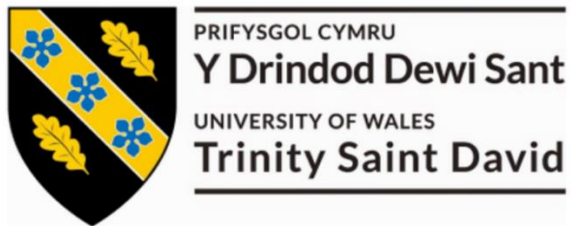


**University of Wales Trinity Saint David.
School of Engineering.**



Name: Matthew Dewhurst

Student Number: 2113040

Date: 10/07/24

Subject: Control Technology ACMA 6001

**Title: Modelling Industrial Control
Systems in MATLAB**

Degree: BEng Manufacturing Systems

Table of Contents

1.0 Task 1	3
1.1 Process Overview	3
1.2 Automated System Setup	3
1.3 System Software	4
1.4 Process Improvements	6
2.0 Task 2	8
2.1 Signal Encoding & Decoding Functions.....	8
2.2 Controller Gains	9
2.3 Bit Encoding & Decoding.....	10
2.4 Sampling Times	11
2.5 Communication Network Delay	12
2.6 Control System Implementation.....	13
3.0 References	14
4.0 Appendix	15

1.0 Task 1

1.1 Process Overview

To carry out the adhesive application to the windscreen of its vehicles, Aston Martin utilises an automated robotic cell. Within this cell, a robotic arm is programmed accordingly to carry out the adhesive application with precision and repeatability.

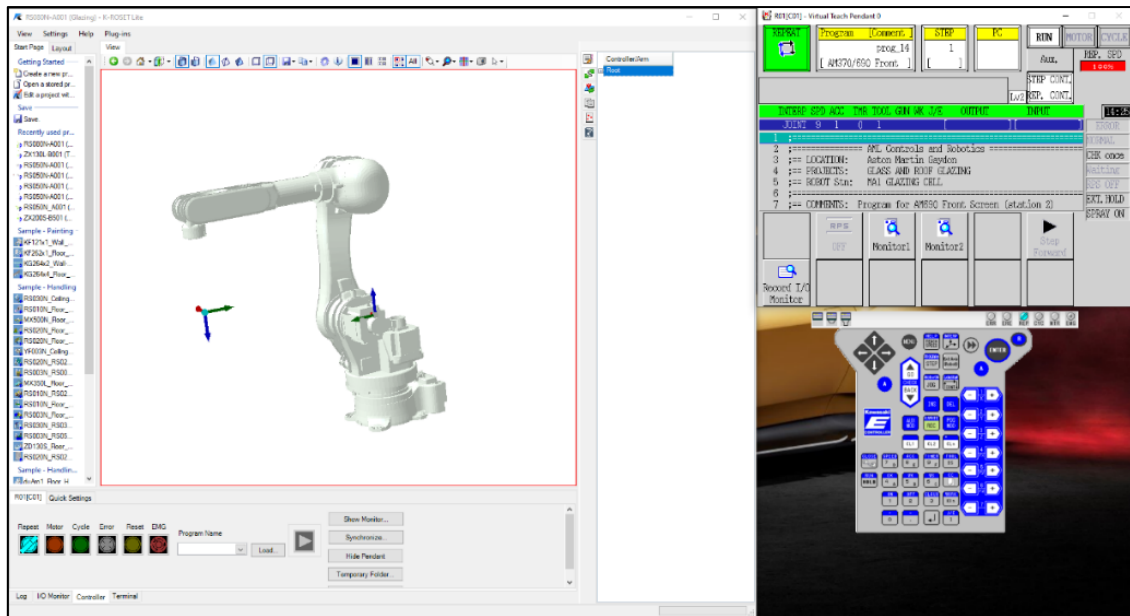
One element of a control system that must be considered is the sampling time, to ensure that the system is sampling data efficiently and effectively. One source describes the requirement of sampling time as, "The sampling time of a system refers to the rate at which a discrete system samples its inputs." (Matlab (2024)). Typically, the sampling time of a system should be '5-10' times the system frequency. As the production line that the adhesive cell is based on utilises a SCADA system, the sampling rate usually runs at a rate of '5' to '10' seconds. This rate is sufficient to control whether a particular behaviour or activity took place within the robot cell as well as across the entire production line. If there are delays in the sampling times, this could cause disturbances in the automated activities such as the adhesive application cell. Subsequently this can lead to further delays and stoppages in production, causing wastes in process time and resource which is critical to automotive manufacturers such as Aston Martin.

1.2 Automated System Setup

The automated adhesive application cell consists of a '6-axis industrial robot', part presence sensors, switches for vacuum cups and an 'E Series' Controller. The particular system used at Aston Martin is a Kawasaki product, the system interfaces with the robot to correctly dispense PU adhesive at a flow rate of '30ccm/s'. The system works by receiving an analogue speed value from the robot. This signal is a voltage of 0-15VDC which the adhesive system interprets, allowing the system to control the dosing speed of material and ensure a consistent adhesive bead is produced.

Before any adhesive is applied, the robot sends an 'glue_init' signal to the supervisory control acquisition, or otherwise known as 'SCA'. The signal sent by the robot triggers the pre-pressure of the material to 150bar, this improves the quality of the start of the bead by creating a uniformed bead instead of a tapered start. By pressurising the meter, this also allows the system to check for air bubbles in the meter as air will compress resulting in the encoder value changing as pressure increases. The robotic arm controls the operation of the dispensing of the glue with 'glueon' and 'glueoff' signals. These signals can be delayed or advanced to further control and improve the quality of the application. At the end of the application process the robot sends a 'glue_end' signal to the 'SCA', leading to the system entering a standby mode as well as recording the total adhesive applied.

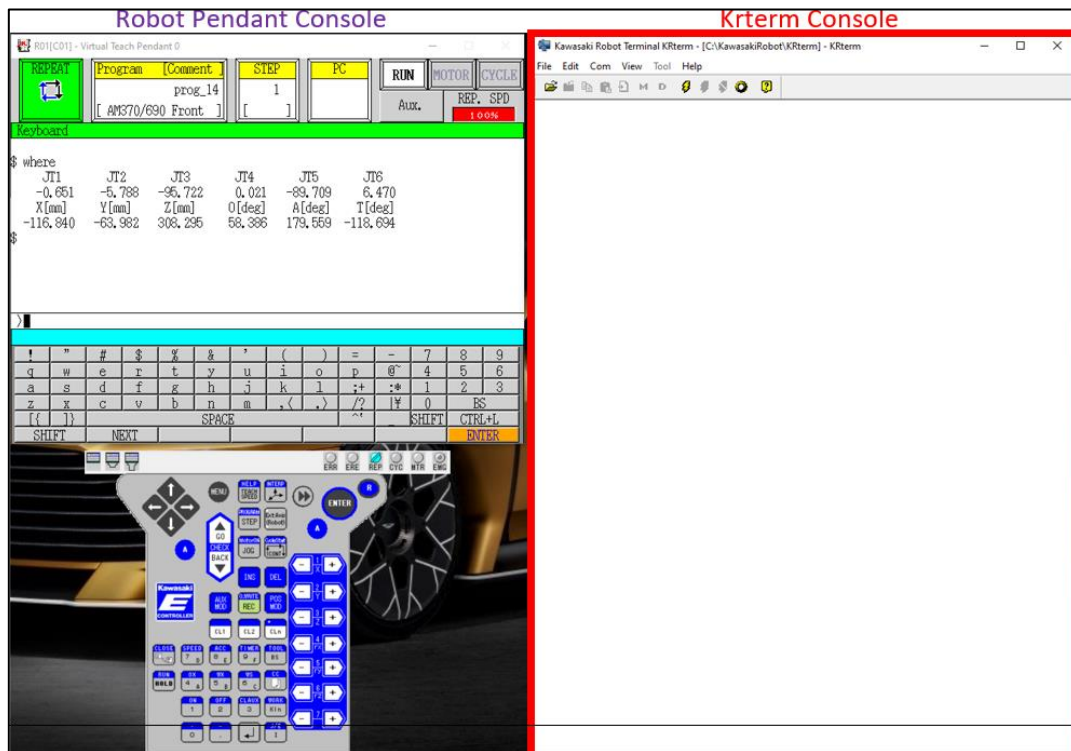
A benefit of the use of sensors in this system is that the amount of adhesive applied can be monitored to identify discrepancies in the application process. If the recorded volume matched the setpoint, the meter fills and waits for the next task. If there is a mismatch the system will stop and report an error to the HMI, this alerts the operator to a potential quality issue. As a result, the system can be considered reliable and effective in creating optimum quality to ensure that each windscreen is adequately sealed.



(Figure 1 – Configuration of 6-axis robotic arm for adhesive application – KrTerm, 2024.)

1.3 System Software

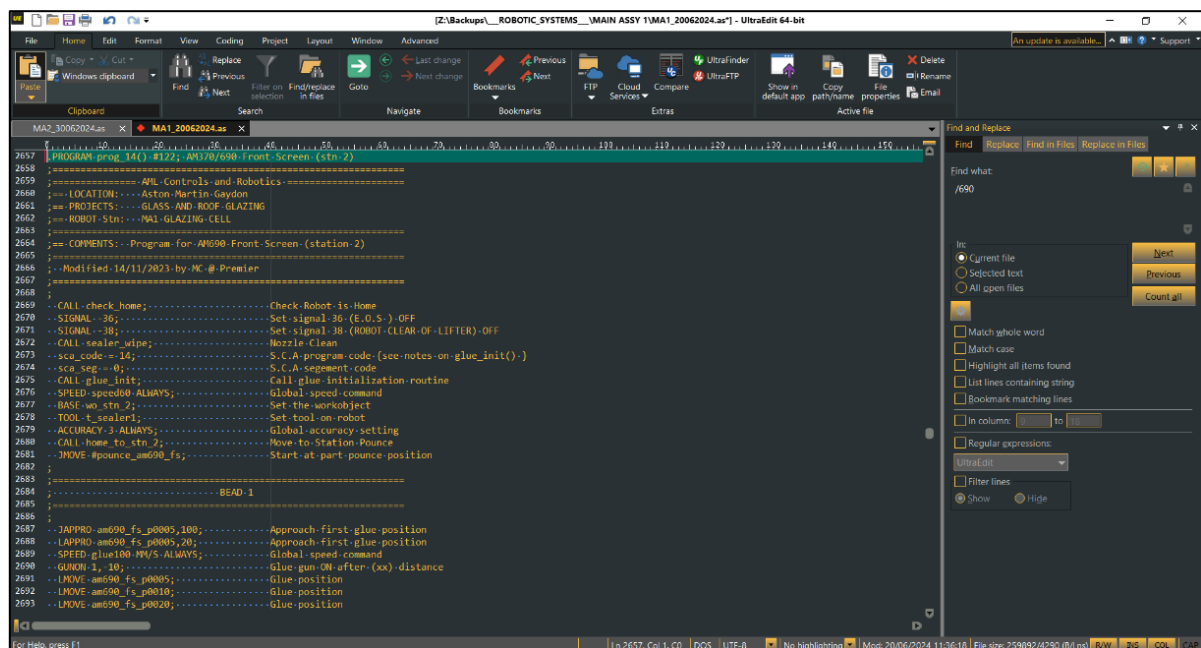
To control the system, a software called 'KrTerm' is utilised. This software allows the user to connect hardware such as a laptop to the controller and execute console commands, which are typically performed on the teach pendant at a much faster rate. There are many different console commands that can be used, most commonly different save commands are used to save a specific part of the robot data directly to the PC instead of a removable storage device. By removing the need for a removable storage device, the risk of data corruption is greatly reduced as well as improving overall cyber security.



(Figure 2 – Configuration of robot pendant console for adhesive application – KrTerm, 2024.)

To simulate the function of the system remotely before implementation, 'Kroset' software allows the user to configure and adjust parameters before the system is set up or adjusted. 'Kroset' is a simulation tool provided by Kawasaki, this tool has many uses and applications that allow programming to take place offline, away from the cell. This streamlines the process of programming allowing some of the work to be done offline, as well as test the system function before introduction. Any programming can then be loaded into the robot controller and modified to better suit the real-world application.

Alternatively, Aston Martin utilises another software package called 'UltraEdit' to manage/modify the robot programs outside of the systems associated software. UltraEdit is a text editing software that allows the user to format text, create new programs and manage existing files with a variety of tools. From this software, files can be created for different vehicle models or robotic cells and then stored in a central location for later use.

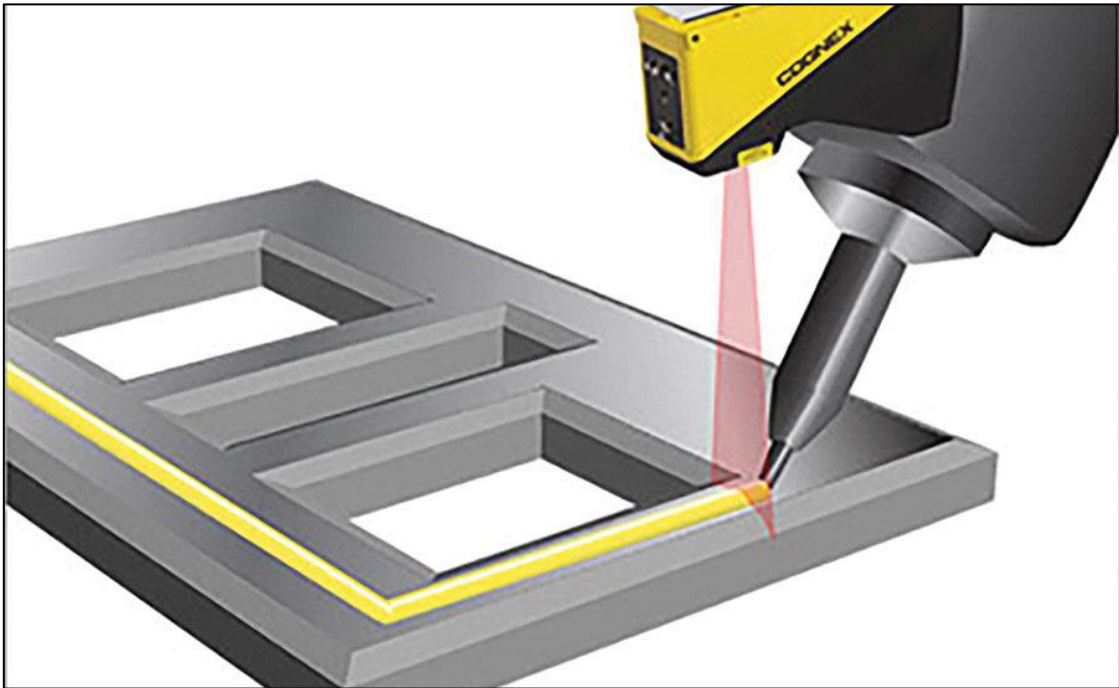


(Figure 3 – Alternative coding method of robotic arm adhesive application – UltraEdit, 2024.)

In addition, a similar tool that can be used to create program scripts is 'UltraCompare'. This text viewing window creates the opportunity to compare multiple text documents and filter down to identify differences only. This software also gives the user, the ability to trace and make amendments to the program using historical backups. Ultimately, these tools reduce the time required to set up new programs or amend current files and therefore leading to shorter down times.

To monitor the adhesive application process, a human machine interface or 'HMI' is available in the automated cell area for technicians and maintenance engineers to quickly assess the progress of the robotic arm. This HMI creates a quick status of the application and alerts the user of errors and the process cycle time, a benefit of this is that down time can be greatly reduced as engineers can identify an issue as soon as it occurs. Typically, the HMI will also present an error code or basic description so that the engineer can diagnose the issue at hand with ease and efficiency.

“The data provided by in-line measurement is another major advantage that often enables quality managers and engineers to rapidly improve the process of applying adhesives in automotive manufacturing.” (Robert Tremblay (2016)).



(Figure 5 – Vision system laser scanning adhesive bead application – Robert Tremblay, 2016.)

2.0 Task 2

2.1 Signal Encoding & Decoding Functions

Matlab is a tool that can be used for many different functions, in particular for the tuning of Proportional-Integral-Derivatives or 'PID's'. In this example, Matlab will be used to simulate a cruise control system in a vehicle and demonstrate how a PID controller is used in a system to maintain a steady driving speed. The settings of the PID controller can be tuned accordingly, allowing the user to analyse the vehicles performance with the use of graphs. Therefore, the optimum setting can be achieved to create a satisfactory performance for the user of the cruise control system.

One of the first elements of coding to set up a cruise control system is the setting of velocity sensor and demand parameters; this creates the minimum and maximum values of velocity that the system can function within.

```
% number of bits used to encode velocity sensor signal. velocity sensor
% range in mph [0,150]
min_velocityS = 0;
Max_velocityS = 150;
n_bits_sensor_signal = 16;

% number of bits used to encode velocity demand signal. demand
% range in mph [30,110]
min_demand = 30; Max_demand = 110;
n_bits_demand_signal = 16;

% number of bits used to encode throttle/brake actuator signal. actuator
% range in mph [-100,100] where [-100,0) is percentage of braking and
% (0,100] is percentage of throttle
n_bits_actuator_signal = 16;
R = 2.^n_bits_actuator_signal;
```

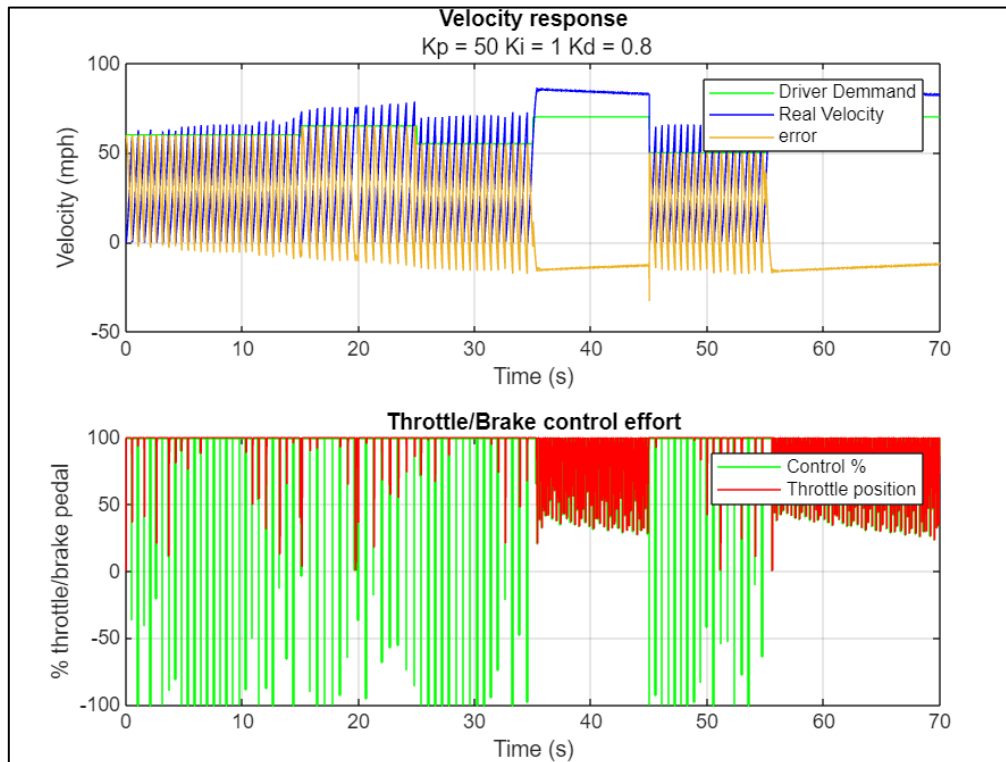
(Figure 6 – Encoding of velocity sensor, demand actuator signals – Matlab, 2024.)

From 'Figure 6', the values that are implemented for the velocity sensor are '0' and '150', these are the values that the cruise system can operate between, and the maximum is defined by the potential top speed of the vehicle. Similar to this, the velocity demand signal parameters are determined by the upper and lower limits of the cruise control system that the user can operate between. The user can utilise the cruise control between the values of '30' and '110'.

The value of bits defined in the system, shown in 'Figure 6' affects the range of values that you can represent, as well as how closely you can represent them. The actuator signal will be determined relative to the value of bits, to control the input of the user such as throttle and braking.

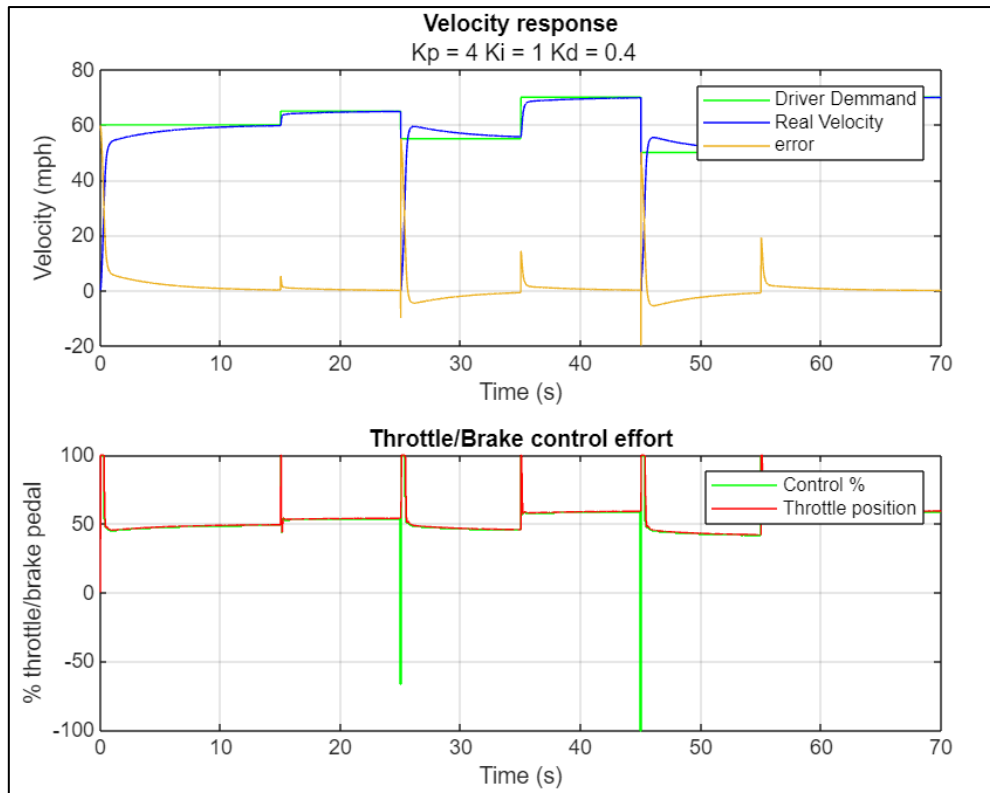
2.2 Controller Gains

Creating a feedback controller that ensures zero steady state error and sufficient performance requires careful selection of the controller type and fine tuning of its gains. In the example of the cruise control system, a PID controller is utilised and benefits from proportional, integral and derivative actions.



(Figure 7 – Graph to demonstrate poor configuration of PID controller – Matlab, 2024.)

The proportional gain in the system or otherwise known as 'P' addresses the present error and increases the control effort as error increases. In 'Figure 7' the proportional gain is set at '50' which increases the velocity response, allowing the system to respond faster. However, due to the high value of 'Kp' it has resulted in high amounts of oscillations and instability. The integral gain value or 'I', in the example is '1', this helps address the accumulation of past errors and eliminates steady state error. But the value of 'Ki' in 'Figure 7' demonstrates that there is steady state error in the results of the system. In addition, the derivative gain or 'D' is a value of '0.8', this gain is used to predict future error based on the current rate of change.



(Figure 8 – Graph to demonstrate improved configuration of PID controller – Matlab, 2024.)

To improve the response of the system the code used for the feedback controller is amended accordingly, to reduce the oscillations and reduce steady state error. Firstly, the proportional gain or ' K_p ' in 'Figure 8' has been changed to a value of '4', this is to reduce the oscillations and improve stability. Secondly, the integral gain or ' K_i ' of the system has been maintained at a value of '1', to eliminate steady state error. Lastly, the derivative gain or ' K_d ' is set at a value of '0.4', this has been changed in line with the proportional gain to improve stability and transient response. From these amendments, it is clear from 'Figure 8' that the feedback stability has been improved significantly. Also, the steady state error has been reduced and the graph now demonstrates an improved correlation between driver demand and real velocity.

2.3 Bit Encoding & Decoding

Experimenting with the number of bits for signal encoding and decoding involves finding a balance between precision of the signal representation and the efficiency of the system. Using more bits increases the precision of the signal representation and reduces error, as the control system can respond appropriately to feedback with precision. As well as this, an increased number of bits can improve signal quality, improving the system's ability to maintain high performance levels. However, increasing the number of bits can lead to processing delays as it puts more strain on the system, which results in a negative impact on the systems response time. By decreasing the number of bits, the response time of the control system can be improved due to faster processing times. However, it can also result in an increase in signal error due to a lower control accuracy. This signal error can accumulate and lead to less accurate or 'jerky' control response.

```

% number of bits used to encode velocity sensor signal. velocity sensor
% range in mph [0,150]
min_velocityS = 0;
Max_velocityS = 150;
n_bits_sensor_signal = 16;

% number of bits used to encode velocity demand signal. demand
% range in mph [30,110]
min_demand = 30; Max_demand = 110;
n_bits_demand_signal = 16;

% number of bits used to encode throttle/brake actuator signal. actuator
% range in mph [-100,100] where [-100,0) is percentage of braking and
% (0,100] is percentage of throttle
n_bits_actuator_signal = 16;
R = 2.^n_bits_actuator_signal;

```

(Figure 9 – Assessment of differing numbers of bits for sensor, demand and actuator signal – Matlab, 2024.)

When defining the number of bits for a control system, you must consider the specific requirements and constraints of the system. In the example of the cruise control system as demonstrated in 'Figure 9', '8' bits may introduce an extensive amount of signal error and therefore cause instability. By increasing the amount to '12' bits, stability and control accuracy are improved without excessive input load into the system. Whereas an increase to '16' bits can marginally improve accuracy but with increased processing time, which may not justify the marginal gain. Therefore, the minimum number of bits that can guarantee reasonable performance is '12' as this number provides adequate precision and efficiency.

2.4 Sampling Times

Experimenting with different sampling times in a control system is important for optimising performance, as it determines how frequently the system updates its state and applies control actions. The length of the sampling time can have effects on a range of system characteristics and the most suitable setting can vary depending on the application.

```

% initialize vector where to store results for plots
d_vec = []; u_vec = []; y_vec = []; e_vec = [] ; T_vec = [];

P = 50.0; % proportional gain
I = 1.00;
D = 0.80;

%% (2) set implementation parameters %%

% sampling time must be an integer multiple of sim_step
Ts = 0.05;

```

(Figure 10 – Experimenting with sampling times within cruise control code – Matlab, 2024.)

A shorter sampling time can improve the systems responsiveness by providing more frequent updates, leading to better response to rapid changes in the system's state. In addition, a short sampling time can also provide a more accurate control level. However, the drawbacks that the system may experience are an increase in computational load and data processing time. If the hardware or software cannot handle the demand of a short sampling time effectively, it can result in instability in the system output. On the other hand, a longer sampling time can reduce computational load, allowing simpler hardware and software to be used. However, this leads to a decrease in system response and creates the potential for rapid changes to be missed, as well as introducing a significant delay in output.

Within the example of the cruise control system as demonstrated in 'Figure 10', the longest sampling time that will still guarantee reasonable performance is '0.10'. This is due to the use of the system for a vehicle that could be potentially travelling at high speeds, creating a need for a fast response time and minimal delay so that the user can react effectively to ever changing surroundings on the road. In 'Figure 8' it has been highlighted that a sampling time of '0.05' is used, this short sampling time ensures that the cruise control system is safe to use and capture rapid variations accurately.

2.5 Communication Network Delay

Communication network delay affects the performance and stability of the control system through the introduction of latency and phase lag. The maximum tolerable delay depends on the system dynamics, which can be determined through simulation and testing.

```
sim_step = 0.01; % simulation time step in seconds
sim_time = 70; % simulation time length in seconds
T = 0:sim_step:sim_time; % define simulation steps vector
throttle_setting = 50; % initialize control action (throttle/brake setting)

% communication delay between controller and actuator in seconds.
% must be an integer multiple of sim_step
actuator_signal_delay = 0.0;
```

(Figure 11 – Actuator signal delay within cruise control code – Matlab, 2024.)

The current delay stated in the example of the cruise control system, as shown in 'Figure 11' is a value of '0.0'. However, the maximum delay that can be tolerated for satisfactory performance is a value of '0.2', as this reduces the phase margin to a critical level and if the value is increased further then there is a risk of significant overshoot or instability. Additionally, adjusting the controller gains can help accommodate larger delays as increasing the proportional gain can help maintain system responsiveness despite the delay. Another gain that can be adjusted to suit is the derivative gain, as this value can be increased to counteract the effects of the delay by predicting future errors and improve the system damping. On the other hand, there are limitations to the adjustment of gains to accommodate delays, due to the physical limitations of the actuators and sensors which may not be able to capacitate large adjustments.

2.6 Control System Implementation

In conclusion, a PID controller with carefully selected gains can ensure zero steady state error and effective performance. The key is to balance the velocity response and control effort of the system, by tuning the values of ' K_p ', ' K_i ' and ' K_d ' accordingly. In addition, during setup of the system, the user can test the gain parameters by methods such as trial and error or the 'Ziegler-Nichol's' method. One source describes the use of the 'Ziegler-Nichol's' method for closed loop systems, "A popular method for tuning P, PI, and PID controllers is the Ziegler–Nichol's method. This method starts by zeroing the integral and differential gains and then raising the proportional gain until the system is unstable. The value of K_P at the point of instability is called K_{MAX} ; the frequency of oscillation is f_0 . The method then backs off the proportional gain a predetermined amount and sets the integral and differential gains as a function of f_0 ." (Tibor Nagy (2019)). Methods such as this can help optimise the performance of a PID controller and reduce the time required to tune the gains adequately.

3.0 References

ctms.engin.umich.edu. (n.d.). Control Tutorials for MATLAB and Simulink - Extras: Steady-State Error. [online] Available at:
https://ctms.engin.umich.edu/CTMS/index.php?aux=Extras_Ess.

Kawasaki Heavy Industries (2022). E Series (Explosion-proof Specification) Installation and Connection Manual. [online] Available at:
https://kawasakirobotics.com/uploads/sites/2/2022/01/manual_controllers_e-controllers_e25_e27_en_01_2021.pdf [Accessed 22 Jul. 2024].

Tibor Nagy (2019) [www.sciencedirect.com](https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method). Ziegler Nichols Method - an overview | ScienceDirect Topics. [online] Available at:
<https://www.sciencedirect.com/topics/computer-science/ziegler-nichols-method#:~:text=The%20Ziegler%2DNichols%20Method%20is>.

[uk.mathworks.com](https://uk.mathworks.com/help/simulink/ug/what-is-sample-time.html). (n.d.). What Is Sample Time? - MATLAB & Simulink - MathWorks United Kingdom. [online] Available at: <https://uk.mathworks.com/help/simulink/ug/what-is-sample-time.html> [Accessed 19 Jul. 2024].

www.adhesivesmag.com. (n.d.). 3-D Vision Enables Challenging Adhesive Inspection Applications | 2016-05-02 | Adhesives Magazine | Adhesives & Sealants Industry. [online] Available at: <https://www.adhesivesmag.com/articles/94619--d-vision-enables-challenging-adhesive-inspection-applications> [Accessed 22 Jul. 2024].

4.0 Appendix

```
%% (1) Initialize %%

clc
clear all
close all % house keeping

% define global variables
global sim_step u j u_vec

% initialize vector where to store results for plots
d_vec = []; u_vec = []; y_vec = []; e_vec = [] ; T_vec = [];

P = 4.00; % proportional gain
I = 1.00;
D = 0.40;

%% (2) set implementation parameters %%

% sampling time must be an integer multiple of sim_step
Ts = 0.05;

% number of bits used to encode velocity sensor signal. velocity sensor
% range in mph [0,150]
min_velocityS = 0;
Max_velocityS = 150;
n_bits_sensor_signal = 16;

% number of bits used to encode velocity demand signal. demand
% range in mph [30,110]
min_demand = 30; Max_demand = 110;
n_bits_demand_signal = 16;

% number of bits used to encode throttle/brake actuator signal. actuator
% range in mph [-100,100] where [-100,0) is percentage of braking and
% (0,100] is percentage of throttle
n_bits_actuator_signal = 16;
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; % define simulation steps vector
throttle_setting = 50; % initialize control action (throttle/brake setting)

% communication delay between controller and actuator in seconds.
% must be an integer multiple of sim_step
actuator_signal_delay = 0.0;

%% Initialise Values - (Removed when in function)
min_ControlSig = -100;
Max_ControlSig = 100;
u = 0;
control = 0;
velocity_real = 0;

%% (4) simulation loop %%
```

```

for j = 1:length(T) % for every simulation step

    loops_remaining = length(T)-j % display progression

    % 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)*(Max_velocityS-
    min_velocityS))/R)+min_velocityS; % INCORRECT DECODING! INSERT YOUR DECODING
    FUNCTION HERE

    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)*(Max_demand-min_demand))/R)+min_demand;%
        INCORRECT DECODING! INSERT YOUR DECODING FUNCTION HERE

        % calculate control signal (range [-100,100])
        [control, error] =
        CalcControl_SC(demand,velocity,control,sim_step,Ts,j,P,I,D,e_vec); % INCORRECT
        CONTROL FUNCTION! INSERT CORRECT FUNCTION HERE

        % encode control signal into x-bit binary number
        control_raw = dec2bin(((control-min_ControlSig)/(Max_ControlSig-
        min_ControlSig))*R); % INCORRECT ENCODING! INSERT YOUR ENCODING FUNCTION HERE

        % 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);
        end

        % store values for plotting
        e_vec = [e_vec error]; % Error Value Vector
        d_vec = [d_vec demand]; % Demand Value Vector
        u_vec = [u_vec control]; % Control Percentage Value Vector
        y_vec = [y_vec velocity_real]; % Real Velocity Value Vector
        T_vec = [T_vec throttle_setting]; % Throttle Setting Value Vector
    end

    %% (5) plot results %%

    figure;

    % plot velocity response
    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)]);
    ylabel('Velocity (mph)'); xlabel('Time (s)');

```

```
legend('Driver Demmand','Real Velocity','error')

% plot throttle/brake actuator effort
subplot(2,1,2)
stairs(T,u_vec,'g-');
hold on;
plot(T,T_vec,'r-'); grid;
title('Throttle/Brake control effort');
ylabel('% throttle/brake pedal'); xlabel('Time (s)');
legend ('Control %','Throttle position')
```