

University of Wales Trinity Saint David

Wales Institute of Science and Art



Prifysgol Cymru

Y Drindod Dewi Sant

University of Wales

Trinity Saint David

MODELLING INDUSTRIAL CONTROL SYSTEMS IN MATLAB

Control Technology – ACMA6001

Student Name: Martin Ashe

Student Number: 2223791

Date: 21/07/25

Table of Contents

List of Figures.....	3
1. Introduction	4
2. Background	4
2.1 PID General Background.....	4
2.2 Cruise Control Standard	5
3. PID Tuning	7
3.1 Method.....	7
3.2 Results and Analysis.....	9
4. Sampling Time Impact.....	13
4.1 Method.....	13
4.2 Results and Analysis.....	13
5. Signal Delay Impact	16
5.1 Method.....	16
5.2 Results and Analysis.....	16
6. Number of Bits Impact.....	19
6.1 Method.....	19
6.2 Results and Analysis.....	19
7. Conclusion	22
8. Bibliography	23
9. Appendix	25

List of Figures

Figure 1: Cruise control PID block diagram adopted from Borase et al. (2021)	5
Figure 2: Upper and lower limits of acceleration according to ISO 15622:2018 (Duan,Schockenhoff and Koch, 2022).....	6
Figure 3: Upper and lower limits of jerk according to ISO 15622:2018 (Duan, Schockenhoff and Koch, 2022)	6
Figure 4: Binary to decimal conversion code.....	7
Figure 5: Code for acceleration and jerk calculations.....	8
Figure 6: Code for control output limits.....	8
Figure 7: Velocity response plot ($K_p = 2.1$, $k_i = 0$, $K_d = 0$)	9
Figure 8: Velocity response plot ($K_p = 2.1$, $k_i = 1$, $K_d = 0$)	9
Figure 9: Vehicle acceleration and jerk plots ($K_p = 2.1$, $k_i = 1$, $K_d = 0$)	10
Figure 10: Vehicle response and control effort plots ($K_p = 2.1$, $k_i = 1$, $K_d = 0.05$)....	10
Figure 11: Code for rate of change limiter on control output.....	11
Figure 12: Vehicle response and control effort plots ($K_p = 2.1$, $k_i = 1.2$, $K_d = 0.1$)...	11
Figure 13: Code for low-pass filter application to the demand signal	12
Figure 14: Vehicle response and control effort plots ($K_p = 2.5$, $k_i = 1.2$, $K_d = 0.07$). 12	
Figure 15: Vehicle response and control effort plots (sampling time = 0.2 s)	14
Figure 16: Vehicle response and control effort plots (sampling time = 0.5 s)	15
Figure 19: Vehicle response and control effort plots (signal delay = 0.5 s)	16
Figure 20: Vehicle response and control effort plots (signal delay = 1 s)	17
Figure 21: Vehicle response and control effort plots (signal delay = 1 s / reduced PID values).....	18
Figure 22: Vehicle response and control effort plots (bit resolution = 12-bit).....	20
Figure 23: Vehicle response and control effort plots (bit resolution = 6-bit).....	21

1. Introduction

The primary objective of this project was to design, implement, and evaluate a digital Proportional-Integral-Derivative (PID) controller for integration within a vehicle cruise control system. MATLAB was used to model the system dynamics and controller behaviour, and a predefined simulation script emulated a realistic velocity demand profile. The PID controller regulated the vehicle's velocity to accurately track the reference input, thereby demonstrating its effectiveness in maintaining speed under varying demand conditions. To enhance the realism of the simulation, digital effects such as quantisation, sampling, and time delay were incorporated to reflect the constraints of real-world embedded control systems.

Additional objectives included analysing the impact of PID gain tuning on system stability and responsiveness, while ensuring zero steady-state error. The study also investigated digital implementation factors that affect control performance, such as time step, bit resolution, and communication delay. Based on the simulation results, engineering recommendations were formulated to support the effective and robust implementation of digital control systems.

2. Background

2.1 PID General Background

PID controllers are widely used in industrial control systems to regulate temperature, speed, and pressure by minimising the error between a desired setpoint and the actual process variable. The controller provides a control signal by combining three terms based on the error signal:

- Proportional (P) Term – Produces a control signal that is proportional to the current error. This increases the system's responsiveness (Bucz and Kozáková, 2012).

- Integral (I) Term – Accumulates past errors over time using integration, helping to eliminate steady-state errors (Bucz and Kozáková, 2012).
- Derivative (D) Term – Reacts to the rate of change using differentiation, aiding in the damping of oscillations and reducing overshoot (Bucz and Kozáková, 2012).

A block diagram of the cruise control PID system is shown in Figure 1.

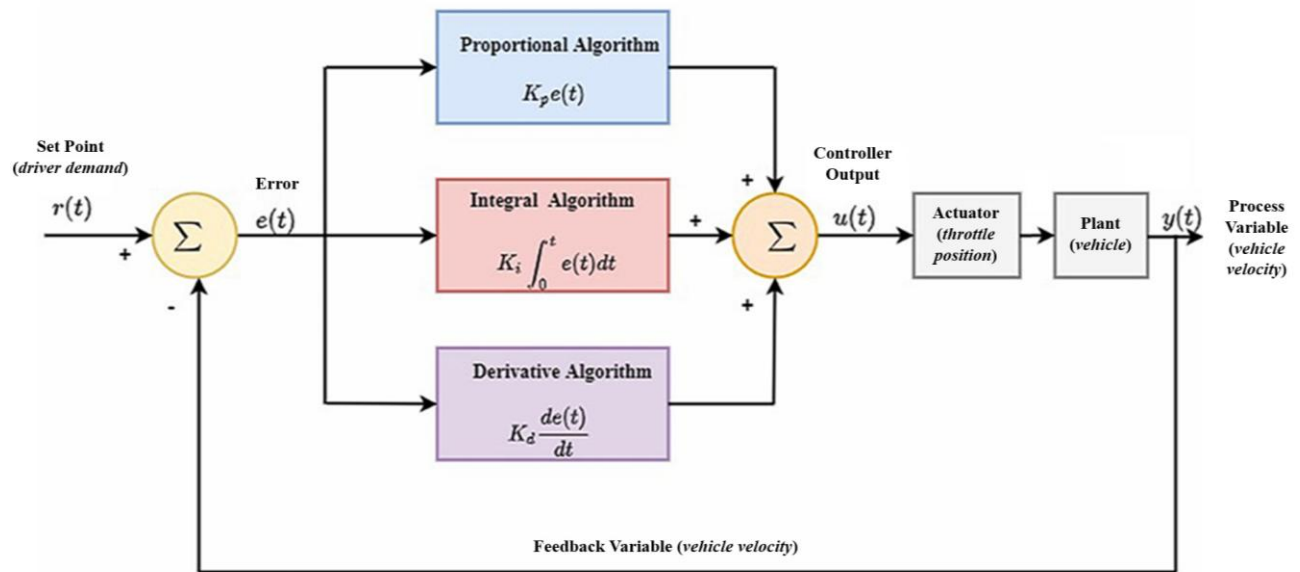


Figure 1: Cruise control PID block diagram adopted from Borase et al. (2021)

2.2 Cruise Control Standard

The ISO standard 15622:2018 outlines performance requirements and test procedures for Adaptive Cruise Control (ACC) systems in vehicles. Although this project focused on a basic cruise control system, many of its principles remained applicable (International Organization for Standardization, 2018). The acceleration and jerk limits shown in Figures 2 & 3 (Duan, Schockenhoff and Koch, 2022) are speed-dependent. However, since the demand scenarios in this project exceed 20 m/s, the following limits were applied:

- Maximum Acceleration: 2.0 m/s²
- Maximum Deceleration: -3.5 m/s²
- Maximum Jerk: 2.5 m/s²

- Maximum Jerk: -2.5 m/s^2

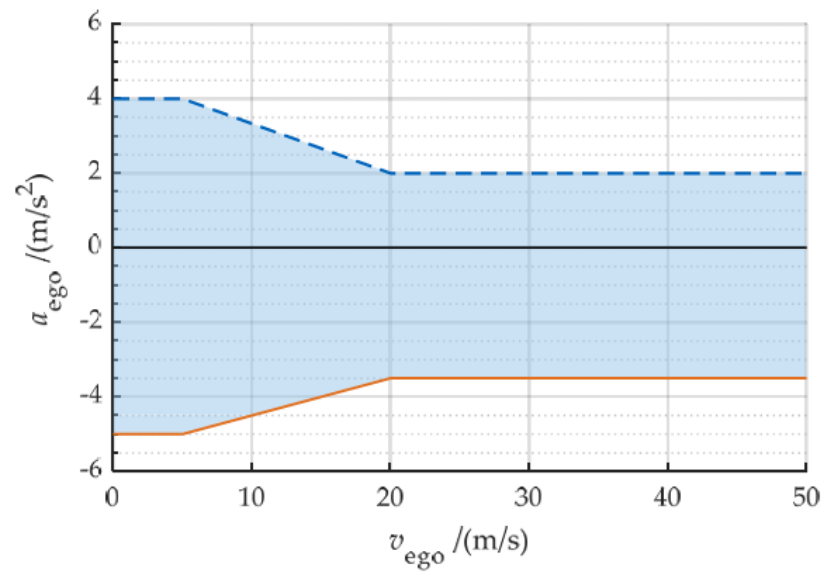


Figure 2: Upper and lower limits of acceleration according to ISO 15622:2018 (Duan, Schockenhoff and Koch, 2022)

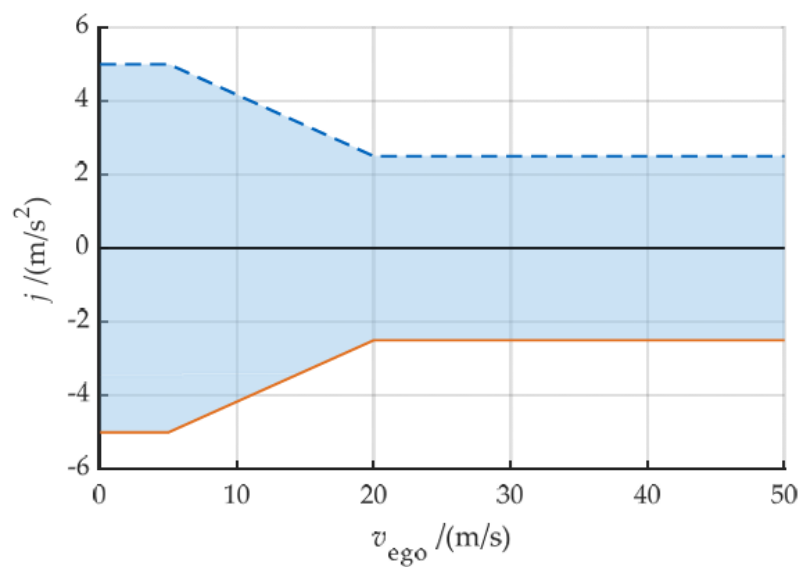


Figure 3: Upper and lower limits of jerk according to ISO 15622:2018 (Duan, Schockenhoff and Koch, 2022)

3. PID Tuning

3.1 Method

The cruise control system was implemented using a digital PID controller, structured as a function (Appendix 2) that receives driver demand, actual velocity, and PID gains to compute the control signal. The trapezoidal rule was used for error integration.

Sensor, demand, and actuator signals were encoded and decoded in 16-bit binary, simulating digital communication, such as a Controller Area Network (CAN) bus. MATLAB's `bin2dec` function and linear scaling converted these values to physical units, introducing quantisation effects that reflect real-world embedded system constraints (Matlab, 2025) – see Figure 4.

```

70      % Decode raw bus signal (from x-bit binary to mph)
71      velocity = ((bin2dec(velocity_raw).*(M_velocityS-m_velocityS))/R)+m_velocityS;

78      %% Decode raw signal (from x-bit binary to mph)
79      demand = (bin2dec(demand_raw).*(M_demand-m_demand)/R)+m_demand;

91      %% Encode control signal into x-bit binary number
92      control_raw = dec2bin(((control-m_ControlSig)*R)/(M_ControlSig-m_ControlSig));

```

Figure 4: Binary to decimal conversion code

The cruise control script (Appendix 1) was extended to compute acceleration and jerk at each simulation step, capturing peak absolute values for comparison with ISO 15622:2018 thresholds (Figure 5). Four key plots were generated: velocity response, control effort, acceleration, and jerk, enabling assessment of dynamic behaviour and identification of critical regions for performance tuning.

```

100      % Convert velocity from mph to m/s
101      velocity_mps = velocity_real * 0.44704;
102
103      % Compute acceleration
104      if j > 1
105          prev_velocity_mps = y_vec(j-1) * 0.44704;
106          a_vec(j) = (velocity_mps - prev_velocity_mps) / sim_step;
107      end
108
109      peak_acc = max(abs(a_vec)); % Stores max absolute acceleration (m/s^2)
110
111      % Compute jerk
112      if j > 2
113          j_vec(j) = (a_vec(j) - a_vec(j-1)) / sim_step;
114      end
115
116      peak_jerk = max(abs(j_vec)); % Stores max absolute jerk (m/s^3)

```

Figure 5: Code for acceleration and jerk calculations

PID tuning was performed through trial-and-error, iteratively adjusting gains based on closed-loop response (Borase *et al.*, 2021). The proportional gain K_p was increased first to improve response speed while avoiding overshoot. Next, integral gain K_i was introduced to eliminate steady-state error. Finally, a derivative gain K_d was added to reduce overshoot and dampen oscillations.

Control output was constrained to $\pm 100\%$ to reflect throttle and brake authority limits (Figure 6).

```

33      %% Actuator Limits
34
35      m_ControlSig = -100; % Lower control limit (brake pedal fully engaged).
36      M_ControlSig = 100; % Upper control limit (accelerator pedal fully engaged).
37      if controlout < m_ControlSig
38          controlout = m_ControlSig; % Ensures control signal output is not less than the lower control limit.
39      end
40      if controlout > M_ControlSig
41          controlout = M_ControlSig; % Ensures control signal output is not greater than the upper control limit.

```

Figure 6: Code for control output limits

3.2 Results and Analysis

Tuning began by increasing K_p in 0.1 increments, which successfully improved responsiveness and reduced error. A value of 2.1 yielded acceptable performance (Figure 7), but higher gains introduced overshoot and prolonged settling time. Adding a K_i value of 0.5 eliminated steady-state error, though settling remained long. Raising K_i to 1 further improved performance, as illustrated in Figure 8; however, values above 1 introduced oscillations.

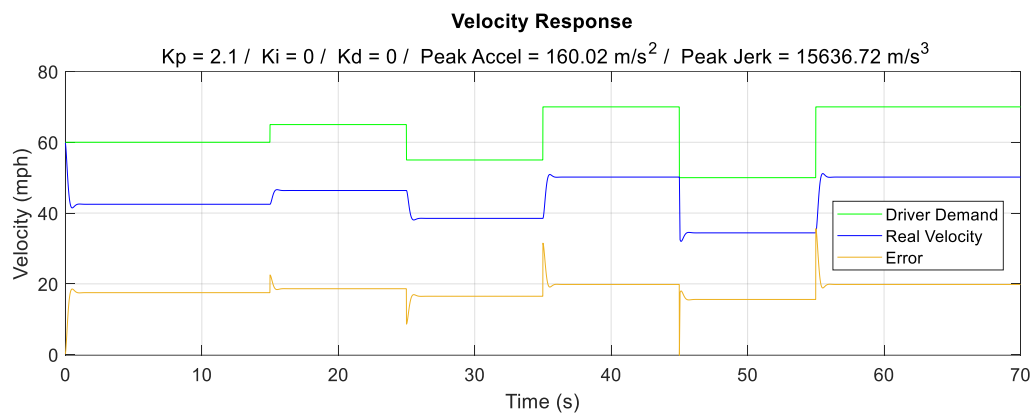


Figure 7: Velocity response plot ($K_p = 2.1$, $k_i = 0$, $K_d = 0$)

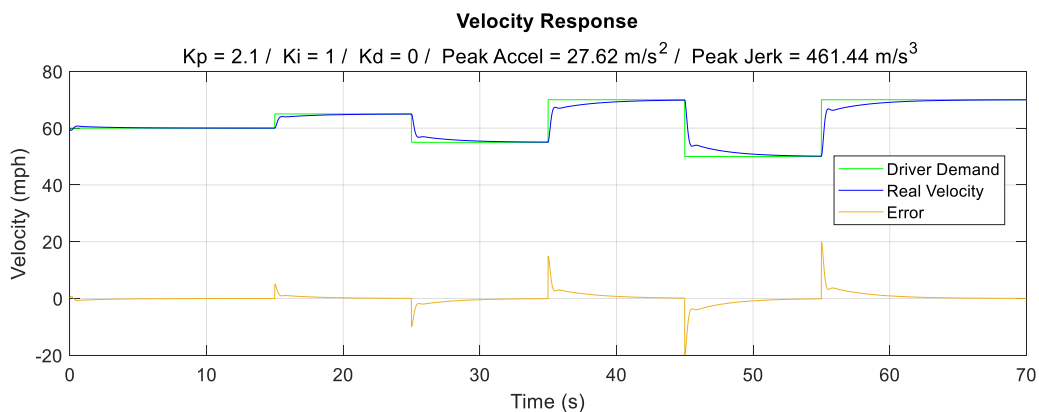


Figure 8: Velocity response plot ($K_p = 2.1$, $k_i = 1$, $K_d = 0$)

Despite improved tracking, acceleration and jerk remained excessive (Figure 9). A derivative gain of 0.1 exacerbated this, producing high-frequency oscillations and extreme rates of change. Reducing K_d to 0.05 brought improvements, cutting peak

acceleration from 27 m/s^2 to 18 m/s^2 , though values still exceeded ISO thresholds (Figure 10).

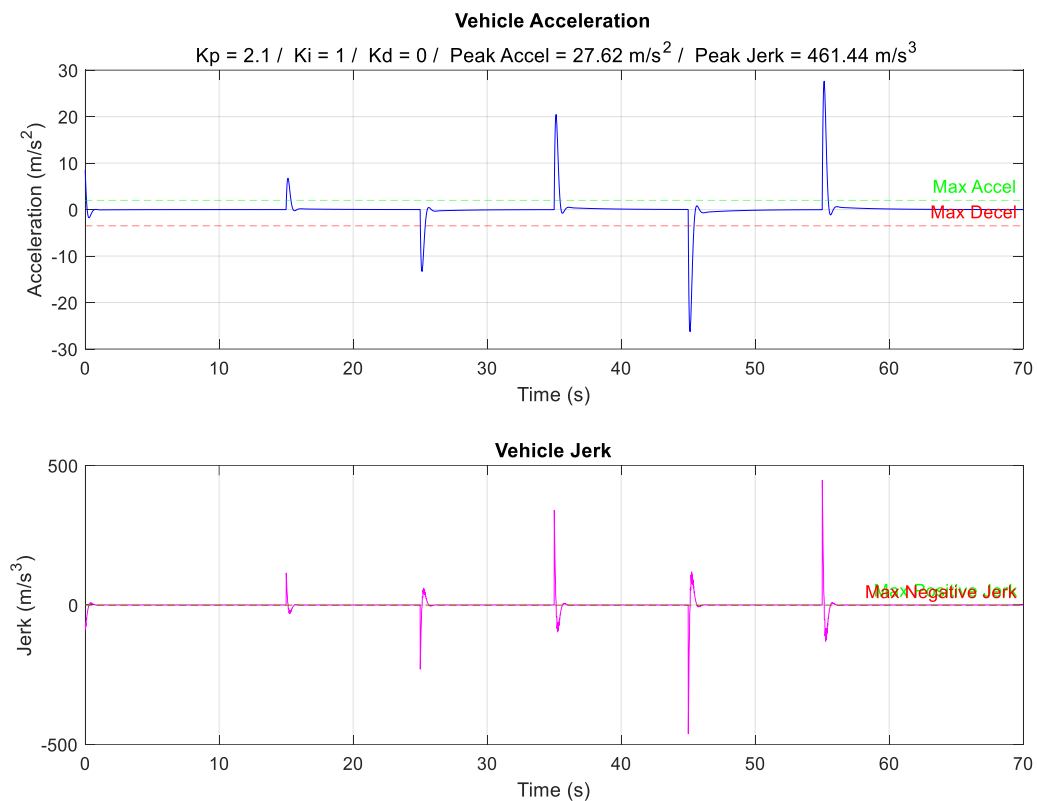


Figure 9: Vehicle acceleration and jerk plots ($K_p = 2.1$, $k_i = 1$, $K_d = 0$)

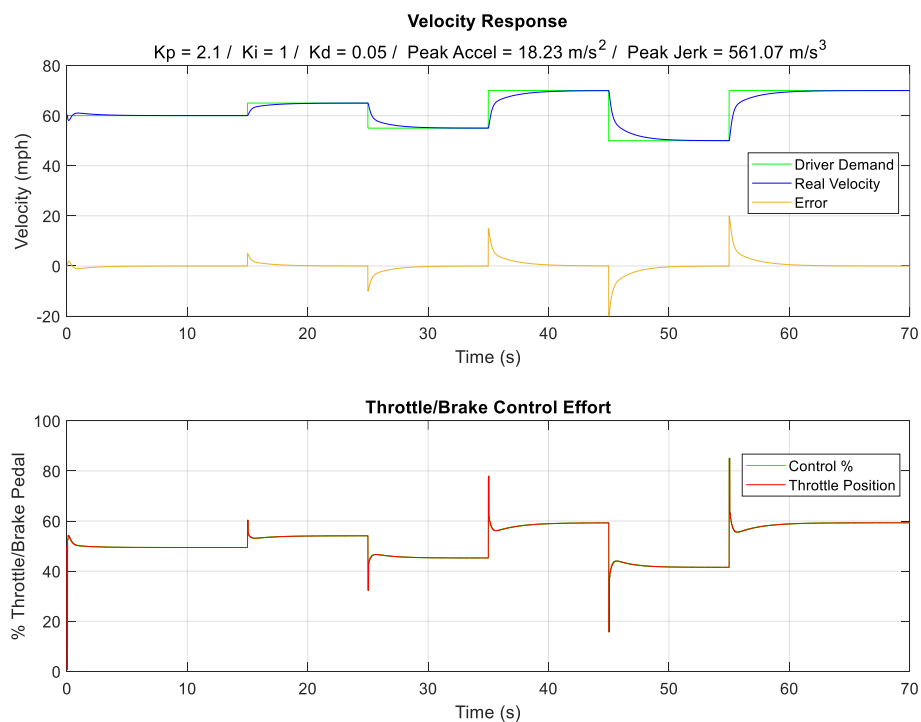


Figure 10: Vehicle response and control effort plots ($K_p = 2.1$, $k_i = 1$, $K_d = 0.05$)

To further reduce acceleration and jerk, a rate limiter was added to cap the control signal change per sample to 25%, preventing sharp throttle/brake spikes and improving comfort (Herbst and Madonski, 2024) – see Figures 11–12.

```

55 control_prev = control; % Initialize previous control value for rate limiter
56 max_rate = 25;          % Max rate of change of control signal (% per second)
57 max_step = max_rate * Ts; % Max change per control update (Ts is the sample time)

88 % Apply rate limiting to control signal
89 control = min(max(control, control_prev - max_step), control_prev + max_step);

```

Figure 11: Code for rate of change limiter on control output

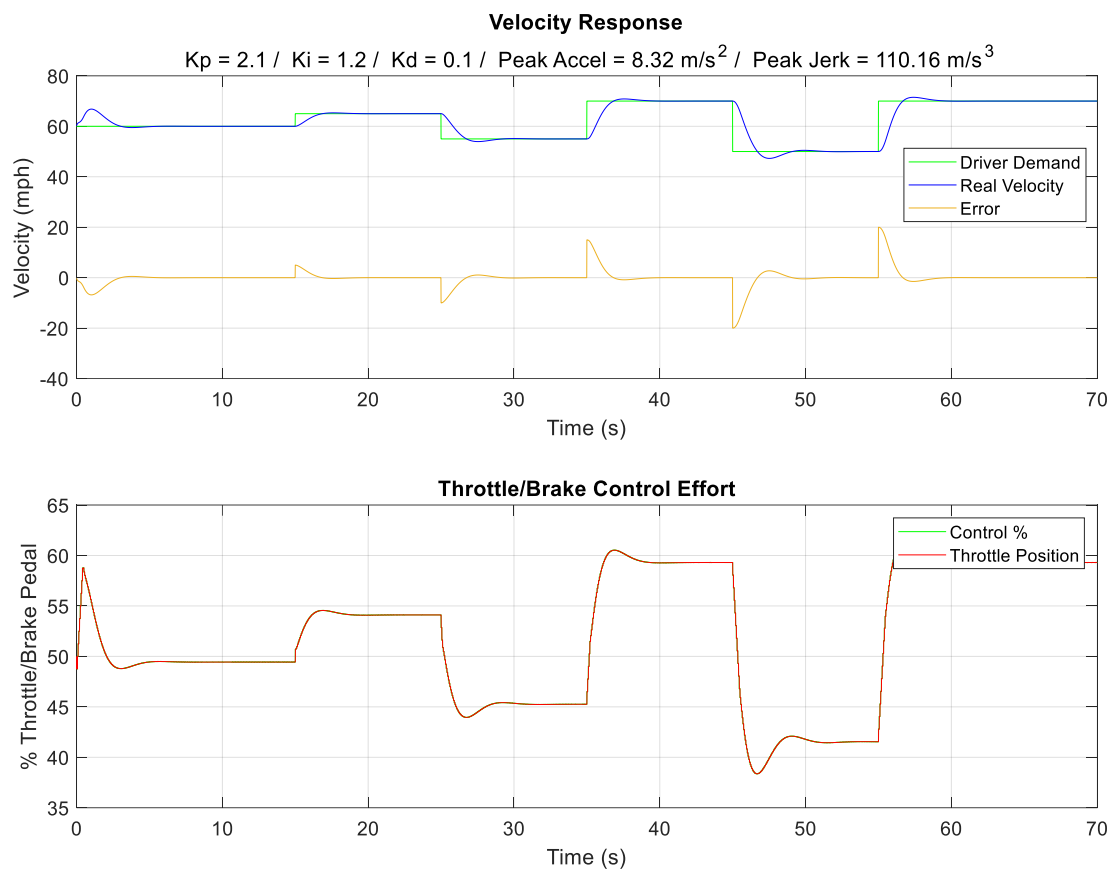


Figure 12: Vehicle response and control effort plots ($K_p = 2.1$, $k_i = 1.2$, $K_d = 0.1$)

However, even with rate limiting, the model's response remained overly aggressive. For instance, deceleration values reached 8.32 m/s^2 —well above realistic levels ($\sim 1.5 \text{ m/s}^2$) for typical passenger vehicles during engine braking (Maeda, Masahiko, 2023). This suggested the vehicle model was inherently too sensitive.

To reduce input jitter, a low-pass filter code was applied to the demand signal (Aridome *et al.*, 2018), using an alpha value of 0.965 (Figure 13). This smoothed set point changes and reduced peak acceleration to 3.62 m/s^2 (Figure 14).

```

81      %% Low pass filter to smoothen demand input
82      alpha = 0.965; % Smoothing factor (closer to 1 = slower response, smoother)
83      demand_filtered = alpha * demand_filtered + (1 - alpha) * demand;

```

Figure 13: Code for low-pass filter application to the demand signal

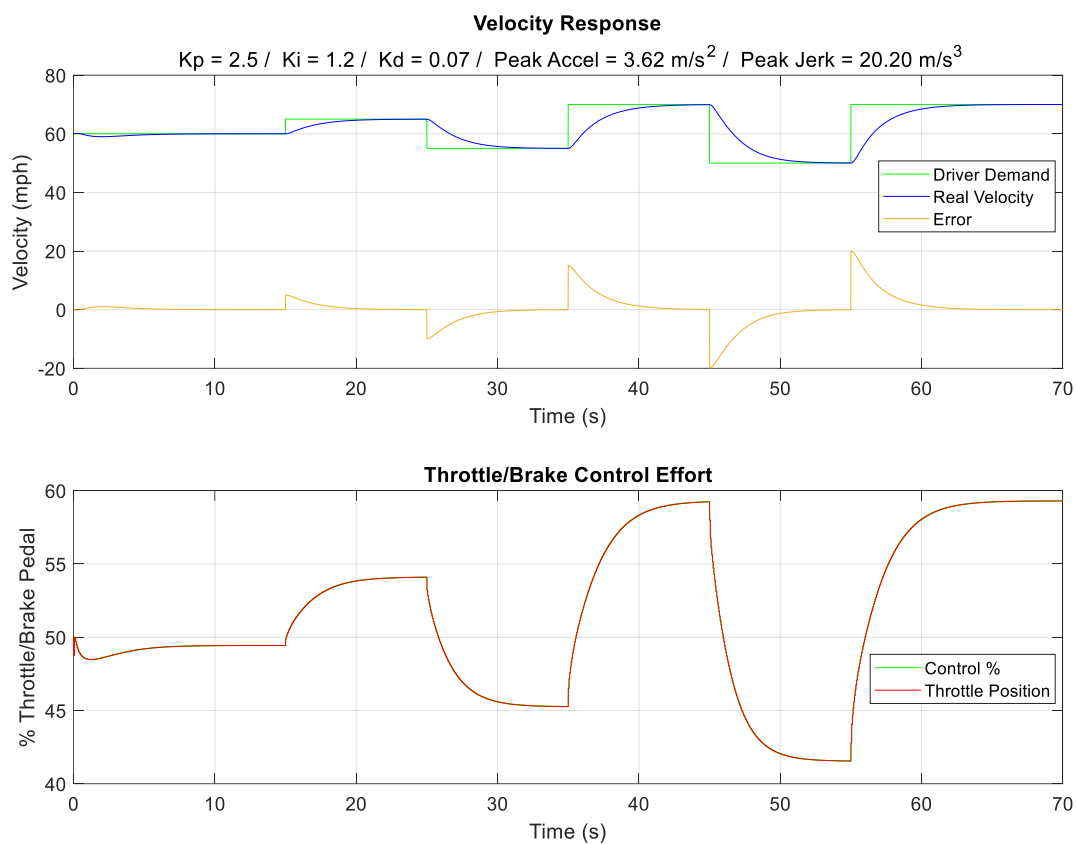


Figure 14: Vehicle response and control effort plots ($K_p = 2.5$, $k_i = 1.2$, $K_d = 0.07$)

4. Sampling Time Impact

4.1 Method

The cruise control script uses two time resolutions: the simulation time step and the sampling time step. The simulation step defines how often the system state (e.g., velocity) updates, while the sampling step controls how frequently the controller executes. It must be an integer multiple of the simulation step to maintain synchronisation.

To assess the impact of sampling time on system performance, the sampling interval was increased from 0.05 s to 0.5 s, doubling at each step. All other parameters were held constant between trials to isolate the effect of the sampling rate.

4.2 Results and Analysis

Increasing the sampling time from 0.05 s to 0.1 s resulted in increased settling time, consistent with the findings of Ismail and Hamza (2015) in their analysis of sampling effects on a heat exchanger system. The threshold sampling time was identified as 0.15 s; beyond this point, further increases led to noticeable degradation in performance, including extended settling times and increased actuator jitter, as shown in Figure 15.

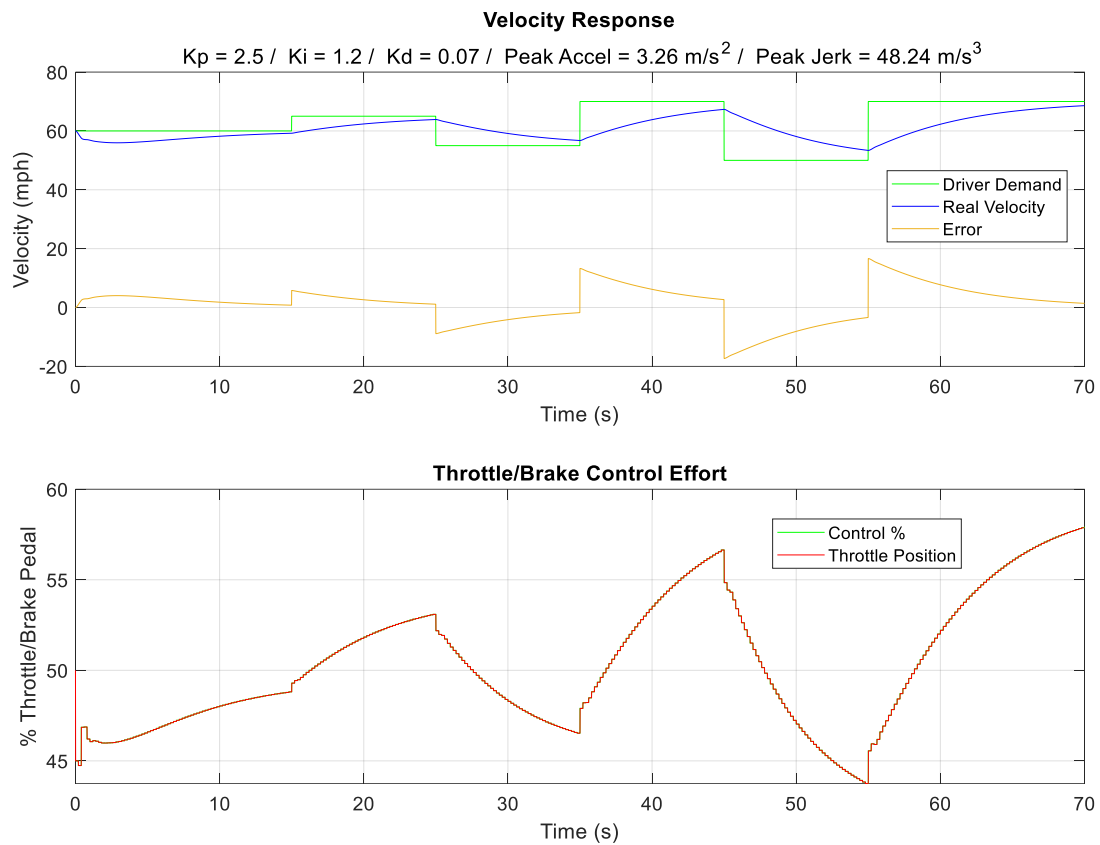


Figure 15: Vehicle response and control effort plots (sampling time = 0.2 s)

A sampling time of 0.5 s introduced substantial oscillations into the system, rendering it unstable (Figure 16). The frequent and rapid changes to the actuator could cause premature failure.

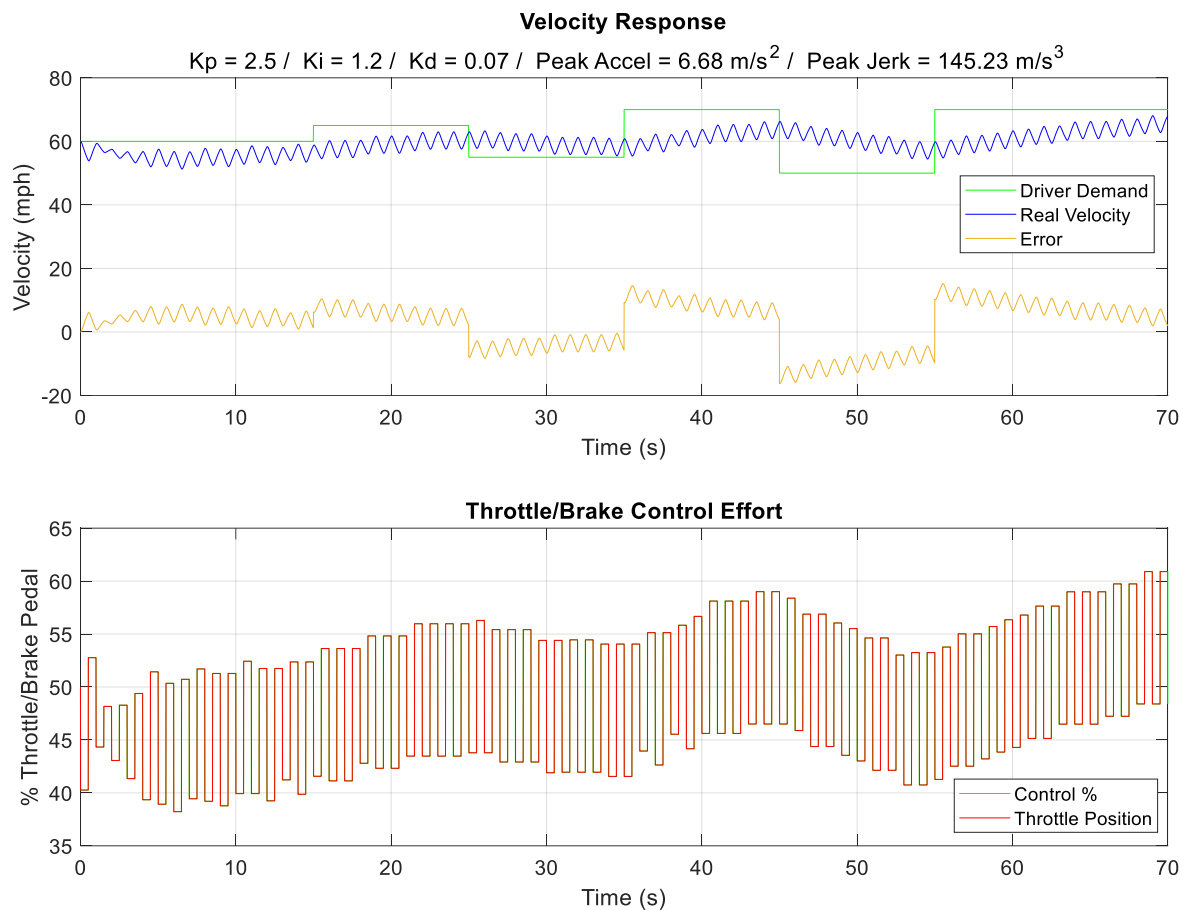


Figure 16: Vehicle response and control effort plots (sampling time = 0.5 s)

Numerous real-world constraints limit the sampling time, including computational power and network limitations (Tognetti and Calliero, 2017). As a result, system designers must carefully balance cost and performance—shorter sampling times can improve control accuracy but often require more expensive hardware and higher bandwidth.

5. Signal Delay Impact

5.1 Method

A function (Appendix 3) was implemented to simulate signal delay between the controller and actuator. This was achieved by substituting the current control signal with a delayed value from u_vec , offset by the desired delay in simulation steps.

Signal delay effects were analysed in 0.1 s increments from 0 to 1 s. PID gain adjustments were introduced to counteract performance degradation.

5.2 Results and Analysis

System performance remained stable up to a delay of 0.4 s. At 0.5 s, instability in vehicle velocity became apparent, caused by increased lag between the control signal and actuator response (Figure 19). Reducing the proportional gain from 2.5 to 2.0 improved the transient response.

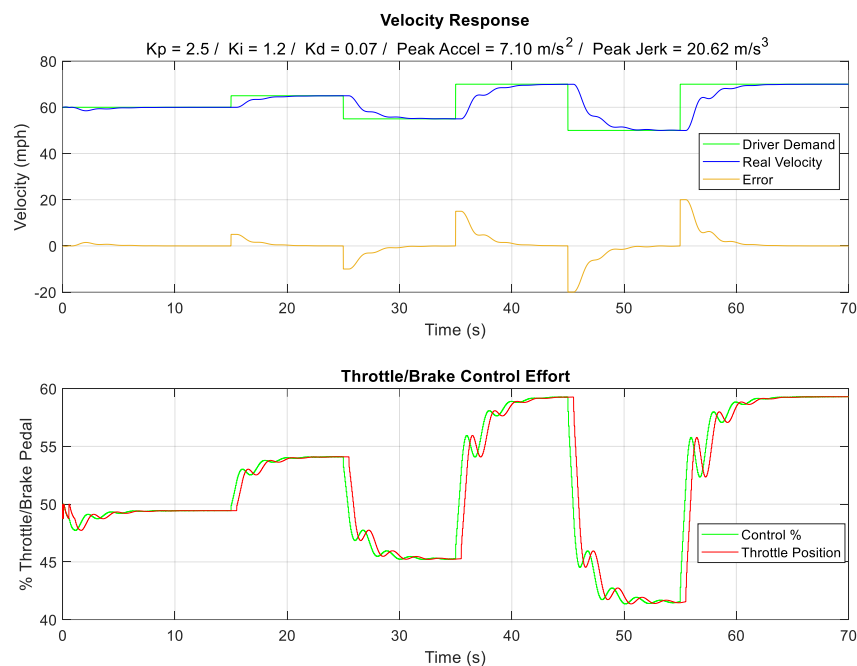


Figure 17: Vehicle response and control effort plots (signal delay = 0.5 s)

Delays exceeding 0.5 s resulted in unacceptable control behaviour. As shown in Figure 20, a 1-second delay resulted in substantial lag and excessive oscillations in velocity.

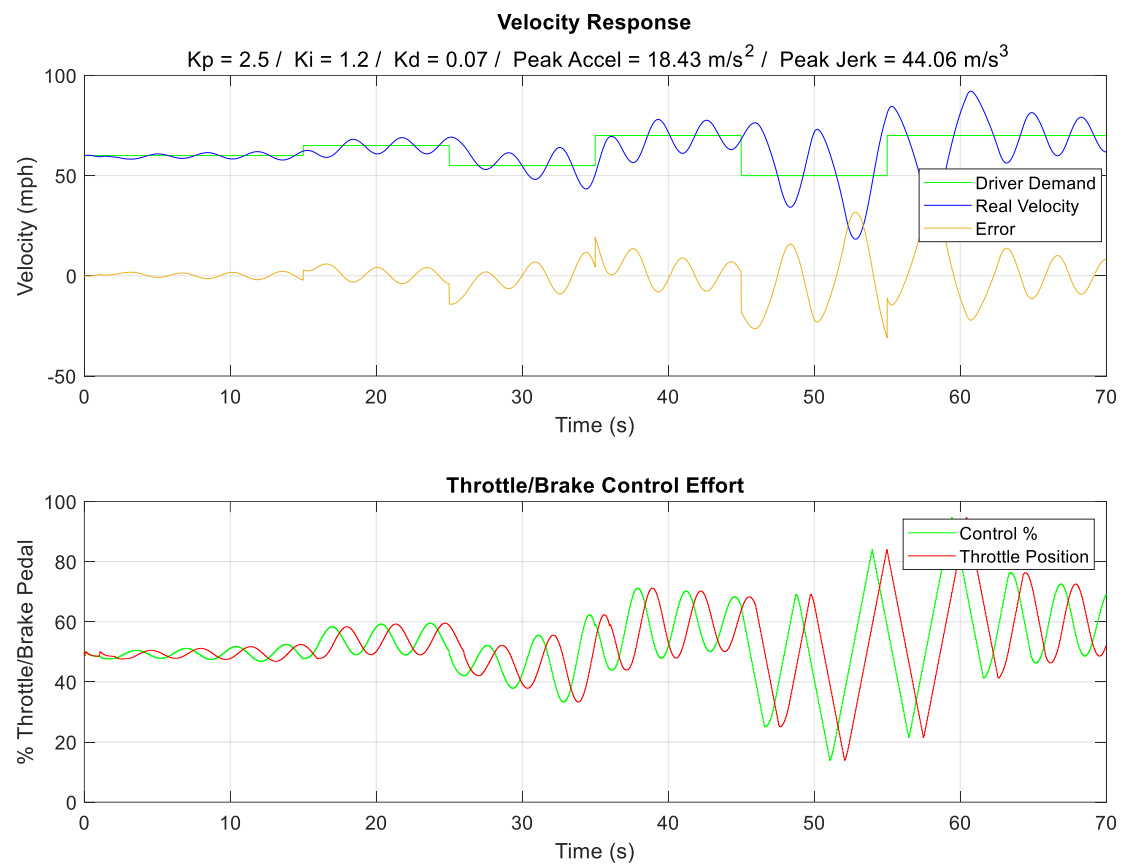


Figure 18: Vehicle response and control effort plots (signal delay = 1 s)

Further tuning—lowering the P, I, and D gains—improved control stability under a 1 s delay (Figure 21). However, this came at the cost of increased settling time and reduced responsiveness.

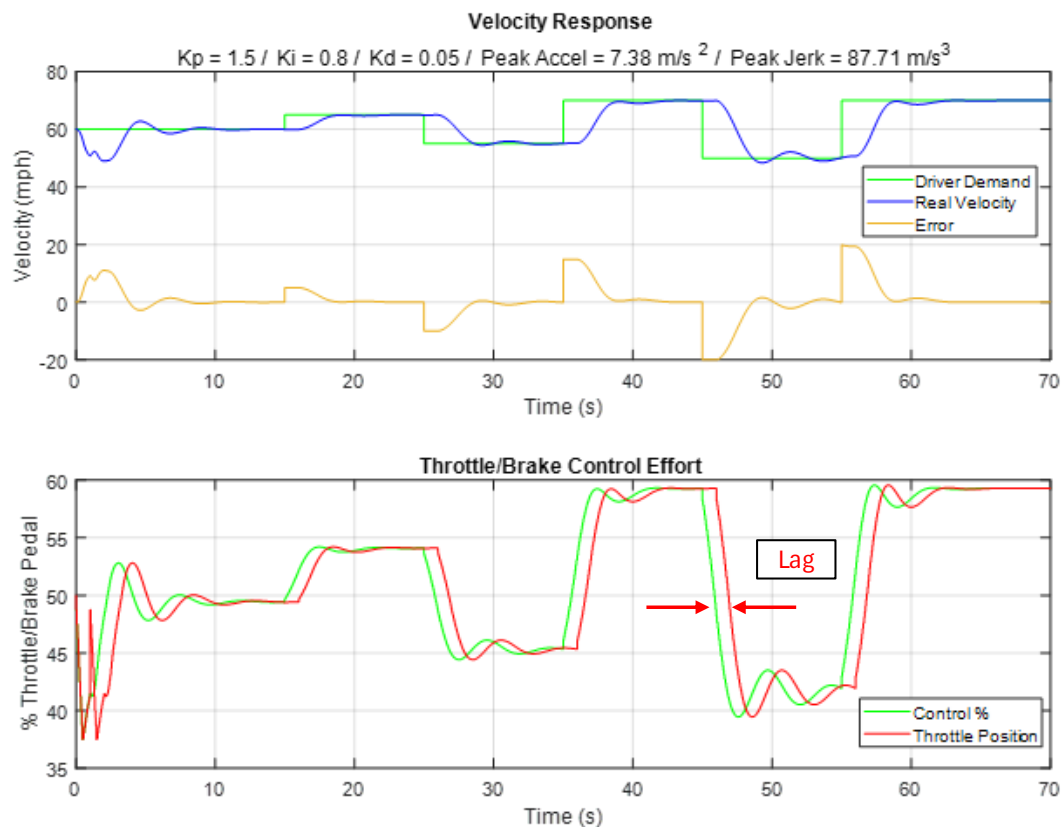


Figure 19: Vehicle response and control effort plots (signal delay = 1 s / reduced PID values)

Signal delays are present in real-world vehicle control systems due to computational processing limitations, actuator lag and communication delays. On CAN bus networks, message priority and bus load determine delay bounds. Even high-priority messages have finite worst-case bounds (Hielscher *et al.*, 2008).

6. Number of Bits Impact

6.1 Method

Quantisation—the conversion of continuous analogue signals into discrete digital values—introduces an inherent approximation, where signal values are rounded to the nearest fixed level based on bit resolution (Nise, 2019). The number of bits used determines how finely a sensor or actuator can capture or produce these values, directly influencing system performance and precision. To simulate the effects of bit resolutions, sensor and actuator signals were systematically varied. The experiment began with a 16-bit configuration (65,536 levels), then reduced by 2-bit increments down to 6-bit. The sensor, demand, and actuator bit resolutions were matched to prevent asymmetrical quantisation effects, which can cause unnecessary control activity or instability (Suh, Ro and Kang, 2010). All other parameters remained constant to isolate the impact of changing the bit resolution.

6.2 Results and Analysis

Reducing the bit resolution had no significant impact on performance until 10-bit, at which point quantisation effects became noticeable in the control effort plot (Figure 22). However, the velocity output remained largely unaffected.

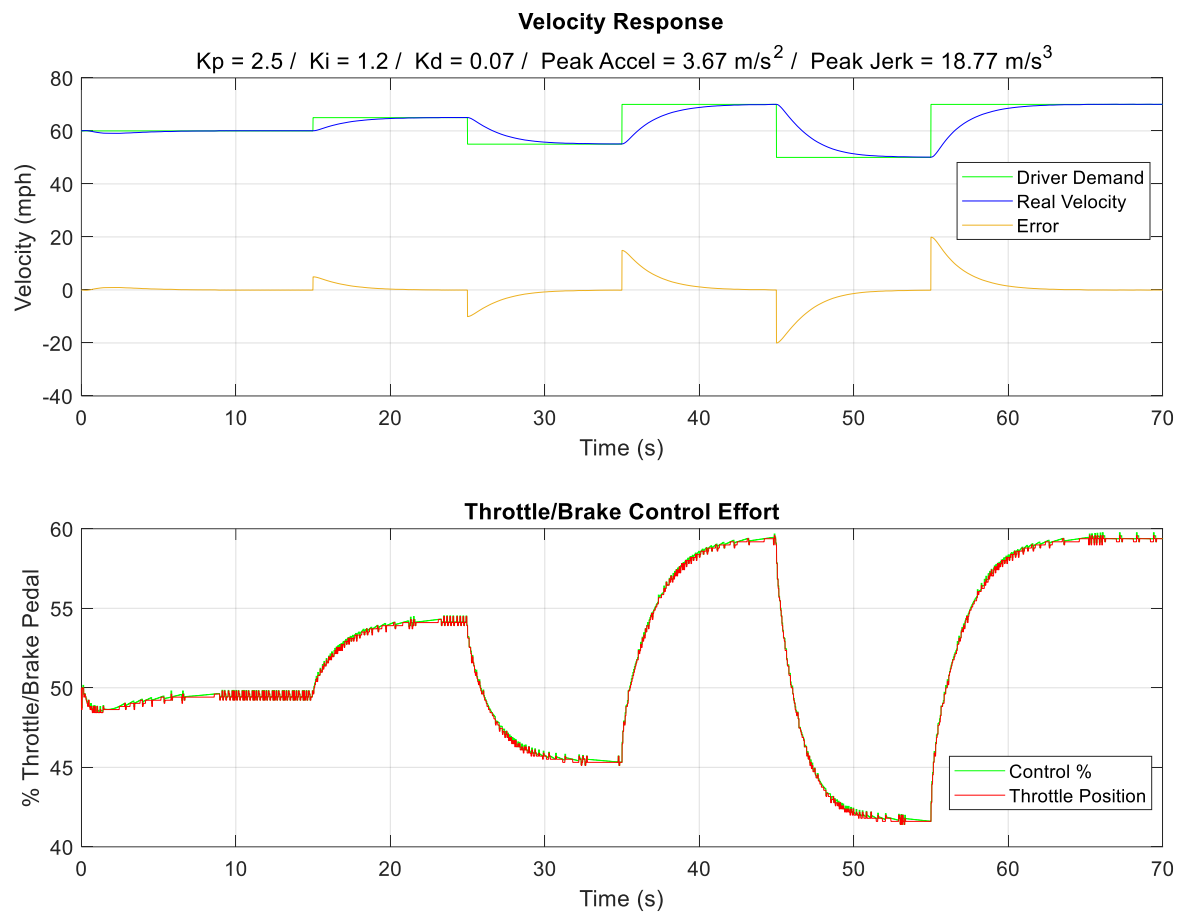


Figure 20: Vehicle response and control effort plots (bit resolution = 12-bit)

Degradation in system performance only became apparent at 6-bit resolution, indicated by increased tracking error in the velocity response, caused by quantisation in the actuator command signal (Figure 23).

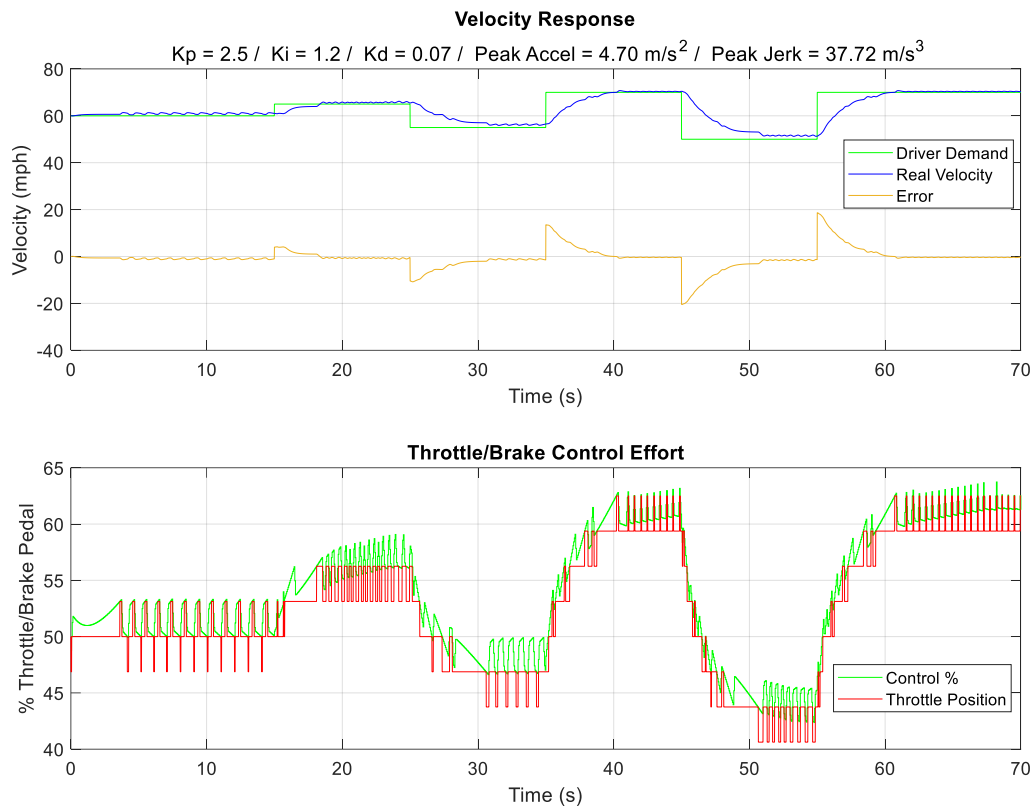


Figure 21: Vehicle response and control effort plots (bit resolution = 6-bit)

In this control system, 8-bit resolution was defined as the limit of functionality. While increasing resolution can improve performance, it also raises hardware cost, power consumption, and system complexity. Therefore, designers must weigh the performance gains against the associated resource demands and bandwidth limitations (Suh, Ro and Kang, 2010).

7. Conclusion

This report examined the design, implementation, and evaluation of a digital PID controller for vehicle cruise control, utilising MATLAB. The controller was tuned to track reference velocity inputs, with system performance benchmarked against acceleration and jerk limits from ISO 16522:2018. Enhancements such as a rate limiter and low-pass filter were introduced to improve ride comfort and stability.

The study also evaluated digital implementation constraints, including sampling time, bit resolution, and signal delay. Lower sampling intervals improved responsiveness but required more processing power and bandwidth. Signal delays introduced instability at higher values, although these effects were partly mitigated by adjusting the PID gains. Bit resolution impacted signal precision, but performance degradation was only notable at resolutions of 6 bits or lower.

Among these factors, increased sampling time and signal delay had the most significant influence on control behaviour, while quantisation effects remained manageable above 8-bit resolution. These findings highlight the trade-offs engineers must consider between performance, cost, and system limitations, particularly for embedded platforms with constrained communication capabilities such as CAN bus networks.

Overall, achieving robust control performance requires a careful balance between tuning strategies and the realities of digital implementation.

8. Bibliography

Aridome, H. *et al.* (2018) 'Design of an I-PD control system with low-pass filter and its application', *Electronics and Communications in Japan*, 101(10), pp. 24–30. Available at: <https://doi.org/10.1002/ecj.12112>.

Borase, R.P. *et al.* (2021) 'A review of PID control, tuning methods and applications', *International Journal of Dynamics and Control*. Springer Science and Business Media Deutschland GmbH, pp. 818–827. Available at: <https://doi.org/10.1007/s40435-020-00665-4>.

Bucz, Š. and Kozáková, A. (2012) *PID Controller Design for Specified Performance*. Available at: www.intechopen.com.

Duan, X., Schockenhoff, F. and Koch, A. (2022) 'Implementation of Driving Cycles Based on Driving Style Characteristics of Autonomous Vehicles', *World Electric Vehicle Journal*, 13(6). Available at: <https://doi.org/10.3390/wevj13060108>.

Herbst, G. and Madonski, R. (2024) *Control Engineering Active Disturbance Rejection Control From Principles to Practice*.

Hielscher, K.-S. *et al.* (2008) *Delay Bounds for CAN Communication in Automotive Applications*. *Delay Bounds for CAN Communication in Automotive Applications*. Available at: <https://www.researchgate.net/publication/221441060>.

International Organization for Standardization (2018) *ISO 15622:2018: Intelligent Transport Systems—Adaptive Cruise Control Systems—Performance Requirements and Test Procedures*. Geneva, Switzerland.

Isa, A.I. and Hamza, M.F. (2015) 'Effect of sampling time on PID controller design for a heat exchanger system', in *IEEE International Conference on Adaptive Science and Technology, ICAST*. IEEE Computer Society. Available at: <https://doi.org/10.1109/ICASTECH.2014.7068134>.

Maeda. Masahiko, *et al* (2023) *Special Feature Toyota's Full Lineup Strategy for Carbon Neutrality*. Aichi, Japan.

Matlab (2025) *Matlab bin2dec description*,
<https://uk.mathworks.com/help/matlab/ref/bin2dec.html>.

Nise, N.S. (2019) *Control systems engineering*. 8th edn. Hoboken, NJ: Wiley.

Suh, Y.S., Ro, Y.S. and Kang, H.J. (2010) 'Optimization of the sampling periods and the quantization bit lengths for networked estimation', *Sensors*, 10(7), pp. 6406–6420. Available at: <https://doi.org/10.3390/s100706406>.

Tognetti, E.S. and Calliero, T.R. (2017) 'Digital control and sampling period assignment of multiple plants in networked control systems', *IET Control Theory and Applications*, 11(17), pp. 3089–3096. Available at: <https://doi.org/10.1049/iet-cta.2017.0151>.

9. Appendix

Appendix 1 – CruiseControlScript.m

```

%% MAIN FILE FOR CRUISE CONTROLLER SYSTEM %%%%%%%%%%
% (1) Initialize
% (2) Set implementation parameters
% (3) Set simulation parameters
% (4) Simulation loop
% (5) Plot results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% (1) Initialize %%%%%%%%%%

clear functions % Ensures persistent variables inside functions are reset
clc % Clears command window
clear % Removes variables from workspace
close all % Closes all open figures
% PID gain values
P = 2.5;
I = 1.2;
D = 0.07;
set(groot,'defaultfigureposition',[400 250 1100 750]) % Sets default size and
position of figures
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% (2) Set implementation parameters %%%%%%%%%%

% Sampling time - must be an integer multiple of sim_step
Ts = 0.05;
% Velocity sensor range in mph [0,150]
m_velocityS = 0;
M_velocityS = 150;
% Number of bits used to encode velocity sensor signal
n_bits_sensor_signal = 16;
% Demand range in mph [30,110]
m_demand = 30;
M_demand = 110;
% Number of bits used to encode velocity demand signal
n_bits_demand_signal = 16;
% Number of bits used to encode throttle/brake actuator signal. Limits are set in
controlstript.
n_bits_actuator_signal = 16;
% Digital resolution, i.e., 16 bit = 65536 distinct values
R = 2.^n_bits_actuator_signal;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% (3) Set simulation parameters %%%%%%%%%%

sim_step = 0.01; % Simulation time step in seconds
sim_time = 70; % Simulation time length in seconds
T = 0:sim_step:sim_time; % Simulation steps vector
% Initialize vector where to store results for plots (7001 time plots)

```

```

% NaN is a placeholder for values that haven't been computed yet
[d_vec,u_vec ,y_vec,e_vec,Th_vec,a_vec,j_vec] = deal(nan(size(T)));
throttle_setting = 50; % Initialize actuator position (throttle/brake setting)
% Communication delay between controller and actuator in seconds - must be an
integer multiple of sim_step
actuator_signal_delay = 0;
[u, control] = deal(50); % Initialize control signal
demand_filtered = throttle_setting; % Initialize demand filtered signal
control_prev = control; % Initialize previous control value for rate limiter
max_rate = 25; % Max rate of change of control signal (% per second)
max_step = max_rate * Ts; % Max change per control update (Ts is the sample time)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% (4) Simulation loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

velocity_real = 60;
for j = 1:length(T) % For every simulation step

    loops_remaining = length(T)-j; % Display progression
    disp(loops_remaining)
    % Get vehicle raw velocity sensor measurement from the communication bus
    [velocity_raw, velocity_real,throttle_setting] =
getRawVelocity(n_bits_sensor_signal,velocity_real,throttle_setting, sim_step,u);

    % Decode raw bus signal (from x-bit binary to mph)
    velocity = ((bin2dec(velocity_raw).*(M_velocityS-m_velocityS))/R)+m_velocityS;

    if mod((j-1)*sim_step,Ts) == 0 % Every sampling time

        %% Get driver's velocity demand raw signal from the communication bus
        demand_raw = getRawVelocityDemand(n_bits_sensor_signal,sim_step,j);

        %% Decode raw signal (from x-bit binary to mph)
        demand = (bin2dec(demand_raw).*(M_demand-m_demand))/R)+m_demand;

        %% Low-pass filter to smoothen demand input
        alpha = 0.965; % Smoothing factor (closer to 1 = slower response,
smoother)
        demand_filtered = alpha * demand_filtered + (1 - alpha) * demand;

        %% Calculate control signal and error (range of control [-100,100])
        [control, error, m_ControlSig, M_ControlSig] =
ControlScript(demand_filtered,velocity,P,I,j,e_vec,Ts,sim_step,control,D);

        %% Apply rate limiting to control signal
        control = min(max(control, control_prev - max_step), control_prev +
max_step);

        %% Encode control signal into x-bit binary number
        control_raw = dec2bin(((control-m_ControlSig)*R)/(M_ControlSig-
m_ControlSig));

        %% Send control command to throttle/brake actuator via communication bus
        % Record the control signal instance as u
        u =
sendControlCommand(control_raw,n_bits_actuator_signal,actuator_signal_delay,j,
sim_step, u_vec);
        control_prev = control;
    end
end

```

```

% Convert velocity from mph to m/s
velocity_mps = velocity_real * 0.44704;

% Compute acceleration
if j > 1
    prev_velocity_mps = y_vec(j-1) * 0.44704;
    a_vec(j) = (velocity_mps - prev_velocity_mps) / sim_step;
end

peak_acc = max(abs(a_vec)); % Stores max absolute acceleration (m/s^2)

% Compute jerk
if j > 2
    j_vec(j) = (a_vec(j) - a_vec(j-1)) / sim_step;
end

peak_jerk = max(abs(j_vec)); % Stores max absolute jerk (m/s^3)

% store values for plotting
e_vec(j) = error;
d_vec(j) = demand;
u_vec(j) = control;
y_vec(j) = velocity_real;
Th_vec(j) = throttle_setting;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% (5) Plot results %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure

% Velocity response plot
subplot(2,1,1)
plot(T,d_vec,'g-');hold on
plot(T,y_vec,'b-');
plot(T,d_vec-y_vec); grid;
title('Velocity Response');
subtitle([' Kp = ', num2str(P), ' / Ki = ', num2str(I), ' / Kd = ', ...
    num2str(D), ' / Peak Accel = ', num2str(peak_acc, '%.2f'), ' m/s^2', ' / '
    'Peak Jerk = ', num2str(peak_jerk, '%.2f'), ' m/s^3']);
ylabel('Velocity (mph)'); xlabel('Time (s)');
legend('Driver Demand','Real Velocity','Error')

% Throttle/brake actuator effort plot
subplot(2,1,2)
stairs(T,u_vec,'g-');
hold on;
plot(T,Th_vec,'r-'); grid;
title('Throttle/Brake Control Effort');
ylabel('% Throttle/Brake Pedal'); xlabel('Time (s)');
legend('Control %','Throttle Position')

supersize(1.25) % Scaling factor for figures

figure

% Acceleration plot

```

```

subplot(2,1,1)
plot(T, a_vec, 'b'); grid on;
title('Vehicle Acceleration');
subtitle([' Kp = ', num2str(P), ' / Ki = ', num2str(I), ' / Kd = ', ...
    num2str(D), ' / Peak Accel = ', num2str(peak_acc, '%.2f'), ' m/s^2', ' /
Peak Jerk = ', num2str(peak_jerk, '%.2f'), ' m/s^3']);
xlabel('Time (s)');
ylabel('Acceleration (m/s^2)');
yline(2, '--g', 'Max Accel'); % ISO 15622:2018 comfort threshold
yline(-3.5, '--r', 'Max Decel'); % ISO 15622:2018 safe braking threshold

% Jerk plot
subplot(2,1,2)
plot(T, j_vec, 'm'); grid on;
title('Vehicle Jerk');
xlabel('Time (s)');
ylabel('Jerk (m/s^3)');
yline(2.5, '--g', 'Max Positive Jerk');
yline(-2.5, '--r', 'Max Negative Jerk'); % ISO 15622:2018 comfort threshold

supersize(1.25) % Scaling factor for figures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

Appendix 2 – ControlScript.m

```

function [controlout, error, m_ControlSig, M_ControlSig] =
ControlScript(demand_filtered, velocity, P, I, j, e_vec, Ts, sim_step, control, D)
%% PID function code for outputting control and error signals, and control limits.
Inputs are demand, velocity, P, I, j, e_vec, Ts, sim_step, control, D

error = demand_filtered-velocity; % Error = difference between setpoint (demand)
and process variable (velocity).

%% Proportional Part

P_part = error*P; % Error signal multiplied by proportional gain.

%% Integral Part

persistent Prev_I_part % Stores the previous integral term.
if isempty(Prev_I_part)
    Prev_I_part = 0; % Assigns zero to Prev_I_part variable when it is empty
(uninitialised).
end
if j>1 % Checks if the simulation index is the first iteration.
    prev = e_vec(j-(Ts/sim_step)); % Obtains previous error value from the error
vector (e_vec).
else
    prev = control; % Refers to initial control signal in CruiseControlScript.
    Prev_I_part = prev; % Sets first integral accumulator.
end

```

```

I_part = (((error+prev)./2).*(Ts)+Prev_I_part).*I; % Trapezoidal rule used to
integrate error over one sampling interval, accumulating it over time; value is
multiplied by integral gain.
Prev_I_part = ((error + prev) / 2) * Ts + Prev_I_part; % Updates the accumulated
integral value.

%% Derivative Part

D_part = ((error-prev/Ts))*D; % Calculates the rate of change of error and then
multiplies it by the derivative gain.

%% Control output

controlout = P_part+I_part+D_part; % Sums the P, I and D terms to produce the
control signal.

%% Actuator Limits

m_ControlSig = -100; % Lower control limit (brake pedal fully engaged).
M_ControlSig = 100; % Upper control limit (accelerator pedal fully engaged).
if controlout<m_ControlSig
    controlout=m_ControlSig; % Ensures control signal output is not less than the
lower control limit.
end
if controlout>M_ControlSig
    controlout=M_ControlSig; % Ensures control signal output is not greater than
the upper control limit.
end
end

```

Appendix 3 – ControlScript.m

```

function u = sendControlCommand(u_raw,x,actuator_signal_delay,j, sim_step, u_vec)
u = bin2dec(u_raw);
u = (u*(100+100))/(2^x) - 100;

if actuator_signal_delay > 0 && actuator_signal_delay/sim_step < j
    u = u_vec(j-actuator_signal_delay/sim_step);
end

```