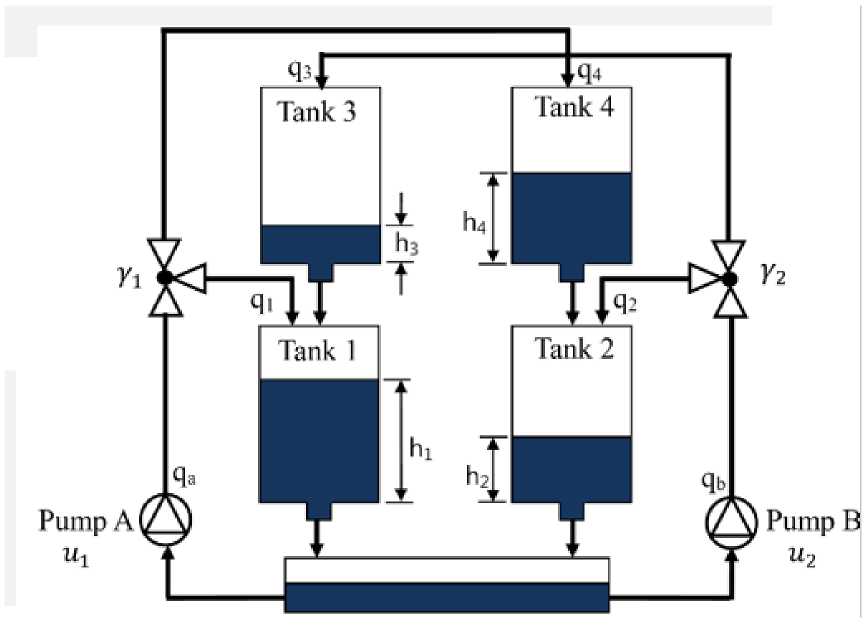

Figure 1: Setup of the experiment

4 Model Predictive Control

4.1 State space model

The state space model of the system can be given as follows: The linearized formulation :

$$\frac{dx}{dt} = \begin{bmatrix} -\frac{1}{T_1} & 0 & \frac{A_3}{A_1 T_3} & 0 \\ 0 & -\frac{1}{T_2} & 0 & -\frac{A_4}{A_2 T_4} \\ 0 & 0 & -\frac{1}{T_3} & 0 \\ 0 & 0 & 0 & -\frac{1}{T_4} \end{bmatrix} x + \begin{bmatrix} \frac{\gamma_1 k_1}{A_1} & 0 \\ 0 & \frac{\gamma_2 k_2}{A_2} \\ 0 & \frac{(1-\gamma_2)k_2}{A_3} \\ \frac{(1-\gamma_1)k_1}{A_4} & 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} k_c & 0 & 0 & 0 \\ 0 & k_c & 0 & 0 \end{bmatrix} x$$

where the constants are given by:

$$T_i = \frac{A_i}{a_i}\sqrt{\frac{2h_i^0}{g}}, \qquad i = 1, 2, 3, 4$$

4.2 Augmented model

To implement a model predictive controller, we need to convert the state space model into an 'augmented' state space model which is represented by:

$$\begin{bmatrix} \Delta x(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A_m & O_m^T \\ C_m A_m & I \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix} \Delta u(k)$$

$$y(k+1) = \begin{bmatrix} 0_m & I \end{bmatrix} \begin{bmatrix} \Delta x(k+1) \\ y(k+1) \end{bmatrix}$$

Here, our augmented matrices are:

$$A_a = \begin{bmatrix} A_m & O_m^T \\ C_m A_m & I \end{bmatrix}$$

$$B_a = \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix}$$

$$C_a = \begin{bmatrix} 0_m & I \end{bmatrix}$$

In this form,

$$\Delta x(k+1) = x(k+1) - x(k)$$

This is the change in change (incremental state), and

$$\Delta u(k+1) = u(k+1) - u(k)$$

which is the change in input or incremental input. We denote the above system as:

$$X(k+1) = A_a X(k) + B_a \Delta u(k)$$

$$y(k+1) = C_a X(k+1)$$

where,

$$X(k) = \begin{bmatrix} \Delta x(k) \\ y(k) \end{bmatrix}$$

Representing $y(k+1)$ in terms of $x(k)$ and $\Delta u(k)$, we get:

$$y(k+1) = Fx(k) + \Phi \Delta u(k)$$

Here:

$$F = \begin{bmatrix} CA \\ CA^2 \\ CA^3 \\ ... \\ ... \\ CA^{N_p} \end{bmatrix} \qquad \Phi = \begin{bmatrix} CB & 0 & ... & ... & 0 \\ CAB & CB & 0 & .... & 0 \\ ... & ... & ... & ... & ... \\ CA^{N_p-1}B & CA^{N_p-2}B & ... & ... & CA^{N_p-N_c}B \end{bmatrix}$$

In the above matrices, N_p and N_c (where, $N_p >> N_c$) are the prediction and control horizons respectively. They act as tuning parameters for the controller.

4.3 Cost function

The cost function for the augmented model is given as:

$$J = (R_s - Y_p)^T Q (R_s - Y_p) + \Delta U^T R \Delta U$$

where,

- R_s is the set point vector
- Y_p is the predicted output point vector
- Q is the state weight matrix
- ΔU is the incremental input matrix
- R is the input weight matrix

For every step, we optimize the above cost function with respect to our incremental input. We only implement the first input step at each iteration.

5 Unconstrained Model Predictive Control

For the unconstrained model predictive control, we do not have constraints on the output, input or the input rate. The following is the code for the unconstrained MPC implementation: The parameters used are:

- Prediction horizon(N_p) = 10
- Control horizon(N_c) = 2
- State weight (Q) = diag([1, 1, 0, 0])
- Input weight (R) = diag([0.5, 0.5])
- Setpoint = [13.4; 13.7; 0; 0]
- Simulation iterations = 200
- Initial covariance = I
- Measurement covariance = 0.01 * I
- Process noise covariance = 0.1 * I

```
clc; clear all;

%parameters of the system
A1 = 28; A2 = 32; A3 = 28; A4 = 32;
a1 = 0.071; a2 = 0.057; a3 = 0.071; a4 = 0.057;
kc = 1; g = 981;

%Operating point parameters
h1_o = 12.4; h2_o = 12.7; h3_o = 1.8; h4_o = 1.4;
h_op = [h1_o; h2_o; h3_o; h4_o]; %operating point
T1 = (A1/a1) * sqrt(2*h1_o/981); T2 = (A2/a2)* sqrt(2*h2_o/981); T3
    = (A3/a3) * sqrt(2 * h3_o /981); T4 = (A4/a4) * sqrt(2 * h4_o
    /981);
v1 = 3; v2 = 3;
```

```
k1 = 3.33; k2  = 3.35;
gamma1 = 0.70; gamma2 = 0.60;

% defining the A, B and H matrices
Ad = [-1/T1 0 A3/(A1*T3) 0; 0 -1/T2 0 A4/(A2*T4); 0 0 -1/T3 0; 0 0 0
    -1/T4 ];

Bd = [gamma1*k1/A1 0; 0 gamma2*k2/A2; 0 (1-gamma2)*k2/A3; (1-gamma1)
   *k1/A4 0];

Hd = [kc 0 0 0; 0 kc 0 0; 0 0 kc 0; 0 0 0 kc];
Dd = zeros(size(Hd,1),size(Bd,2));

% MPC parameters
Np = 20;  % Prediction horizon
Nc = 5;   % Control horizon
Q = diag([1, 1, 0, 0]);  % State weights
R = diag([0.5, 0.5]);    % Input weights

% Constraints
% u_min = [0; 0];
% u_max = [20; 20];
% du_min = [-5; -5];
% du_max = [5; 5];

%% Converting to an augmented matrix
A_a = [Ad, zeros(size(Ad,1)); Hd*Ad, eye(size(Hd))];
B_a = [Bd; Hd*Bd];
C_a = [zeros(size(Ad,1)), eye(size(Hd))];

%% building F and phi matrix
F = [];
phi = [];
for i = 1:Np
    F = [F; C_a*(A_a^i)];
    row = [];
    for j = 1:Nc
        if i >= j
            row = [row, C_a * A_a^(i-j) * B_a];
        else
            row = [row, zeros(size(Bd))];
        end
    end
    phi = [phi; row];
end
```

```
% Cost matrices
Q_bar = kron(eye(Np), Q); % Weighting for predicted states
R_bar = kron(eye(Nc), R); % Weighting for control inputs

%set point
ref = [13.4; 13.7; 0; 0];
ref_matrix = repmat(ref,Np,1);
Nsim = 200; %sim iteration

x = h_op; % Initial state

x_log = [];
u_log = [];
y_log = [];

% Initial Kalman filter setup
x_hat = h_op; % Initial state estimate
P = eye(size(Ad)); % Initial error covariance matrix

u_prev = [0; 0];

%% simulating mpc and kf
for i = 1:Nsim

    y_meas = Hd * x + randn(size(Hd,1),1) * 0.01; % measurement with
         noise
    [x_hat, P] = kalman_update(Ad, Bd, Hd, x_hat, P, y_meas); %
        kalman filter function call, state estimate

    x_aug = [x_hat - x ; y_meas]; %augmented state definition

    %quadratic function defined as 1/2 x' * H * x + f' x, we define
        H and f
    %below, for augmented ss, x --> Del u

    H_cost = phi' * Q_bar * phi + R_bar;
    f_cost = -(ref_matrix - F * x_aug)' * Q_bar' * phi;

    % quadprog function call, returns minimizer
    del_u_opt = quadprog(H_cost,f_cost,[], [], [], [], [], [], [],
        optimoptions('quadprog', 'Display', 'off')); %delta u mpc
    del_u = del_u_opt(1:size(Bd, 2)); %delta u

    u = u_prev + del_u;
    % Logging control and state variables
    u_log(:, i) = u;
    x_log(:, i) = x_aug;
    y_log(:, i) = C_a * x_aug;

    % Apply input to the augmented plant
    x_aug = A_a * x_aug + B_a * del_u;

    x = x + x_aug(1:4) + randn(size(Ad,1),1) * 0.1; %get x from
       del_x(= augmented) and process noise
    u_prev = u;
end

%% KF function
function [x_hat, P] = kalman_update(Ad, Bd, Cd, x_hat, P, y_meas)
    % Kalman filter measurement update
    R_kal = eye(size(Cd, 1)) * 0.0001; % Measurement noise
       covariance
    Q_kal = eye(size(Ad, 1)) * 0.01; % Process noise covariance

    % Prediction step
    x_hat = Ad * x_hat;
    P = Ad * P * Ad' + Q_kal;

    % Update step
    K = P * Cd' / (Cd * P * Cd' + R_kal); % Kalman gain
    x_hat = x_hat + K * (y_meas - Cd * x_hat);
    P = (eye(size(K, 1)) - K * Cd) * P;
end

figure();
subplot(2,1,1)
plot(1:Nsim, y_log(1,:), '-b', LineWidth=1.5);title('Tank height 1
   vs. iterations'); xlabel('Iterations (1 = 0.1s)'); ylabel('height
    of tank 1 (cm)');hold on;
plot(1:Nsim, repmat(ref(1), 1, Nsim), '--b', LineWidth=1.5); hold
   off;
legend('Controlled tank height 1 (cm)', 'Given reference')
subplot(2,1,2);
plot(1:Nsim, y_log(2,:), '-r', LineWidth=1.5);xlabel('Iterations (1
   = 0.1s)'); ylabel('height of tank 2(cm)'); hold on;
plot(1:Nsim, repmat(ref(2),1,Nsim), '--r', LineWidth=1.5); title('
   Tank height 2 vs. iterations'); hold off;
legend('Controlled tank height 2 (cm)', 'Given reference')

figure();
subplot(2,1,1)
```

```
stairs(1:Nsim, u_log(1,:), '-b', LineWidth=1.5); title('Change in
    Input voltage v1 (V)')
subplot(2,1,2);
stairs(1:Nsim, u_log(2,:), '-r', LineWidth=1.5); title('Change in
    Input voltage v2 (V)')
```

The results of the control and change in input is given below.

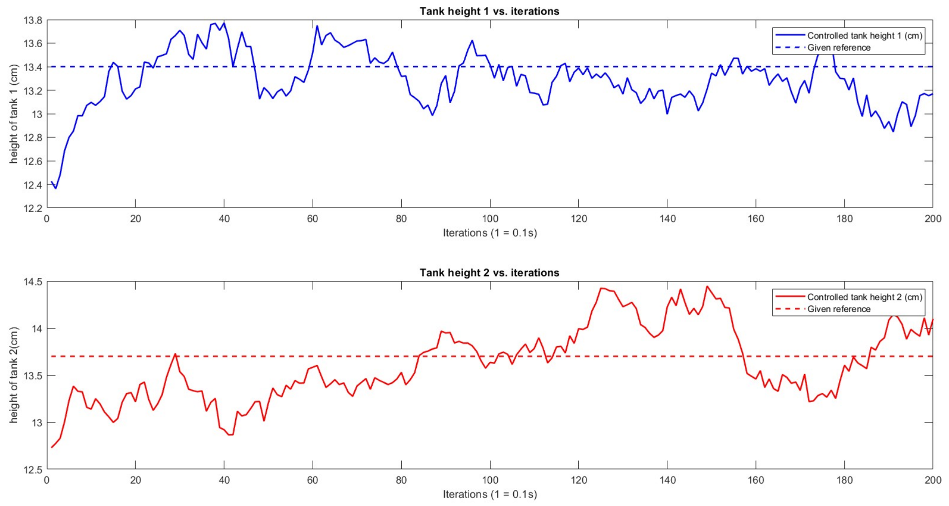

Figure 2: Tank height - a.a

5.1 Heuristics justification (a.b)

To justify the heuristics choice, the simulation was run multiple times with different values of N_p, N_c, Q and R.

- The choice of N_p above 10 made process faster. A commensurate lower control horizon was chose and it is 2.
- For the state and input weights, we give equal weight to states we are trying to reach the set point with and since u1 affects h1 and u2 affects h2, we give equal weights to the inputs as well.
- As for the magnitude, 0.5 for the input matrix made sure the change in input is less.

5.2 Closed loop stability analysis (a.c)

To analyze the closed loop stability of the system, we can represent the change in input as the following:

$$\Delta U = K_y r(k) - K_{MPC}\hat{x}_k$$

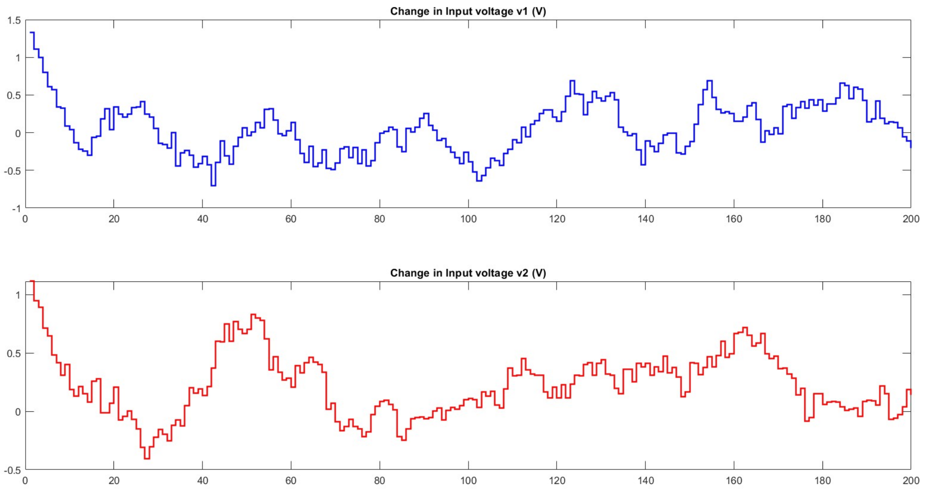

Figure 3: Change in input voltage - a.a

Where we take first elements of $K_{MPC} = (\Phi' Q \Phi + R)^{-1} \Phi' F$; $K_y = (\Phi' Q \Phi + R)^{-1} \Phi' R_s$. We need to find the eigenvalues of the matrix $(A - BK_{MPC})$ to comment about stability.

5.2.1 After first move

For the first move, the gain matrix of MPC comes out to be:

$$K_{MPC} = \begin{bmatrix} -0.0253 & 0.0001 & 0.0641 & -0.0001 & 1.6121 & -0.0046 & 0 & 0 \\ -0.0002 & -0.0169 & 0.0005 & 0.0495 & 0.0131 & 1.5453 & 0 & 0 \end{bmatrix}$$

The corresponding closed loop state matrix's eigenvalue is the following:

$$\begin{bmatrix} 1.0000 \\ 1.0000 \\ 0.8677 \\ 0.9042 \\ -0.0419 \\ -0.0159 \\ -0.0111 \\ -0.0333 \end{bmatrix}$$

The last 4 elements of the matrix represent the output eigenvalues and since they have a negative real part, they take the system to equilibrium making it stable.

5.2.2 Around set-point

Around the set point, the eigenvalues comes out to be: $\begin{bmatrix} -9.1913 \\ 0.5263 \\ -0.0131 \\ -0.0459 \\ -0.0342 \\ 1.0000 \\ 1.0000 \\ 1.0000 \end{bmatrix}$ The output eigenvalues are not negative. Hence, it is not stable. This is due to the noise in the system.

6 Constrained Model Predictive Control

For the constrained case, the constraints are on the absolute input and change in input. Since our decision variable is ΔU, we need to convert the constraints and represent them as a matrix in ΔU form. The given constraints are:

$$\Delta U_{min} = \begin{bmatrix} -5 \\ -5 \end{bmatrix}$$

$$\Delta U_{max} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$U_{min} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$U_{max} = \begin{bmatrix} 20 \\ 20 \end{bmatrix}$$

Converting these to ΔU formulation, gives us:

$$-C_2 \Delta U \leq -U_{min} + C_1 u(k-1)$$

$$C_2 \Delta U \leq U_{max} - C_1 u(k-1)$$

where $C_1 = \begin{bmatrix} I \\ I \\ I \\ ... \\ I \end{bmatrix}$ and $C_2 = \begin{bmatrix} I & 0 & 0 & ... & 0 \\ I & I & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ I & I & I & ... & I \end{bmatrix}$ And for input change constraints, we get:

$$\begin{bmatrix} -I \\ I \end{bmatrix} \Delta U \leq \begin{bmatrix} -\Delta U_{min} \\ -\Delta U_{max} \end{bmatrix}$$

The code for the constrained MPC controller is presented below:

```
clc; clear all; close all;

%parameters of the system
A1 = 28; A2 = 32; A3 = 28; A4 = 32;
```

```
a1 = 0.071; a2 = 0.057; a3 = 0.071; a4 = 0.057;
kc = 1; g = 981;

%Operating point parameters
h1_o = 12.4; h2_o = 12.7; h3_o = 1.8; h4_o = 1.4;
h_op = [h1_o; h2_o; h3_o; h4_o]; %operating point
T1 = (A1/a1) * sqrt(2*h1_o/981); T2 = (A2/a2)* sqrt(2*h2_o/981); T3
   = (A3/a3) * sqrt(2 * h3_o /981); T4 = (A4/a4) * sqrt(2 * h4_o
   /981);
v1 = 3; v2 = 3;
k1 = 3.33; k2  = 3.35;
gamma1 = 0.70; gamma2 = 0.60;

% defining the A, B and H matrices
Ad = [-1/T1 0 A3/(A1*T3) 0; 0 -1/T2 0 A4/(A2*T4); 0 0 -1/T3 0; 0 0 0
    -1/T4 ];

Bd = [gamma1*k1/A1 0; 0 gamma2*k2/A2; 0 (1-gamma2)*k2/A3; (1-gamma1)
   *k1/A4 0];

Hd = [kc 0 0 0; 0 kc 0 0; 0 0 kc 0; 0 0 0 kc];
Dd = zeros(size(Hd,1),size(Bd,2));


% MPC parameters
Np = 20;  % Prediction horizon
Nc = 5;   % Control horizon
Q = diag([0, 0, 100, 100]);  % State weights
R = diag([1, 1]);    % Input weights

% Constraints
u_min = 0 * ones(Nc * size(Bd,2), 1);
u_max = 20 * ones(Nc * size(Bd,2), 1);
du_min = -5 * ones(Nc * size(Bd,2), 1);
du_max = 5 * ones(Nc * size(Bd,2), 1);

%% Converting to an augmented matrix
A_a = [Ad, zeros(size(Ad,1)); Hd*Ad, eye(size(Hd))];
B_a = [Bd; Hd*Bd];
C_a = [zeros(size(Ad,1)), eye(size(Hd))];

%% building F and phi matrix
F = [];
phi = [];
for i = 1:Np
    F = [F; C_a*(A_a^i)];
    row = [];
    for j = 1:Nc
        if i >= j
            row = [row, C_a * A_a^(i-j) * B_a];
        else
            row = [row, zeros(size(Bd))];
        end
    end
    phi = [phi; row];
end


% Cost matrices
Q_bar = kron(eye(Np), Q); % Weighting for predicted states
R_bar = kron(eye(Nc), R); % Weighting for control inputs

%set point
ref = [0; 0; 2.4; 2.8];
ref_matrix = repmat(ref,Np,1);
Nsim = 200; %sim iteration

x = h_op; % Initial state

x_log = [];
u_log = [];
y_log = [];

% Initial Kalman filter setup
x_hat = h_op; % Initial state estimate
P = eye(size(Ad)); % Initial error covariance matrix

%setting initial u
u_prev = [0;0];

n = Nc; % Number of blocks
m = 2; % Size of identity matrices

I_block = eye(m); % Create identity matrix block
T = blkdiag(I_block, I_block, I_block, I_block, I_block); %
   Initialize diagonal

for i = 2:n
    for j = 1:i-1
        T((i-1)*m+1:i*m, (j-1)*m+1:j*m) = I_block; % Add identity to
             lower diagonal blocks
    end
```

```
end

%% simulating mpc and kf
for i = 1:Nsim

    y_meas = Hd * x + randn(size(Hd,1),1) * 0.01; % measurement with
         noise
    [x_hat, P] = kalman_update(Ad, Bd, Hd, x_hat, P, y_meas); %
        kalman filter function call, state estimate

    x_aug = [x_hat - x ; y_meas]; %augmented state definition

    %quadratic function defined as 1/2 x' * H * x + f' x, we define
        H and f
    %below, for augmented ss, x --> Del u

    H_cost = phi' * Q_bar * phi + R_bar;
    f_cost = -(ref_matrix - F * x_aug)' * Q_bar' * phi;

    %% finding Aineq and Bineq for constraints
    % rate of input change constraint
    Bineq_rate = [-du_min; du_max];
    Aineq_rate = [-eye(Nc * 2); eye(Nc * 2)];

    %input constraint
    Bineq_abs = [-u_min + repmat(u_prev, Nc, 1); u_max - repmat(
        u_prev, Nc, 1)];
    Aineq_abs = [-T; T];

    %stacking the matrices on top of each other
    Aineq = [Aineq_abs; Aineq_rate];
    Bineq = [Bineq_abs; Bineq_rate];

    % quadprog function call, returns minimizer
    del_u_opt = quadprog(H_cost,f_cost,Aineq, Bineq, [], [], [], [],
         [], optimoptions('quadprog', 'Display', 'off')); %delta u
        mpc
    del_u_applied = del_u_opt(1:size(Bd, 2)); %delta u

    u_applied = u_prev + del_u_applied;

    % Logging control and state variables
    u_log(:, i) = u_applied;
    x_log(:, i) = x_aug;
    y_log(:, i) = C_a * x_aug;
    % Apply input to the augmented plant
    x_aug = A_a * x_aug + B_a * del_u_applied;

    x = x + x_aug(1:4) + randn(size(Ad,1),1) * 0.1; %get x from
       del_x(= augmented) and process noise
    u_prev = u_applied;

end

%% KF function
function [x_hat, P] = kalman_update(Ad, Bd, Cd, x_hat, P, y_meas)
    % Kalman filter measurement update
    R_kal = eye(size(Cd, 1)) * 0.0001; % Measurement noise
       covariance
    Q_kal = eye(size(Ad, 1)) * 0.01; % Process noise covariance

    % Prediction step
    x_hat = Ad * x_hat;
    P = Ad * P * Ad' + Q_kal;

    % Update step
    K = P * Cd' / (Cd * P * Cd' + R_kal); % Kalman gain
    x_hat = x_hat + K * (y_meas - Cd * x_hat);
    P = (eye(size(K, 1)) - K * Cd) * P;
end

figure();
subplot(2,1,1)
plot(1:Nsim, y_log(3,:), '-b', LineWidth=1.5);title('Tank height 3
   vs. iterations'); xlabel('Iterations (1 = 0.1s)'); ylabel('height
    of tank 3 (cm)');hold on;
plot(1:Nsim, repmat(ref(3), 1, Nsim), '--b', LineWidth=1.5); hold
   off;
legend('Controlled tank height 3 (cm)', 'Given reference')
subplot(2,1,2);
plot(1:Nsim, y_log(4,:), '-r', LineWidth=1.5);xlabel('Iterations (1
   = 0.1s)'); ylabel('height of tank 4(cm)'); hold on;
plot(1:Nsim, repmat(ref(4),1,Nsim), '--r', LineWidth=1.5); title('
   Tank height 4 vs. iterations'); hold off;
legend('Controlled tank height 4 (cm)', 'Given reference')

figure();
subplot(2,1,1)
stairs(1:Nsim, u_log(1,:), '-b', LineWidth=1.5); title('Change in
   Input voltage v1 (V)')
subplot(2,1,2);
```

```
stairs(1:Nsim, u_log(2,:), '-r', LineWidth=1.5); title('Change in
    Input voltage v2 (V)')
```

6.1 b. h_3 and h_4 are controlled

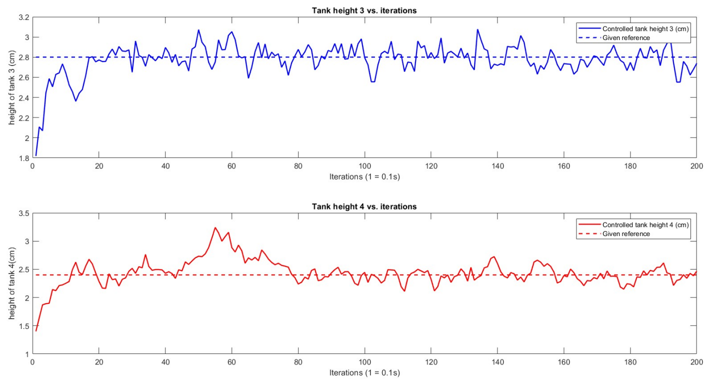

Figure 4: Tank heights

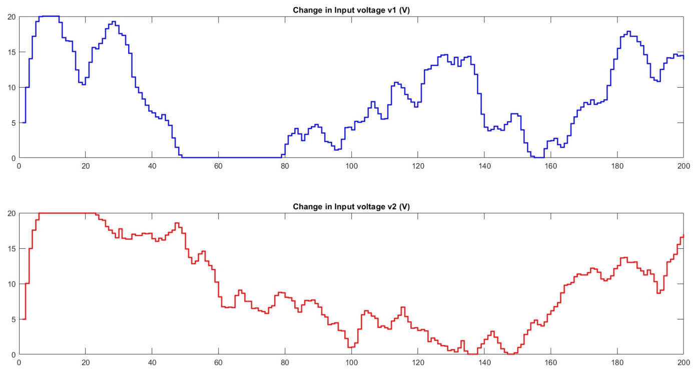

Figure 5: Inputs

6.1.1 Kalman filter performance

The performance of the controller increases with smaller initial co variance. The lesser is the Q and R matrices, lesser is its deviations from the set-point during tracking.

6.2 c. h_2 and h_3 are controlled

When h_1 and h_4 are measured, we control h_2 and h_3. The plots are presented below. We see that the controller is able to track h_1 but is not able to track h_3

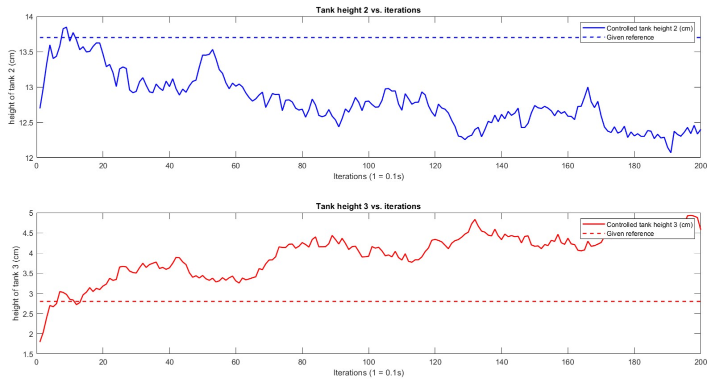

Figure 6: Tank heights

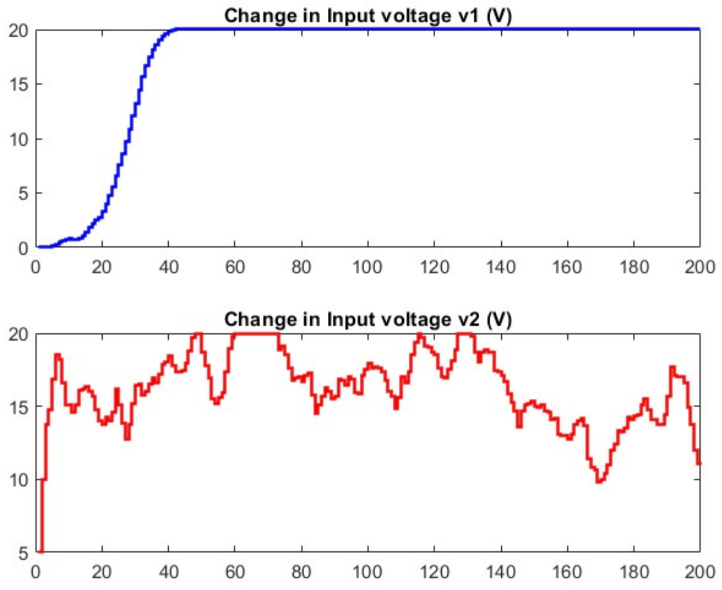

Figure 7: Enter Caption

6.3 d. h_1 and h_3 are controlled

When h_2 and h_4 are measured, we control h_1 and h_3. The plots are presented below. We see that the controller is able to track h_1 but is not able to track h_3

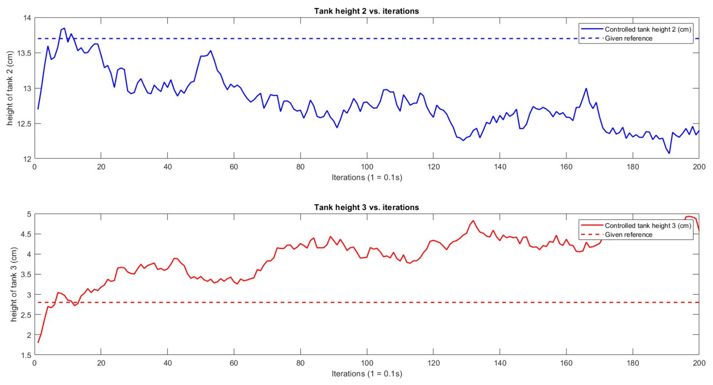

Figure 8: Tank heights

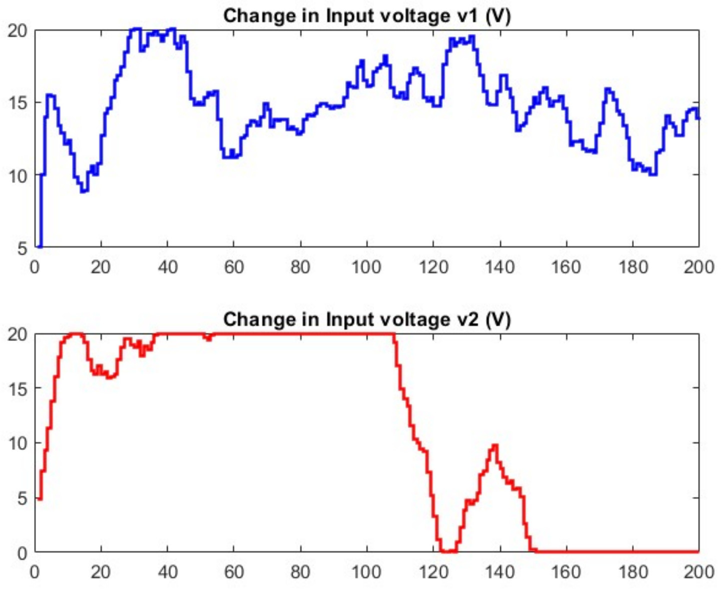

Figure 9: Inputs

7 Conclusion

The Model Predictive Controller and its advantage of using it in MIMO systems and addition of constraints makes it a very versatile controller. However, it does have a lower stability margin which can be handled by using an internal PID.