COMP0211 - Control 2
Group 06
Project Report

Tim Frankel — 24009624
Yash Joshi — 24003413
Xavier Parker — 24077390
Artemis Androutselli Theotoki — 24045883

19/12/2025

Contents

1 Code and Derivation

1.1 PID implementation and tuning

For our PID controller, we used the code shown below within the Simulink model shown in figure 1. Through iterative testing (trial and error), we found the optimal gain values to be: $K_p = 40, K_i = 0.00001, K_d = 2$. With these parameters, the pendulum is able to maintain upright balance, with the pendulum angle $|\alpha| < 0.6$ degrees, for at least 20 seconds, as seen in figure 2.

```
function [u_pend,integral_pend] = fcn(error_pend, pre_error_pend, pre_integral_pend)
    % PID controller parameters
    Kp=40;          % Proportional gain
    Ki=0.00001;     % Integral gain
    Kd=2;           % Derivative gain
    ts=0.002;       % Sampling time (in seconds)

    % Update the integral term: previous integral + error * sample time
    integral_pend = pre_integral_pend + (error_pend*ts) ;

    % Compute the derivative term: rate of change of error
    derivative = (error_pend - pre_error_pend)/ts ;

    % Calculate the PID output
    u_pend= (Kp*error_pend) + (Ki*integral_pend) + (Kd*derivative);
end
```

Listing 1: PID Controller Function

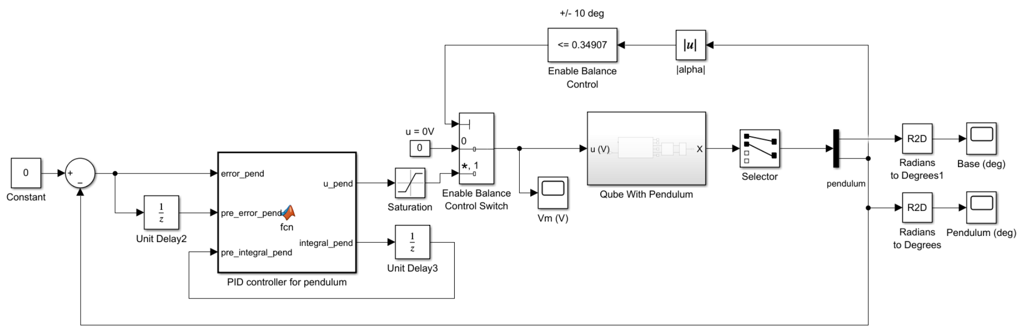


Figure 1: PID Controller Simulink Diagram

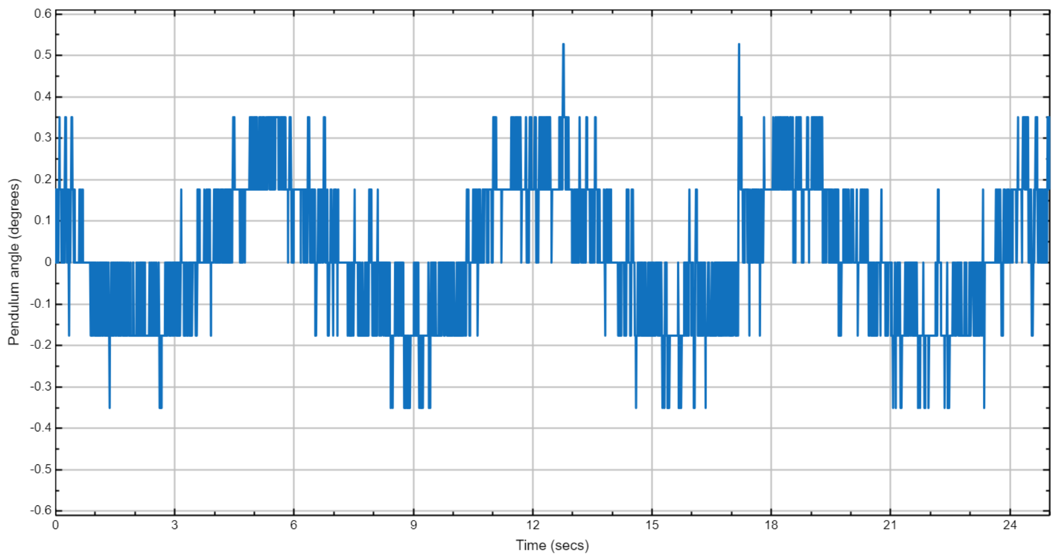


Figure 2: With our PID Controller, the pendulum angle is within the limits specified

1.2 State-space modelling from ODEs

We begin with the four equations given to us:

$$(J_r + m_p r^2)\ddot{\theta} - m_p l r \ddot{\alpha} + b_r \dot{\theta} = \tau \tag{1}$$

$$m_p l r \ddot{\theta} - J_p \ddot{\alpha} + m_p g l \alpha - b_p \dot{\alpha} = 0 \tag{2}$$

$$v_m - R_m i_m - k_m \dot{\theta} = 0 \tag{3}$$

$$\tau = k_t i_m \tag{4}$$

We can rearrange equation 4 in terms of i_m, and substitute this into equation 3:

$$i_m = \tau / k_t \tag{5}$$

$$v_m - \frac{R_m \tau}{k_t} - k_m \dot{\theta} = 0 \tag{6}$$

We rearrange equation 6 in terms of τ and substitute this into equation 1:

$$v_m - k_m \dot{\theta} = \frac{R_m}{k_t}\tau \tag{7}$$

$$\tau = \frac{k_t}{R_m} v_m - \frac{k_t k_m}{R_m}\dot{\theta} \tag{8}$$

$$(J_r + m_p r^2)\ddot{\theta} - m_p l r \ddot{\alpha} + b_r \dot{\theta} = \frac{k_t}{R_m} v_m - \frac{k_t k_m}{R_m}\dot{\theta} \tag{9}$$

$$(J_r + m_p r^2)\ddot{\theta} + \frac{k_t k_m}{R_m}\dot{\theta} - m_p l r \ddot{\alpha} + b_r \dot{\theta} = \frac{k_t}{R_m} v_m \tag{10}$$

To make the equations easier to follow and to allow for extended analysis, we create a signal flow diagram in the phase variable canonical form, where the state variables are the results of integrators.

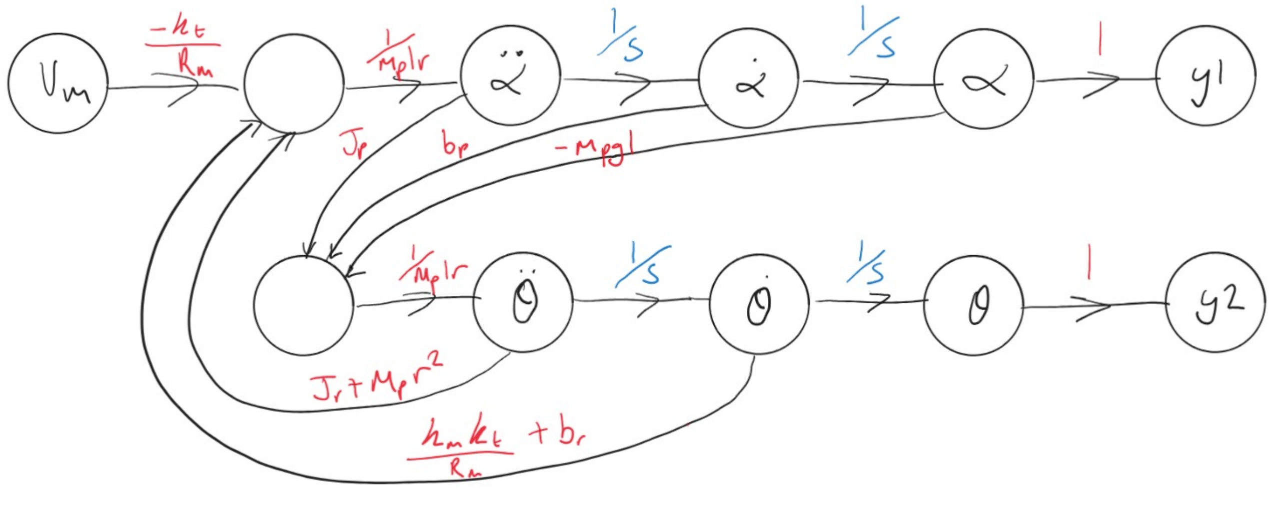


Figure 3: Signal flow diagram for the system

We can see from this diagram that there is an algebraic loop that must be solved. Let us define the following constants:

$$b_0 = \frac{1}{m_p I r} \qquad b_1 = J_p \qquad b_2 = b_p \qquad b_3 = -m_p g I$$

$$b_4 = \frac{k_m k_t}{R_m} + b_r \qquad b_5 = J_r + m_p r^2 \qquad b_6 = -\frac{k_t}{R_m}$$

We substitute these constants into equation 2 and rearrange in terms of $\ddot{\theta}$:

$$\ddot{\theta} = b_0(b_1\ddot{\alpha} + b_2\dot{\alpha} + b_3\alpha) \tag{11}$$

We also substitute the constants into equation 10 and rearrange in terms of $\ddot{\alpha}$:

$$\ddot{\alpha} = b_0(b_5\ddot{\theta} + b_4\dot{\theta} + b_6 v_m) \tag{12}$$

We then substitute the respective equations into each other so we only have 1 second order term in each. First we substitute the equation for $\ddot{\alpha}$ into the equation for $\ddot{\theta}$ (equation 12 into equation 11):

$$\ddot{\theta} = b_0(b_1 b_0(b_5\ddot{\theta} + b_4\dot{\theta} + b_6 v_m) + b_2\dot{\alpha} + b_3\alpha) \tag{13}$$

$$\ddot{\theta} = b_0^2 b_1 b_5 \ddot{\theta} + b_0^2 b_1 b_4 \dot{\theta} + b_0^2 b_1 b_6 v_m + b_0 b_2 \dot{\alpha} + b_0 b_3 \alpha \tag{14}$$

$$\ddot{\theta}(1 - b_0^2 b_1 b_5) = b_0^2 b_1 b_4 \dot{\theta} + b_0^2 b_1 b_6 v_m + b_0 b_2 \dot{\alpha} + b_0 b_3 \alpha \tag{15}$$

$$\ddot{\theta} = \frac{b_0^2 b_1 b_4 \dot{\theta} + b_0^2 b_1 b_6 v_m + b_0 b_2 \dot{\alpha} + b_0 b_3 \alpha}{1 - b_0^2 b_1 b_5} \tag{16}$$

Then we substitute the equation for $\ddot{\theta}$ into the equation for $\ddot{\alpha}$ (equation 11 into equation 12):

$$\ddot{\alpha} = b_0(b_5 b_0(b_1\ddot{\alpha} + b_2\dot{\alpha} + b_3\alpha) + b_4\dot{\theta} + b_6 v_m) \tag{17}$$

$$\ddot{\alpha} = b_0^2 b_5 b_1 \ddot{\alpha} + b_0^2 b_5 b_2 \dot{\alpha} + b_0^2 b_5 b_3 \alpha + b_0 b_4 \dot{\theta} + b_0 b_6 v_m \tag{18}$$

$$\ddot{\alpha}(1 - b_0^2 b_5 b_1) = b_0^2 b_5 b_2 \dot{\alpha} + b_0^2 b_5 b_3 \alpha + b_0 b_4 \dot{\theta} + b_0 b_6 v_m \tag{19}$$

$$\ddot{\alpha} = \frac{b_0^2 b_5 b_2 \dot{\alpha} + b_0^2 b_5 b_3 \alpha + b_0 b_4 \dot{\theta} + b_0 b_6 v_m}{1 - b_0^2 b_1 b_5} \tag{20}$$

With the second order terms fully separated, we can define our states to construct the state space representation ($\dot{x} = Ax + Bu$):

$$\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \\ \ddot{\theta} \\ \ddot{\alpha} \end{bmatrix} = A \begin{bmatrix} \theta \\ \alpha \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} + Bu \tag{21}$$

In this state space model, our input u is the voltage v_m. Hence we can construct the full state space model:

$$\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \\ \ddot{\theta} \\ \ddot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b_0 b_3}{1-b_0^2 b_1 b_5} & \frac{b_0^2 b_1 b_4}{1-b_0^2 b_1 b_5} & \frac{b_0 b_2}{1-b_0^2 b_1 b_5} \\ 0 & \frac{b_0^2 b_3 b_5}{1-b_0^2 b_1 b_5} & \frac{b_0 b_4}{1-b_0^2 b_1 b_5} & \frac{b_0^2 b_2 b_5}{1-b_0^2 b_1 b_5} \end{bmatrix} \begin{bmatrix} \theta \\ \alpha \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{b_0^2 b_1 b_6}{1-b_0^2 b_1 b_5} \\ \frac{b_0 b_6}{1-b_0^2 b_1 b_5} \end{bmatrix} u \tag{22}$$

From the state space model, we can determine the following matrices, where $(\cdot)_c$ is the matrix in continuous time:

$$A_c = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{b_0 b_3}{1-b_0^2 b_1 b_5} & \frac{b_0^2 b_1 b_4}{1-b_0^2 b_1 b_5} & \frac{b_0 b_2}{1-b_0^2 b_1 b_5} \\ 0 & \frac{b_0^2 b_3 b_5}{1-b_0^2 b_1 b_5} & \frac{b_0 b_4}{1-b_0^2 b_1 b_5} & \frac{b_0^2 b_2 b_5}{1-b_0^2 b_1 b_5} \end{bmatrix} \quad \text{and} \quad B_c = \begin{bmatrix} 0 \\ 0 \\ \frac{b_0^2 b_1 b_6}{1-b_0^2 b_1 b_5} \\ \frac{b_0 b_6}{1-b_0^2 b_1 b_5} \end{bmatrix} \tag{23}$$

For our output equation $y = Cx + Du$, we have the following C matrix to extract $\theta, \alpha, \dot{\theta}$ and $\dot{\alpha}$:

$$C_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

We choose this output matrix because although our primary objective is to achieve the balancing of the pendulum (steady state $\alpha = 0$), we have a physical limits: a plug in the way of the servo limiting θ. Therefore driving our θ towards 0 naturally helps us avoid this limit. The other states at a balanced inverted pendulum's unstable equilibrium are having 0 angular velocities, thus we would like to drive the other 2 states to 0, meaning that if we go by reference tracking methodology, we would like all of our states in the output.

There is no D matrix. We do not account for disturbances in this model, as there are no appreciable disturbances in the running conditions of our experiments.

Then, we use an Euler discretisation to convert the continuous-time matrices to discrete-time, where $(\cdot)_d$ is the discretized matrix and t_s is the time-step ($t_s = 0.002$):

$$A_d = I + t_s A_c \tag{25}$$

$$A_d = \begin{bmatrix} 1 & 0 & t_s & 0 \\ 0 & 1 & 0 & t_s \\ 0 & t_s \frac{b_0 b_3}{1-b_0^2 b_1 b_5} & 1 + t_s \frac{b_0^2 b_1 b_4}{1-b_0^2 b_1 b_5} & t_s \frac{b_0 b_2}{1-b_0^2 b_1 b_5} \\ 0 & t_s \frac{b_0^2 b_3 b_5}{1-b_0^2 b_1 b_5} & t_s \frac{b_0 b_4}{1-b_0^2 b_1 b_5} & 1 + t_s \frac{b_0^2 b_2 b_5}{1-b_0^2 b_1 b_5} \end{bmatrix} = \begin{bmatrix} 1.0000 & 0 & 0.0020 & 0 \\ 0 & 1.0000 & 0 & 0.0020 \\ 0 & 0.1103 & 0.9909 & -0.0004 \\ 0 & 0.3372 & -0.0090 & 0.9989 \end{bmatrix} \tag{26}$$

$$B_d = t_s B_c \tag{27}$$

$$B_d = \begin{bmatrix} 0 \\ 0 \\ t_s \frac{b_0^2 b_1 b_6}{1-b_0^2 b_1 b_5} \\ t_s \frac{b_0 b_6}{1-b_0^2 b_1 b_5} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.0412 \\ 0.0407 \end{bmatrix} \tag{28}$$

The output equation matrices remain unchanged:

$$C_d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

1.3 MPC implementation

MPC implementation was in 2 separate designs: a reference tracking pure state space controller, and a stabilising or state tracking augmented state space controller. Both controllers used a quadratic optimiser, the OSQP solver. This was one of the optimisers included on the Yalmip documentation that was performant and free to use.

The initial design was a reference tracking pure state controller as this is the simplest to implement.

We take the parameters of the MPC algorithm and first produce our controller. We create yalmip decision variables of our state X and our input U, each with the corresponding dimensions of 4 rows and N+1 columns for state, and 1 row and N+1 columns for the input. Then for each timestep N we set the constraints, the first being the discrete time equation for progressing the state space $x(t+k+1) = A_d x(t+k) + Bu(t+k)$. Then we add the constraint for control input magnitude, to setting a hard constraint for input voltage between -10V and 10V.

This model had a state constraint, to keep α within 10 degrees of the pendulum, in an attempt to keep our controller away from infeasible states. There was also a calculated input rate limit to attempt to smooth the input, avoiding high oscillations.

All references were set to 0, and while there was no consistent steady state error, RMS steady state error was high due to large oscillations.

Overall this method worked well at keeping the pendulum from falling, although there were major oscillations that did not converge at steady state, of around 5-9 degrees from this controller. A smaller state limit was infeasible many times to keep so this approach, while being a good start, was not the best approach.

A second approach was to use an augmented state space. Rather than using a full velocity form the state space was augmented with a 5th state being the previous control input. From this we could optimise for state X and delta U, allowing to keep some of the simplicity of a pure state space while better smoothing our input and allowing for fine grained control over input characteristics.

The new augmented state space had the following state, which we represented as a 5x1 sdpvar:

$$x_{aug} = \begin{bmatrix} \theta & \alpha & \dot{\theta} & \dot{\alpha} & u_{k-1} \end{bmatrix}^T \tag{30}$$

We represented the new augmented matrices A_{aug} and B_{aug} as a 5x5 matrix and a 5x1 matrix respectively:

$$A_{aug} = \begin{bmatrix} A_d & B_d \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad B_{aug} = \begin{bmatrix} B_d \\ 1 \end{bmatrix} \tag{31}$$

The new augmented state space equation is then:

$$x_{aug}(t+k+1) = A_{aug} x_{aug}(t+k) + B_{aug} \Delta u(t+k) \tag{32}$$

With this approach, not much change was needed architecturally, but the control input could be smoothed out, and the previous input could be included in the cost function. This meant that our Q matrix weighting could penalise high "previous" control inputs and regulate our system to use the minimum control input needed to achieve our goal. This led to a system with much less overshoot, less jittery or oscillatory motion that still kept a fast response time and next to no steady state error.

To keep our hard voltage limit, the current input u was calculated from our Δu and previous u inside the optimiser.

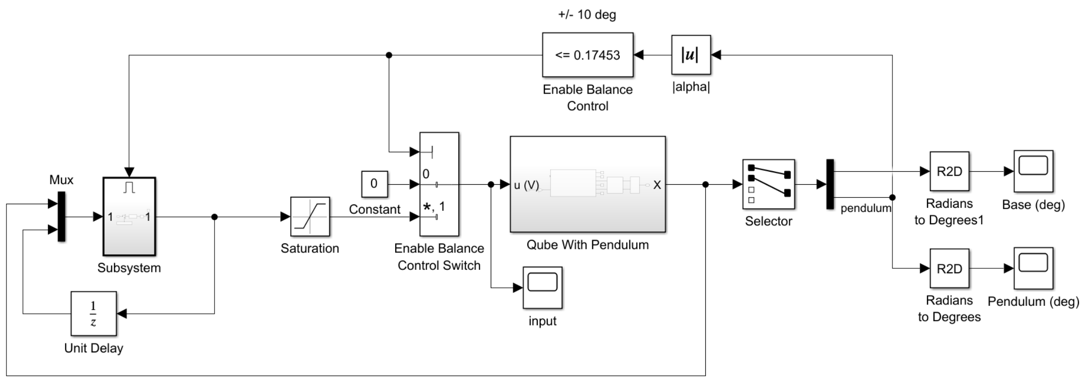


Figure 4: MPC Controller Simulink Diagram

```
function u_out = quarc_mpc(x_cur_in)
    tic  % to measure computation time

    % save the following for optimisation purposes
    persistent controller x_aug delta_u constraints objective parameters_in solutions_out P;
    persistent A_aug B_aug N Q R nx ndeltau;

    if isempty(A_aug)
        % initializing constants
        ts = 0.002; Rm = 7.5; kt = 0.042; km = 0.042; mr = 0.095; r = 0.085; Jr = (mr*r^2)/3;
        br = 1e-3; mp = 0.024; Lp = 0.129; I = Lp/2; Jp = (mp*Lp^2)/3; bp = 5e-5; g = 9.81;

        b0 = 1/(mp*I*r); b1 = Jp; b2 = bp; b3 = -mp*g*I;
        b4 = ((km*kt)+(br*Rm))/Rm; b5 = Jr+mp*r^2; b6 = -kt/Rm;

        den = 1/(1-(b0^2 * b1 * b5));
        t1 = den*b0*b3; t2 = den*b0^2 * b1 * b4; t3 = den*b0*b2; t4 = den*b0^2 * b6 * b1;

        a1 = den*b0^2 * b3 * b5; a2 = den*b0*b4; a3 = den*b0^2 * b2 * b5; a4 = den*b0*b6;

        % A, B matrices from differential equations
        A = [0 0 1 0;
             0 0 0 1;
             0 t1 t2 t3;
             0 a1 a2 a3];
        B = [0;0;t4;a4];

        % discretised with euler method
        Ad = eye(4)+ts*A;
        Bd = ts*B;

        A_aug = [Ad Bd; % augmented state space with u{k-1} for delta u
                 0 0 0 0 1];
        B_aug = [Bd ; 1];

        nx = 5; % Number of states (augmented)
        ndeltau = 1;

        % ideal values for Q, R, and N
        Q = [1 0 0 0 0;          % slowly center pendulum
             0 1e5 0 0 0;        % penalise angular deviation of pendulum most
             0 0 1 0 0;
             0 0 0 1 0;
             0 0 0 0 1e3];       % penalise magnitude of previous input to get least possible input
                                   signal for all steps

        R = 2; % low penalisation of gradient, we want a fast but smooth response
        N = 50; % prediction horizon
    end

    % enforce state as first 5 in input vector
    x_cur_in = (x_cur_in)
    x_cur = x_cur_in(1:5);

    if isempty(controller)
        % 2 decision variables: augmented state and change in input
        delta_u = sdpvar(repmat(ndeltau,1,N),repmat(1,1,N));
        x_aug = sdpvar(repmat(nx,1,N+1),repmat(1,1,N+1));

        % DARE for terminal cost
        [K P ev] = dlqr(A_aug,B_aug,Q,R);

        % steady state tracking reduces to stabilisation as we want all states to 0
        % xss is therefore zeros(5,1)

        constraints = [];
        objective = 0;
        for k = 1:N
            % constraints
            objective = objective + x_aug{k}'*Q*x_aug{k} + delta_u{k}'*R*delta_u{k};
            constraints = [constraints, x_aug{k+1} == A_aug*x_aug{k}+B_aug*delta_u{k}];
            constraints = [constraints, -2 <= x_aug{k}(1) <= 2]; % limiting base angle (always)
            constraints = [constraints, -2 <= x_aug{k}(2) <= 2]; % limitng pendulum angle
            constraints = [constraints, -3.5 <= x_aug{k}(3) <= 3.5]; %limitng base speed
            constraints = [constraints, -2.8 <= x_aug{k}(4) <= 2.8]; % limitng angle speed
            u_k = x_aug{k}(5) + delta_u{k}; % u_k = u_{k-1} + Delta_u_k (always)
            constraints = [constraints, -10 <= u_k <= 10]; % output constraint (always)
        end
        % defining objective with terminal cost (for no terminal cost, comment out second part)
        objective = objective + x_aug{N+1}'*P*x_aug{N+1};

        parameters_in = {x_aug{1}};
        solutions_out = {[delta_u{:}], [x_aug{:}]};
        % using "quadprog" solver for optimization problem, osqp can also be used by changing this
        controller = optimizer(constraints, objective,sdpsettings('solver','quadprog'),
                                 parameters_in,solutions_out);
    end

    inputs = {x_cur};
    [solutions,diagnostics] = controller{inputs};
    U = solutions{1};
    % for infeasibility
    if diagnostics == 1
        disp('The problem is infeasible');
        u_out = 0;
    else
        % if its feasible, update
        u_out(1) = U(1) + x_cur(5); % add delta u to previous u for new input
    end
    u_out(2) = toc; % output computation time
end
```

Listing 2: Fully commented MPC Controller Function with model construction, YALMIP variables, constraints, objective, solver settings, terminal cost, input magnitude limits, and input-rate limits

2 Experimental Design

2.1 PID vs MPC Controllers

2.1.1 Experiment Results

In this experiment, we aim to quantify the effectiveness of our optimally tuned PID and MPC controllers (codes and diagrams shown previously) in terms of key performance metrics: rise time, settling time, overshoot, input magnitude, steady state error. We will also attempt to display the performance of both controllers in terms of stability and constraint handling.

We will compare each controller's performance in the QLabs simulated environment, using the angle provided by the "lift arm" button as our starting angular displacement.

Note: To calculate our metrics, we wanted to observe the performance once each controller had become active (when $|\alpha| < 10$). To do this, we found the first timestep once the controller was active and started analysing the pendulum angle from there. Finally, we added 10 to each of the pendulum angles, so that our plot would start from $(0, 0)$ to make it easier to visualize results. Hence, in our pendulum response plots, the settling angle will be 10 degrees instead of zero degrees, as shown in figures 5 & 6.

Metric	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
QLabs PID	0.12	0.16	3.45	0.17	1.10
QLabs MPC	0.12	1.82	12.50	-0.20	0.34

Table 1: Performance metrics of PID vs MPC scenarios

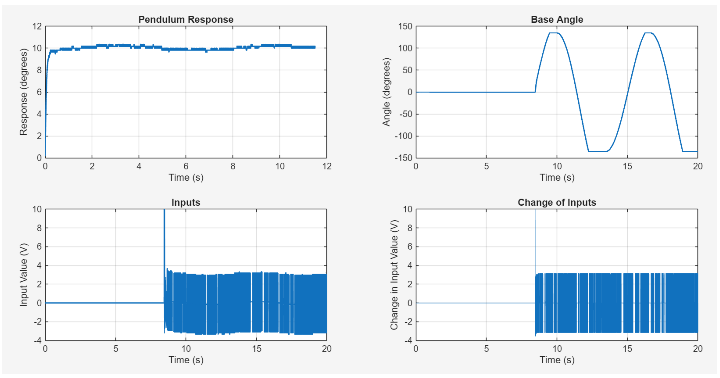


Figure 5: Plots of $\alpha(t), \theta(t), u(t)$ and $\Delta u(t)$ for PID Controller

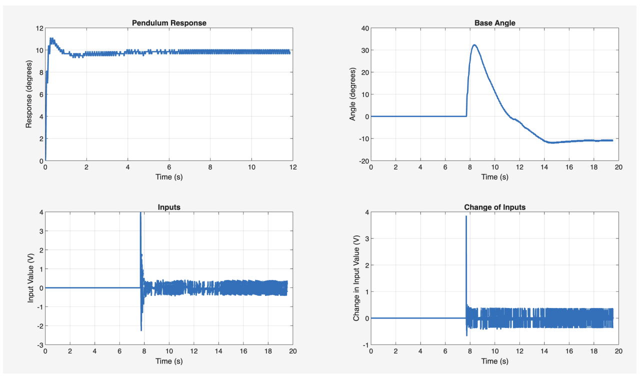


Figure 6: Plots of $\alpha(t), \theta(t), u(t)$ and $\Delta u(t)$ for MPC Controller

2.1.2 Comments on Results

As seen from table 2 and figures 5 and 6, we can observe multiple differences between the results produced by the PID and MPC controller.

Firstly, in terms of constraint handling, MPC controllers can implement constraints into the optimizer, and so the Optimal Control Problem aims to find the most feasible solution while adhering to all of the constraints. However, PID controllers do not handle any constraints, which can make them unsuitable in some real-life applications. In this case, the inverted pendulum requires constraints, such as limiting the input voltage between -10V and 10V for safety and limiting the base velocity due to torque limits. Therefore, the MPC controller is better suited for this inverted pendulum, as we can implement these constraints explicitly into the optimizer itself, but we cannot do so with the PID controller.

Generally, stability guarantees the system's trajectory stays bounded and converges to the equilibrium. MPC controllers usually do this by adding a terminal constraint into the optimizer, which forces the system's state to a zero state (equilibrium) at the end of the prediction horizon, or by adding a terminal cost to the cost function, to model the infinite-horizon cost which encourages convergence to the equilibrium. However, since PID controllers do not consider constraints and the PID control function only depends on the feedback error, they do not ensure closed-loop stability, so they could cause unstable system responses. In our inverted pendulum, we have implemented a terminal cost by adding the term $x^T P x$ to our objective function, where P is the solution to the DARE equation. This give us the optimal LQR solution, so we have ensured that all of our states are stable for the duration of each simulation. However in our PID controller, we cannot add this terminal cost, so for some values of K_p, K_i and K_d, the pendulum acts unstably and rotates rapidly indefinitely. Therefore the MPC controller is more suitable for the inverted pendulum as it produces a completely stable response.

In terms of quantitative metrics, we can observe the responses from both controllers, from the plots and the table. For similarities, we can see the rise times for both controllers are the same (0.12 seconds), so they both perform very quickly once active. The magnitudes of the steady state errors are also very similar and quite low, so both controllers are accurate. These steady state errors may be due to noise in the QLabs pendulum.

The biggest differences are the overshoot % and the settling time, as we can observe from the plots and the table that the PID response settles much faster with much less overshoot than the MPC response. This may be because the MPC handles constraints on the base as well as change in the input voltage, preventing aggressive actions that could violate these limits. Hence the MPC controller's response is slower, but ensures safety and feasibility. This links to the next largest difference – the RMS input is much lower for the MPC response than the PID response. This is proved from the plots of $u(t)$, because the PID's maximum input is 10V whereas the MPC's maximum input is only 4V. In addition, we see that $\Delta u(t)$ has a much smaller variance for the MPC controller than the PID controller. From this, we can deduce that the MPC's response is much more feasible and stable for the motors, due to the lower input inputs and input changes. Finally, from the plots of $\theta(t)$, we can determine that the MPC response is more desired, since the PID controller causes the base to oscillate indefinitely, whereas the MPC controller causes the base to settle to a constant value. Therefore, in terms of motor efficiency and power consumption, the MPC controller is more optimal.

In conclusion, the PID controller provides a faster response with less overshoot, but the MPC controller is more suitable because it handles several constraints, guarantees stability with a terminal cost, and ensures feasibility by requiring lower input voltages and input changes.

2.2 Investigating the Effect of Q/R

2.2.1 Experiment Results for Investigation of Q

The ideal values for Q and R we found while tuning are:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1e5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1e3 \end{bmatrix} \quad \text{and} \quad R = 2 \tag{33}$$

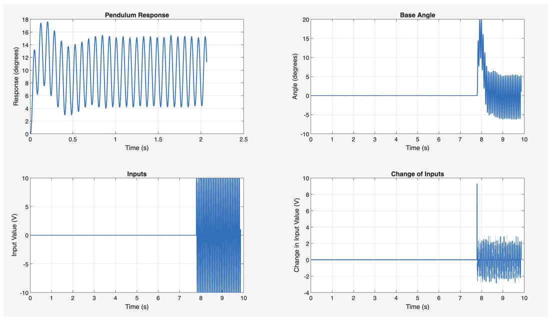
From there, we changed the values of Q to see their effects on performance metric. Q is a square, diagonal matrix with the elements defined in the "diag" column in the table. In a cell, a NaN value represents that configuration not creating that metric, usually caused by the pendulum not settling. For this experiment, the prediction horizon (N) and R will stay constant at 50 and 2 respectively. The relevant sections of code which we edit and the results are shown below:

```matlab
function u_out = quarc_mpc(x_cur_in)
    ...
    if isempty(A_aug)
        ...
        Q = [1 0 0 0 0;          % slowly center pendulum
             0 1e5 0 0 0;        % penalise angular deviation of pendulum most
             0 0 1 0 0;
             0 0 0 1 0;
             0 0 0 0 1e3];       % penalise magnitude of previous input to get least possible input
                                   signal for all steps
        R = 2; % low penalisation of gradient, we want a fast but smooth response
        ...
    end
    ...
end
```

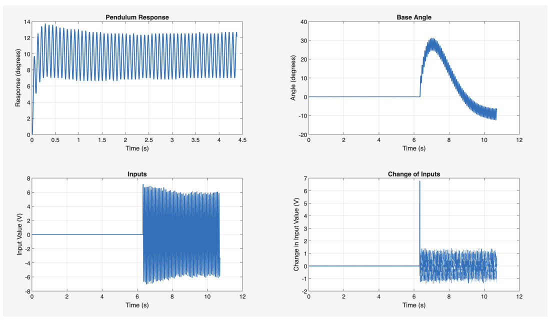
Listing 3: Relevant section of code to change Q and R

Q	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
diag(1,1e5,1,1,1e3)	0.12	1.86	12.50	0.0066	0.35
diag(1,1,1,1,1e3)	0.12	1.66	14.29	0.13	0.38
diag(1,1e5,1,1,1)	NaN	NaN	NaN	NaN	9.73
diag(1,1e5,1,1,100)	0.03	NaN	39.29	NaN	4.57
diag(1e5,1e5,1,1,1e3)	4.41	NaN	5.61	NaN	7.84
diag(1,1e5,1e5,1,1e3)	0.02	NaN	76	NaN	9.66
diag(1,1e5,1,1e5,1e3)	NaN	NaN	NaN	NaN	6.88
diag(1,1e5,1,1,1e5)	0.12	2.03	14.29	0.09	0.62
diag(1,1e7,1,1,1e3)	0.02	NaN	76.79	NaN	9.32

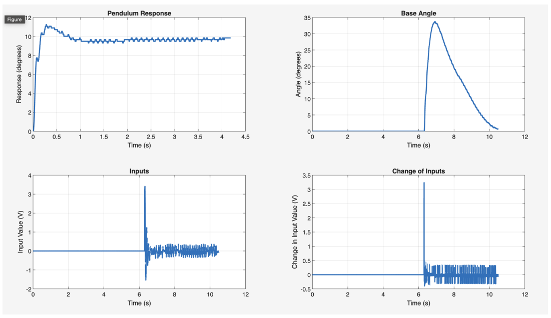
Table 2: Performance metrics of our MPC controller when the values of Q are changed.



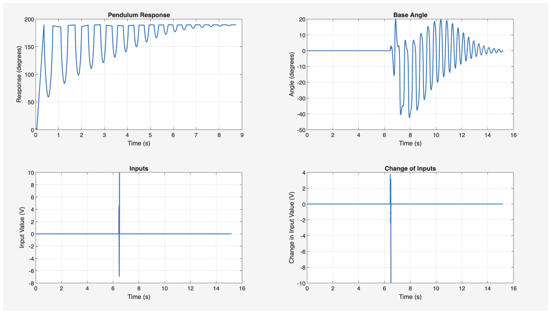
(a) Q=diag(1e5,1e5,1,1,1e3)



(b) Q=diag(1,1e5,1,1,100)



(c) Q=diag(1,1e5,1,1,1e5)



(d) Q=diag(1,1e5,1,1e5,1e3)

Figure 7: Plots of $\theta(t)$, $\dot{\theta}(t)$, $u(t)$ and $\Delta u(t)$ for different Q matrices

2.2.2 Comments on Results

As confirmed before, diag(1,1e5,1,1,1e3) produced the best results overall as the most emphasis is put on getting the desired pendulum deflection angle α while having less emphasis on the base rotary angle θ allowing it to swing more freely to get the upright pendulum pointing directly up and being penalised less for not being at the perfect position. This can be seen when we increased the θ tracking variable to match α's in row 5 (creating Q = diag (1e5, 1e5,1,1,1e3) in Figure 7a) when the pendulum did not settle and took a lot longer to rise and then had a greater overshoot compared to our best values as trying to get θ to 0 is not a good strategy. Having the α tracking variable be high was important as well to help eliminate the steady state error as when Q was diag(1,1,1,1,1e3), everything stayed pretty constant apart from the steady state error becoming worse (0.0066 to 0.13). However, when this variable was too high, in diag(1,1e7,1,1,1e3), the overshoot was massive in 76.79 and the RMS was 9.32 as it was trying too hard to get the main pendulum angle correct and ignoring all other parameters, leading to the Pendulum not settling and failing.

The second largest Q values being for the previous input aided dramatically in smoothing the input and reducing the input signal at all time steps. This Q value was invaluable for producing stable movement as when we lowered it in rows 3 and 4 to 1 (diag(1,1e5,1,1,1)) and 100 (diag(1,1e5,1,1,100), Figure 7b) respectively the pendulum did not settle. Interestingly, when it was 100 in row 4 the rise time was 0.03 seconds compared to 0.12 when using the best values. However, the RMS input when q was 100 was 4.57 compared to the standard 0.37, representing that the pendulum reaches the desired angle quickly by just increasing the input as the q value is not high enough to limit the input as drastically. Although when that q value was increased to 1e5 (diag(1,1e5,1,1,1e5), Figure 7c) the results are still good, the main difference was that the RMS input was higher with the greater q value which seems contrary to what a higher q value should do, but what happens is that the pendulum needs to correct more aggressively as the initial start is too smooth and not enough voltage is given to the motors.

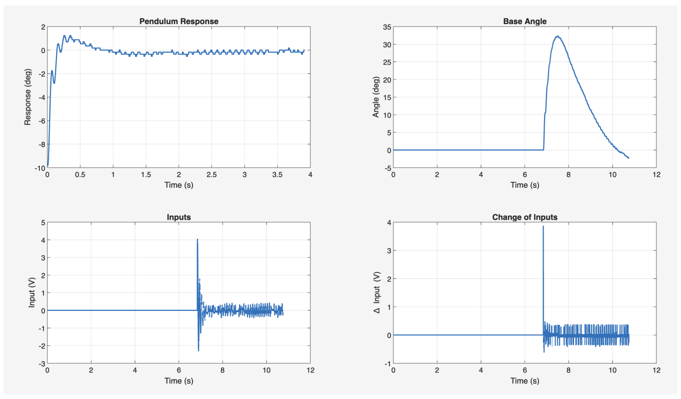
Increasing the velocity trackers to greater values such as 1e5 in rows 6 (diag(1,1e5,1e5,1,1e3)) & 7 (diag(1,1e5,1,1e5,1e3), Figure 7d) produced MPCs that did not settle at the desired upright angle. These trackers are trying to get the angular velocities as close to 0 as possible, but this reduces the velocity as the pendulum is trying to lift up, causing it to have more damping and not gain enough speed to reach the top (experiment 7) or stop the base too much that momentum carries the pendulum too far causing no settling and a big overshoot (experiment 6).

2.2.3 Experiment Results for Investigation of R

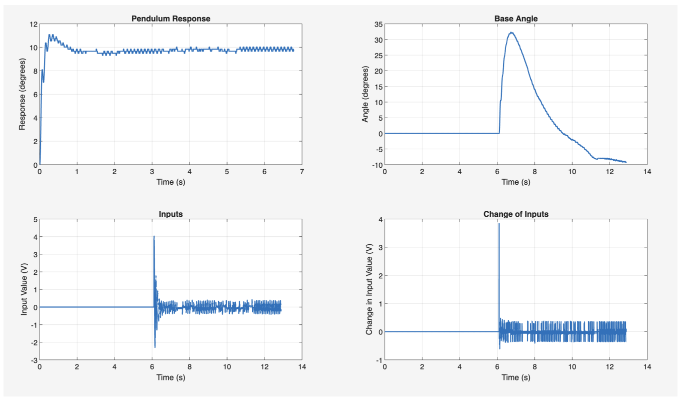
For this experiment, N was held constant at 50, and Q was held constant at diag(1, 1e5, 1, 1, 1e3).

R	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
0	0.12	2.04	13.5	0.09	0.41
2	0.12	2.02	12.50	0.002	0.39
10	0.12	2.05	14.73	0.19	0.48
50	0.12	2.07	13.8	0.11	0.36

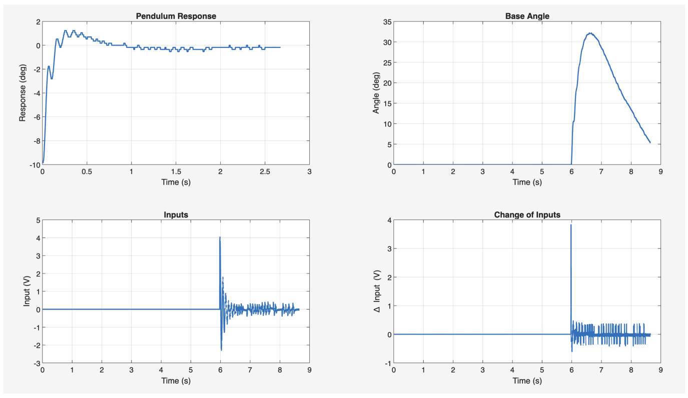
Table 3: Performance metrics of our MPC controller when the value of R is changed.



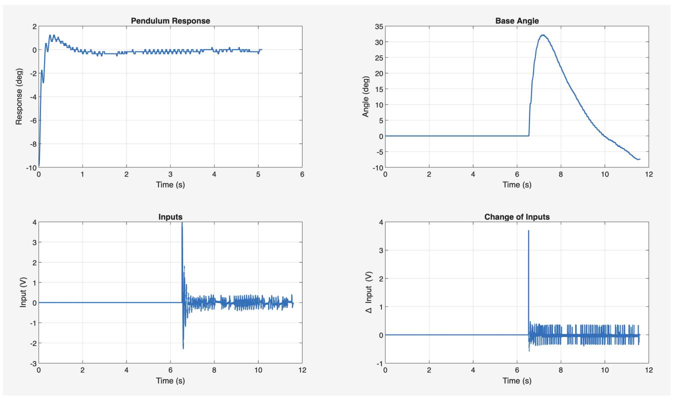
(a) R=0



(b) R=2



(c) R=10



(d) R=50

Figure 8: Plots of $\theta(t)$, $\dot{\theta}(t)$, $u(t)$ and $\Delta u(t)$ for different R values

2.2.4 Comments on Results

We found that the results when changing R demonstrated minor noticeable differences between experiments, our tuned value, R = 2 (Figure 8b), produced the best results as expected. The main difference between the different experiments was that when R = 2, there was the best steady state Error at 0.002, compared to all the others being around 0.1. The RMS input stayed fairly consistent between all experiments, signifying that this is the rough RMS needed to raise the pendulum, no matter what punishment there is on the control input through R. Rise time and settling time were the same for all runs, however overshoot was best when R = 2, even though the worst, R = 10 (Figure 8c), was only 2.23% greater, signifying when R = 2, the MPC had the most control.

2.3 Experiment 3: Prediction Horizon (N)

2.3.1 Experiment Results

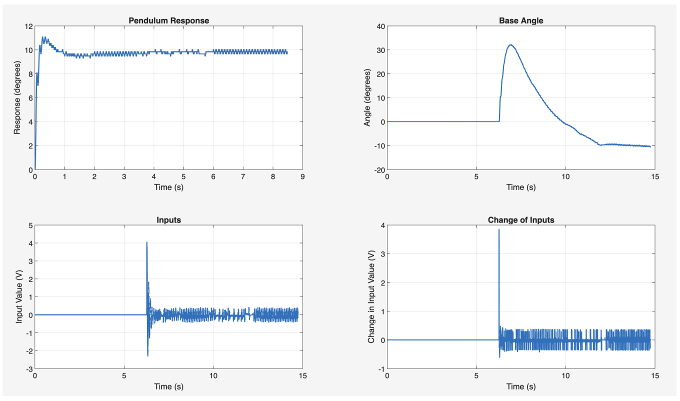
For this experiment, the matrices Q and R will stay constant at their optimal values.

```
function u_out = quarc_mpc(x_cur_in)
    ...
    if isempty(A_aug)
        ...
        N = 50; % prediction horizon
        ...
    end
    ...
end
```

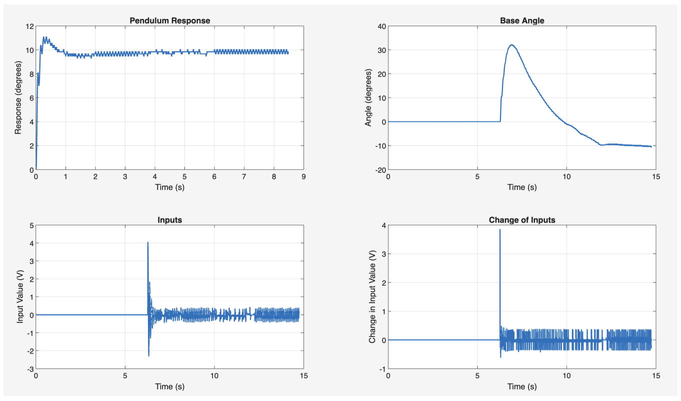
Listing 4: Relevant section of code to change N

N	Computation Time	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
2	0.0015	0.12	1.87	12.5	0.007	0.34
5	0.0019	0.12	1.83	12.5	0.004	0.36
10	0.0025	0.12	2.12	12.5	0.002	0.35
25	0.0042	0.12	1.86	12.5	0.002	0.35
50	0.0077	0.12	1.87	12.5	0.006	0.35
100	0.0171	0.12	1.59	12.5	0.002	0.33
300	0.0819	NaN	NaN	NaN	NaN	NaN
500	0.2408	NaN	NaN	NaN	NaN	NaN
1000	0.9913	NaN	NaN	NaN	NaN	NaN

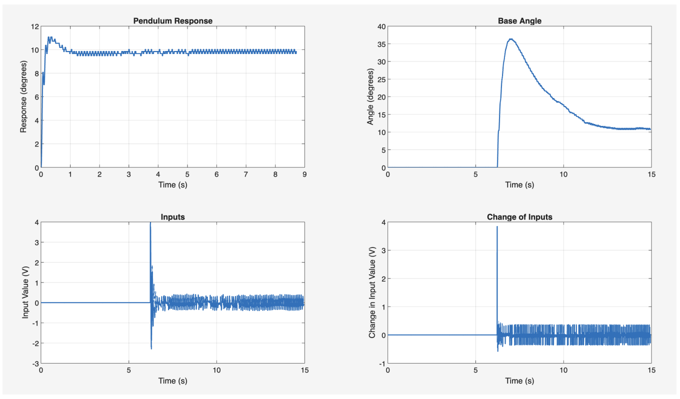
Table 4: Performance metrics for varying prediction horizons



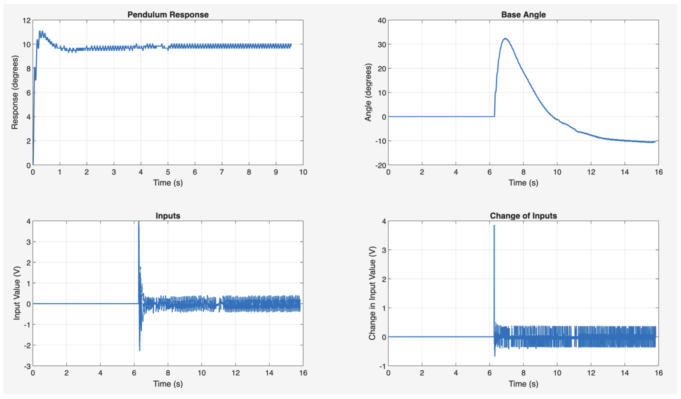
(a) N=2



(b) N=25



(c) N=50



(d) N=100

Figure 9: Plots of $\alpha(t)$, $\theta(t)$, $u(t)$ and $\Delta u(t)$ for different prediction horizons

2.3.2 Comments on Results

For each prediction horizon N, the controller was tested three times over 15 simulation seconds and the mean was taken to ensure reliability and consistency.

The computational time rises from 0.0015s at N=2 to almost 1s at N=1000, indicating a monotonic increase with prediction horizon. This trend is also illustrated in figure 8.

Across all prediction horizons, the rise time and the overshoot remain constant at 0.12 and 12.5% respectively. This indicates that the initial responsiveness of the system and the transient peaking are not influenced by the changes in the prediction horizon.

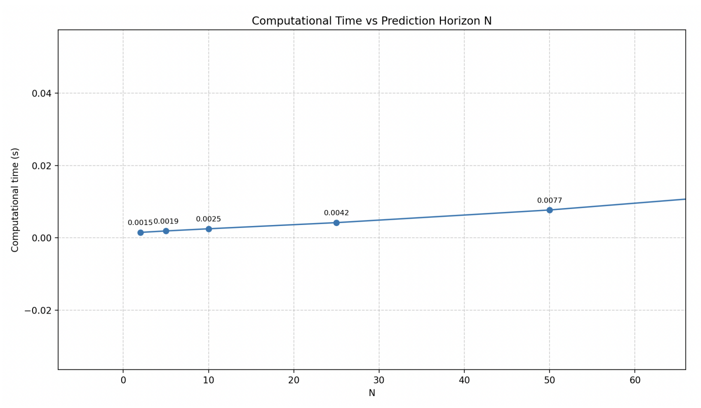
The RMS input and steady state error also remain relatively constant, with values in the ranges

[0.33, 0.36] and [0.002, 0.007] respectively. This indicates that varying prediction horizons does not significantly affect the average control effort or controller's ability to eliminate steady-state bias.

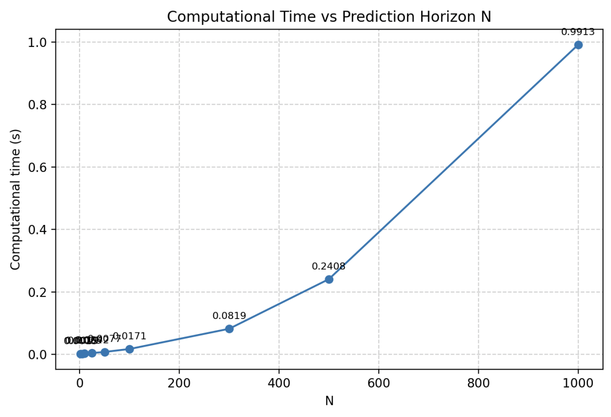
The settling time also remains relatively consistent, with values ranging from 1.59 to 2.12, without showing any pattern in response to different values of N. This suggests that the prediction horizon doesn't affect the settling time.

For values N=300 onwards, the controller produced NaN performance metrics and the simulation became unstable, indicating numerical or solver-related issues for any higher prediction horizon values. Nevertheless, computation time was still recorded prior to the controller failure.

Overall, the experiment results indicate that increasing dramatically the prediction horizon mostly increases the computational time without consistently beneficial changes for the system's performance. It also leads to numerical instability for very high values. In non-simulation conditions, the increased computational times can prove to make the ineffective for cases which require fast responses.



(a) Computational time for up to N=50



(b) Computational time up to N=1000

Figure 10: Plots of computational time for different prediction horizon values

2.4 Experiment 4: Terminal cost (P)

Terminal cost is used in MPC configuration to improve stability and feasibility even for short prediction horizons. To explore this further, the controller was tested for different predictions horizons, with and without a terminal cost for. The relevant code and results are detailed below.

```matlab
function u_out = quarc_mpc(x_cur_in)
    ...
    if isempty(controller)
        ...
        % DARE for terminal cost
        [K P ev] = dlqr(A_aug,B_aug,Q,R);
        ...
        objective = 0;
        ...
        for k = 1:N
            objective = objective + x_aug{k}'*Q*x_aug{k} + delta_u{k}'*R*delta_u{k};
            ...
        end
        % defining objective with terminal cost (for no terminal cost, comment out second part)
        objective = objective + x_aug{N+1}'*P*x_aug{N+1};
        ...
    end
    ...
end
```

Listing 5: Relevant code to add/remove the terminal cost

2.4.1 Experiment Results

N	Terminal Cost	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
2	yes	0.12	1.87	12.5	0.007	0.34
2	no	NaN	NaN	NaN	NaN	NaN
10	yes	0.12	2.12	12.5	0.002	0.35
10	no	NaN	NaN	NaN	NaN	NaN
45	yes	0.12	1.84	12.5	0.009	0.36
45	no	NaN	NaN	NaN	NaN	NaN
50	yes	0.12	1.87	12.5	0.006	0.35
50	no	0.12	8.40	7.14	0.30	0.29
60	yes	0.12	1.68	12.5	0.008	0.36
60	no	0.14	2.98	7.14	0.07	0.21

Table 5: Performance metrics of different predictions with and without terminal costs

2.4.2 Comments on Results

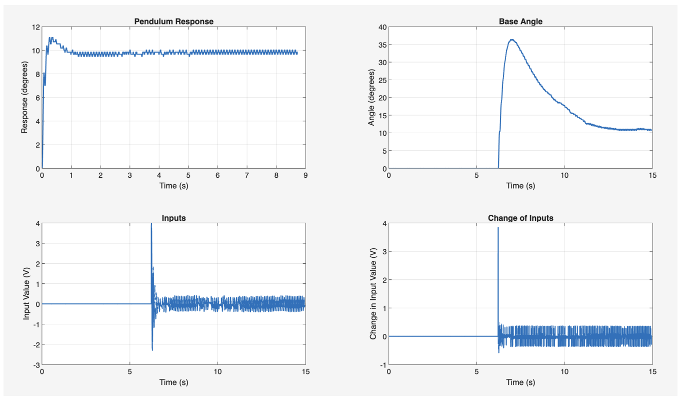
Table 4 illustrates the significant impact of the terminal cost on controller performance. When the terminal cost is not included, the controller frequently fails to stabilize the pendulum. This indicates loss of feasibility and highlights the significance of the terminal cost in guiding the predicted state towards the equilibrium while remaining within the defined constraints, especially for small prediction horizons.

For prediction horizons of 50 onwards, the controller managed to stabilise the inverted pendulum, but with degraded performance. The steady state error values for both N=50 and N=60 is significantly larger, and the settling times longer compared to the corresponding cases without terminal cost. This illustrates the controller's poorer performance when the terminal cost is omitted as illustrated in figure 9. However, the controller without the terminal cost, exhibits smaller overshoot and RMS input, indicating reduced control effort.

Increasing the prediction horizon improves performance in the absence of terminal cost as reflected in the improved behaviour at N=60 over N=50. This suggests that longer prediction horizons can partially compensate for the lack of terminal cost but at the expense of very large prediction horizons requiring increased computational effort.

For prediction horizons smaller than 50, although the controller ultimately failed, the performance improved as the horizon increased. For example, at N=25 the pendulum fell almost immediately after the MPC controller took over, whereas for N=45, the pendulum was initially stabilised, but this was achieved by allowing the base to rotate. Once the base angle constraint was reached, the controller could no longer satisfy all constraints, causing the optimisation problem to become infeasible and the run to fail. This is also illustrated in the corresponding response graphs of figure 10.

Overall, this experiment demonstrates that including a terminal cost is essential for a reliable and consistent controller performance, particularly at smaller prediction horizons and to eliminate infeasibility of the optimization problems.


(a) N=50 with terminal cost

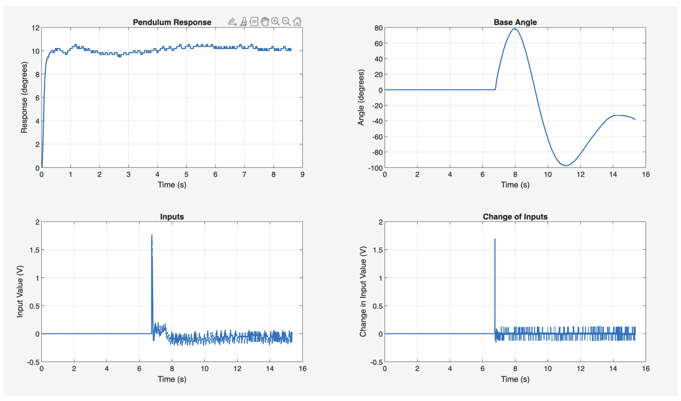
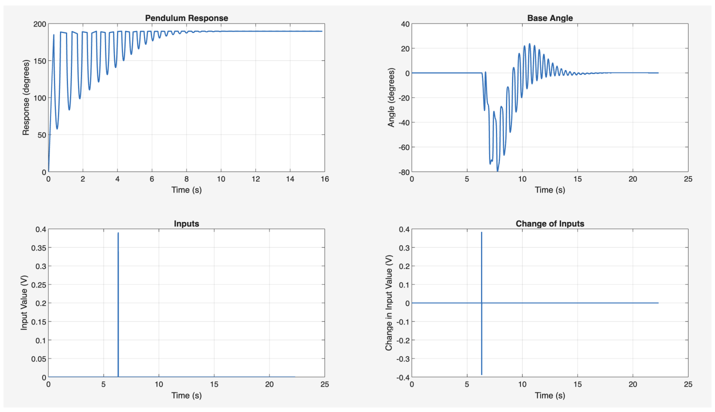

(b) N=50 without terminal cost

Figure 11: Plots of $\alpha(t)$, $\theta(t)$, $u(t)$ and $\Delta u(t)$ for N=50 with and without terminal cost


(a) N=25

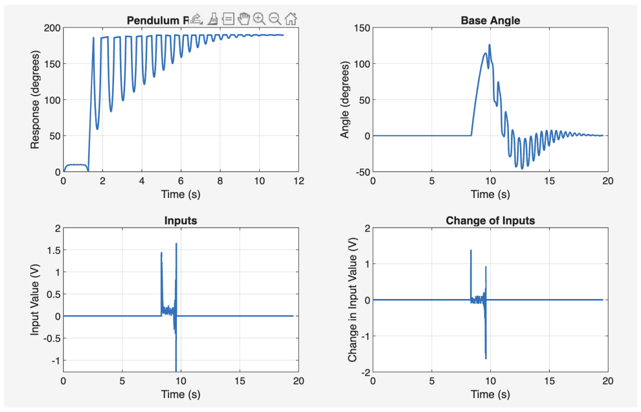

(b) N=45

Figure 12: Plots of $\alpha(t)$, $\theta(t)$, $u(t)$ and $\Delta u(t)$ for failed runs with no terminal cost

2.5 Experiment 5: Constraints

2.5.1 Experiment Results

In this experiment, we study how input magnitude and input-rate limits affect system behaviour. The relevant code to change the constraints and the results are shown below:

```
function u_out = quarc_mpc(x_cur_in)
    ...
    if isempty(controller)
        ...
        constraints = [];
        for k = 1:N
            constraints = [constraints, x_aug{k+1} == A_aug*x_aug{k}+B_aug*delta_u{k}];
            constraints = [constraints, -2 <= x_aug{k}(1) <= 2]; % limiting base angle (always)
            constraints = [constraints, -2 <= x_aug{k}(2) <= 2]; % limitng pendulum angle
            constraints = [constraints, -3.5 <= x_aug{k}(3) <= 3.5]; %limitng base speed
            constraints = [constraints, -2.8 <= x_aug{k}(4) <= 2.8]; % limitng angle speed
            u_k = x_aug{k}(5) + delta_u{k}; % u_k = u_{k-1} + Delta_u_k (always)
            constraints = [constraints, -10 <= u_k <= 10]; % output constraint (always)
        end
        ...
    end
    ...
end
```

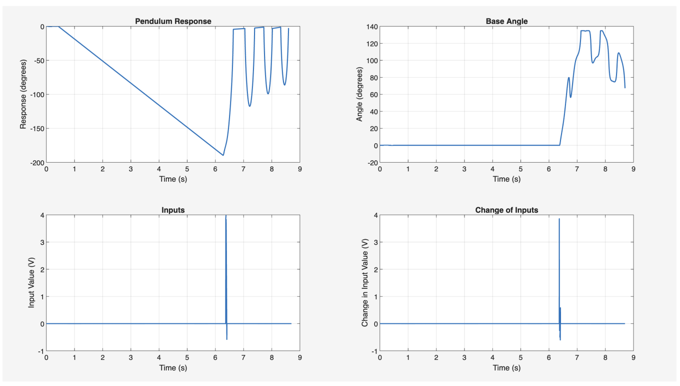
Listing 6: Relevant code to edit input and input rate constraints

Constraint	Rise Time	Settling Time	Overshoot	Steady State Error	RMS Input
Normal	0.12	1.86	12.5	0.006	0.35
output $[-100, 100]$	0.12	1.99	12.5	0.002	0.51
output $[-5, 5]$	0.12	2.2	12.5	0.19	0.52
output $[-0.2, 0.2]$	NaN	NaN	NaN	NaN	0.20
$\dot{\theta}\,[-0.5, 0.5]$	NaN	NaN	NaN	NaN	3.00
$\dot{\theta}\,[-3.2, 3.2]$	NaN	NaN	NaN	NaN	3.06
$\dot{\theta}\,[-3.5, 3.5]$	0.12	0.41	12.5	0.028	0.45
$\dot{\alpha}\,[-2.6, 2.6]$	NaN	NaN	NaN	NaN	3.06
$\dot{\alpha}\,[-2.8, 2.8]$	0.12	0.76	12.5	0.03	0.53

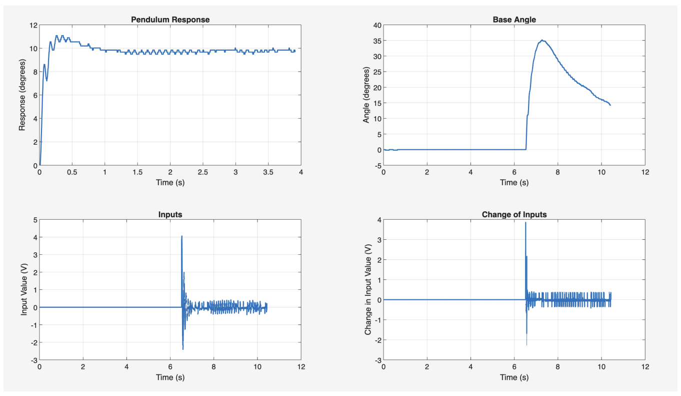
Table 6: Performance metrics across experiments with a variety of different constraints

2.5.2 Comments on Results

When we go from our normal model with only one constraint, θ must be within -0.2 and 0.2, to constraining the output to be in the range [-100, 100] there is no noticeable effect as the output never got close to the constraint, so the constraint is never needed. Constraining the output to be within [-5, 5] also had no significant effect other than the steady state error was slightly worse but still within an acceptable final range. However, when the output was constrained to [-0.2,0.2] the pendulum failed to lift, we know this is because the input cannot get high enough because the rms input becomes 0.2, the constraint limit, meaning the optimisation the MPC sees it getting up is by applying more torque through he motor. The RMS input for the previous constraints were around 0.5, considerably greater than 0.2, leading to the pendulum failing to reach the top.



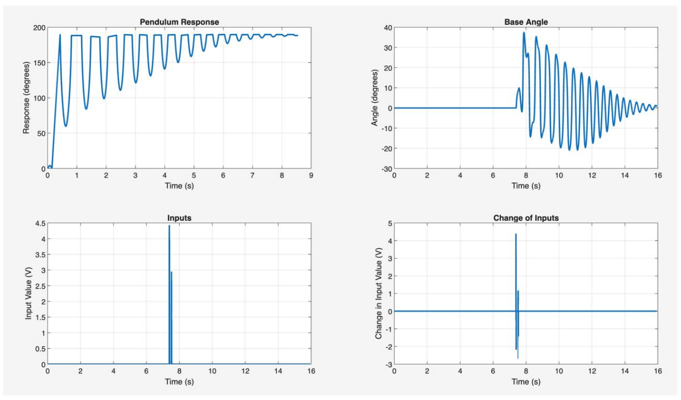
(a) Pendulum angle speed [-2.6, 2.6] fails



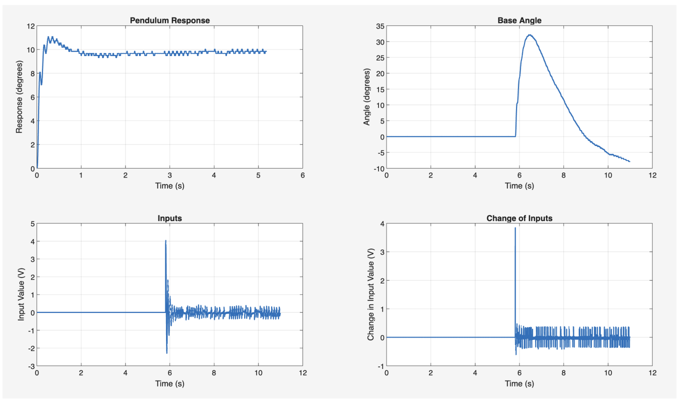
(b) Pendulum angle speed [-2.8, 2.8] successful

Figure 13: Varying $\dot{\alpha}$ and observing its impact on success/failure

Adding a constraint to the base angular velocity, $\dot{\theta}$, affected the feasibility of the controller. For the constraint for $\dot{\theta}$, [-0.5, 0.5] (Figure 14a), the pendulum failed to reach the top as there was not enough input velocity. However, when the constraint was slightly increased to be between -3.5 and 3.5, Figure 14b, the pendulum could successfully reach the top, signifying that the boundary between feasibility and not is there. The constraint in the base speed resulted in a much quicker settling time of 0.41s compared to 1.86s. The steady state error is slightly worse with constraints but still very good and acceptable at 0.028. The RMS input is higher but still not too high at 0.45, overall adding a constraint to $\dot{\theta}$ does not have a major noticeable impact on the system other than there being more oscillations.



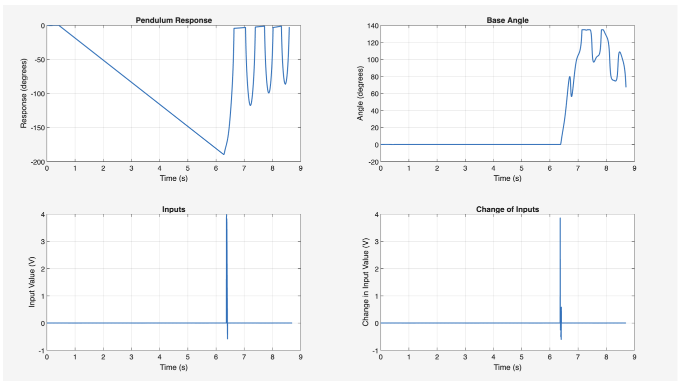
(a) $\dot{\theta}$ constrained to [-0.5, 0.5]



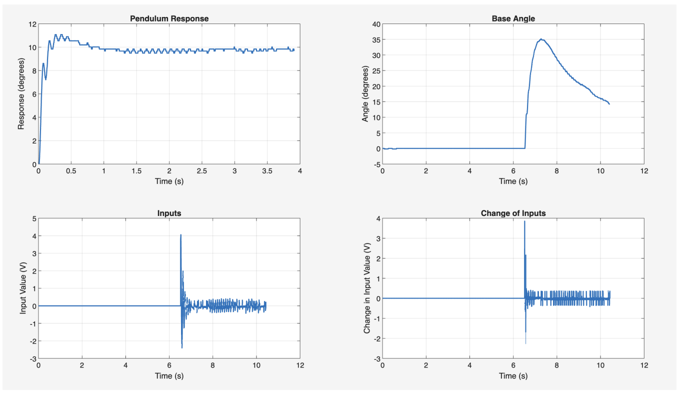
(b) $\dot{\theta}$ constrained to [-3.5, 3.5]

Figure 14: Varying $\dot{\theta}$ and observing its impact on metrics

Adding a constraint to the pendulum arm angular velocity, $\dot{\alpha}$, has very similar affects to adding constraints to $\dot{\theta}$, [-2.6, 2.6] (Figure 15a) failed to settle but [-2.8,2.8] (Figure 15b) worked with minor noticeable effects compared to our base model.



(a) $\dot{\alpha}$ constrained to [-2.6, 2.6]



(b) $\dot{\alpha}$ constrained to [-2.8, 2.8]

Figure 15: Varying $\dot{\theta}$ and observing its impact on metrics