DEPARTMENT OF MECHANICAL & AEROSPACE ENGINEERING

University of Strathclyde Engineering

Class Code/ Title:	ME528: Control Systems Design
Student Number:	201934594
Date:	6/12/2023

Title: ME528: Control Systems Design Project Report

1.0 Abstract

Modern cruise control systems are a staple of 21st century driving, allowing drivers allocate speed control to the car leading to more consistent speeds, better fuel economy and a less tiring journey.

The aim of this project was to develop a computer representation of a car cruise control system using Simulink and MATLAB. The project was divided into two parts; first using a simplified version of the overall equation of motion for the car, then progressing on to simulation of the non-linear dynamics, summing all velocity variant forces and using these to determine engine output control torque.

PI control was determined suitable for Part 1. Gains of $K_p = 1.81$, $K_i = 1.47$ were selected for this section as they had desirable controller characteristics. The gain combination did prove unstable at high angles of decent however, loosing stability past 12^o downhill.

A dynamic model was developed for Part 2, incorporating all forces at play in the system. To this, control torque was added and used as the main source of control. This led to snappy responses with potentially unrealistic behaviour. Further work should be done into ensuring a comfortable passenger experience when using control torque-based systems.

Contents

Nomenclature

Symbol	**Description**	**Units**
ζ	Damping Ratio	[-]
K_p	Proportional Gain	[-]
K_i	Integral Gain	[-]
K_d	Derivative Gain	[-]
θ	Road Angle	[rad]
g	Gravitational Acceleration	[m/s^2]
α_n	Force to torque ratio	[m^{-1}]
β	Engine characteristic	[-]
T_m	Max engine torque	[Nm]
Ω_m	Max engine speed	[rad s^{-1}]
m	Mass	[kg]
g	Gravitational acceleration	[m s^{-2}]
C_r	Rolling coefficient	[-]
C_d	Drag coefficient	[-]
ρ	Air density	[kg m^{-3}]
A	Wetted area	[m^2]
k_v	Control torque gain	[-]
k_d	Control torque gain	[-]
$\dot{x}_{sp}$	Velocity set point	[m/s]
x	Position	[m]
τ_c	Control torque	[Nm]

2.0 Introduction

2.1 Cruise Control

Cruise control is a functionality in modern cars which allows the driver to set a specified speed, and the car's on-board computers will ensure that the car stays at that speed until specified otherwise or deactivated [1].

Using a cruise control system can lead to an improved experience for the driver. Allowing them to rely on the car's computer to keep the car at a safe and legal speed means the driver can allocate more attention to the road and ensuring safe driving. The cruise control system can also use optimal acceleration to save fuel over a long journey, saving money over time for the driver.

2.2 PID Control

There are many combinations of gains that can be used in controllers. These are the combinations of P, I, D, PI, PD, ID, & PID.

- P represents the proportional gain of the controller. This is a single static gain which is multiplied with the input signal before summing.
- I represents the integral gain; the error signal is integrated over time after being multiplied by the gain.
- D represents the derivative gain, where the error after multiplication is differentiated with respect to time.

Examples of each controller combination with unity gain applied to a step input are shown below in Figure 1:

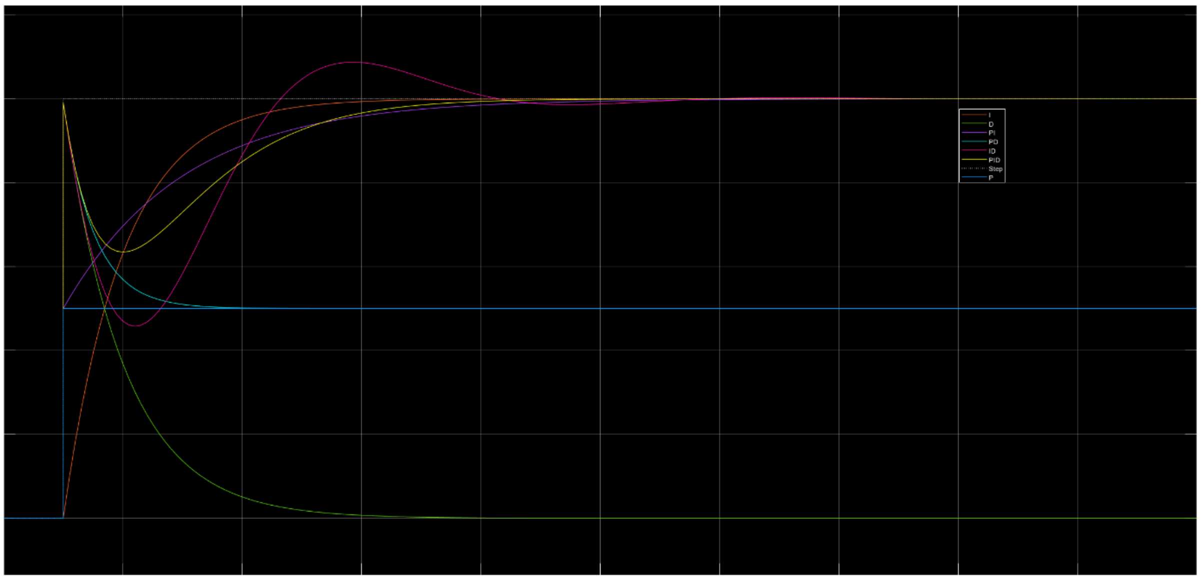

Figure 1 - Controller Combinations

It was determined that the combination of PI was suitable for the cruise control system. This allowed one gain to be dropped, simplifying the tuning of the controller in Part 1.

3.0 Part 1

3.1 Methodology

3.1.1 Simulink Model

For part 1, the following Simulink model shown below in Figure 2 was developed.

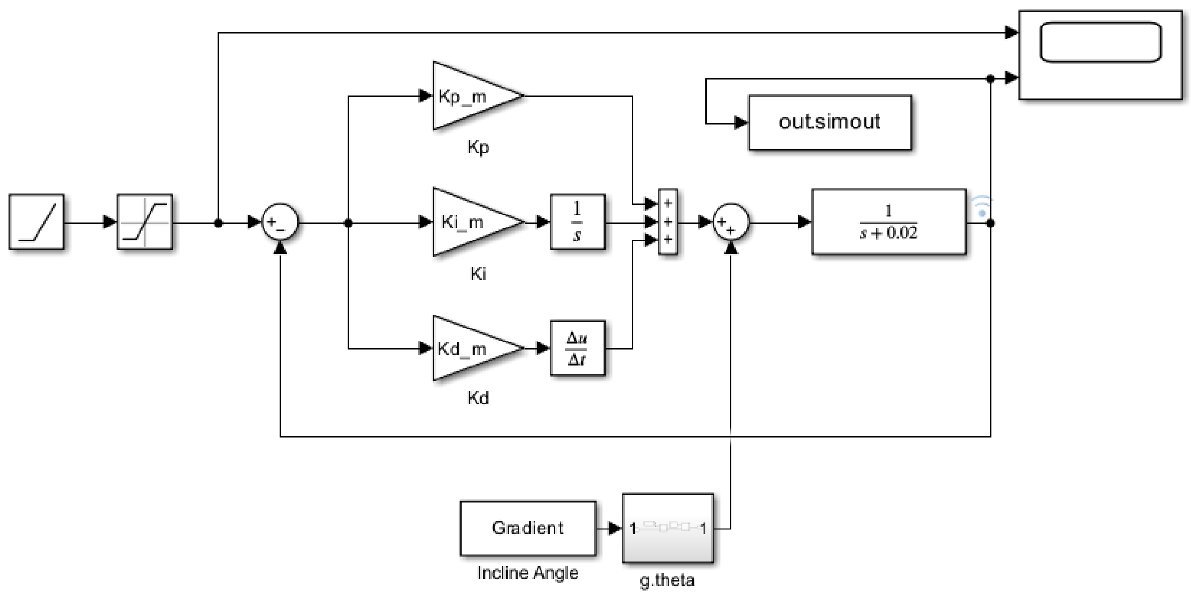

Figure 2 - Part 1 Simulink Model

Initially, the model used a step input to represent set velocity, however this was changed to a ramp input. The ramp input used a gradient that corresponded to a typical saloon car's 0 – 60 mph time, in this case 8.7s which corresponded to 6.9 mph/s. Converting units to m/s gave a 0 – 26.82 m/s time which corresponded to 3.08 m/s^2.

A saturation block was used to limit the input velocity of the ramp block. The lower limit of the saturation was set to 0 m/s, and the upper limit of saturation was set to 30 m/s. This corresponded in the simulation to accelerating from rest to 30 m/s in 9.74s. These settings were used for the following data logging steps.

3.1.2 Data Logging

Initially, PID values were manually iterated and recorded to an excel file. Each iteration involved a change to a single gain or disturbance variable, in this case these included K_p, K_i, K_d, and θ. Around 50 iterations were performed manually, with a comment of the results added to the excel sheet along with the accompanying gains.

This method was good for getting a general feel to how the variables affected the controller behaviour, and further showed that a PI controller was suitable for the problem. In the case of this report, however, it was not particularly effective for determining an effective combination of gains. The process was slow and tedious, and required a lot of manual data logging.

An output block was added to the Simulink file, which allowed the integration of a MATLAB script. By running the MATLAB script, Simulink would run and output the data as a structure. By importing the structure to MATLAB and logging the structure data to an array along, the controller behaviour was recorded for a given set of PID gains.

The use of a MATLAB script for running meant that variables used in Simulink were externally controlled. These variables were manually input for testing the script, and then initially again for iterating and testing controller behaviour. Doing this manually though was not necessary, as MATLAB allowed for the automatic iteration of gains and data logging.

Figure 2 was used for both stages of data logging,

By setting up 3 nested for loops, each of the three gains, K_p, K_i, and K_d were iterated between a specified minimum and maximum value at a specified step size for each. For each simulation the controller behaviour and damping ratio was recorded using MATLAB's inbuilt function – `damp(sys)`.

This initial script was iterated through values for K_p and K_i, leaving $K_d = 0$ for all iterations and produced a large array of gain combinations and performance statistics for each control system. To select suitable PI combinations from this large array, a second script was created which used the parameters of:

- Overshoot – Greater than 0%, Less than 4%.
- Damping Ratio – Greater than 0.75, Less than 1.

These conditions ensured desirable controller behaviour, and eliminated the majority of the undesirable PI combinations which left a selection small enough to manually analyse and choose from.

A further script, or more stringent parameters could have been applied to refine a more optimal controller in the end, however this was out of the scope of this project. Manual selection also allowed more than one candidate to be investigated, therefore allowing different behaviours to be studied.

3.1.3 Controller Testing

Three sets of gains were chosen for further study. For each set of gains, the Bode plot, Root Locus plot, and Routh-Hurwitz Array were generated. These each allowed the stability and response characteristics of each controller to be analysed. A transfer function was produced for the analysis of the zero-gradient case, shown below in 1:

$$T(s) = \frac{K_d s^2 + K_p s + K_i}{(1 + K_d)s^2 + (0.02 + K_p)s + K_i} \tag{1}$$

Once the most suitable of the three controllers was selected for further study, a more detailed study of effects was generated, this included the variance of gradient. To determine the effects of gradient on stability, the transfer had to be modified to accommodate this. The modified transfer function was shown below in 2:

$$T(s) = \frac{K_d s^2 + (K_p - g\theta)s + K_i}{(1 + K_d)s^2 + (0.02 + K_p - g\theta)s + K_i} \tag{2}$$

3.2 Results & Analysis

3.2.1 Overall Performance Assessment

The following: Figure 3, Figure 6, and Figure 9 were shown throughout this section. These were chosen as 3 PI combinations from the refined array. Each combination was chosen for a specific and unique reason and studied further.

3.2.1.1 $K_p = 0.26$, $K_i = 0.02$

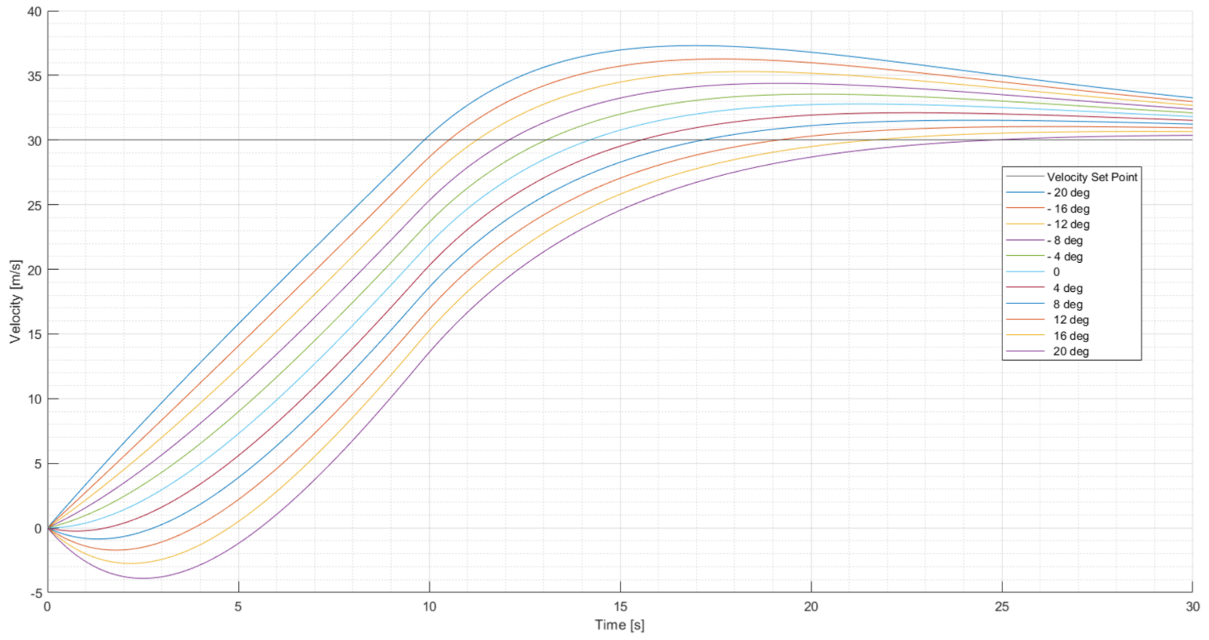

Figure 3 - K_p = 0.26 , K_i = 0.02: Velocity & Time

MATLAB determined this PI combination was numerically within the constraints set within the selection script, those being touched on above. In this case, the damping coefficient was determined to be $\zeta = 0.9899$, so it was barely within the range and was underdamped. This was confirmed by analysing the root locus plot:

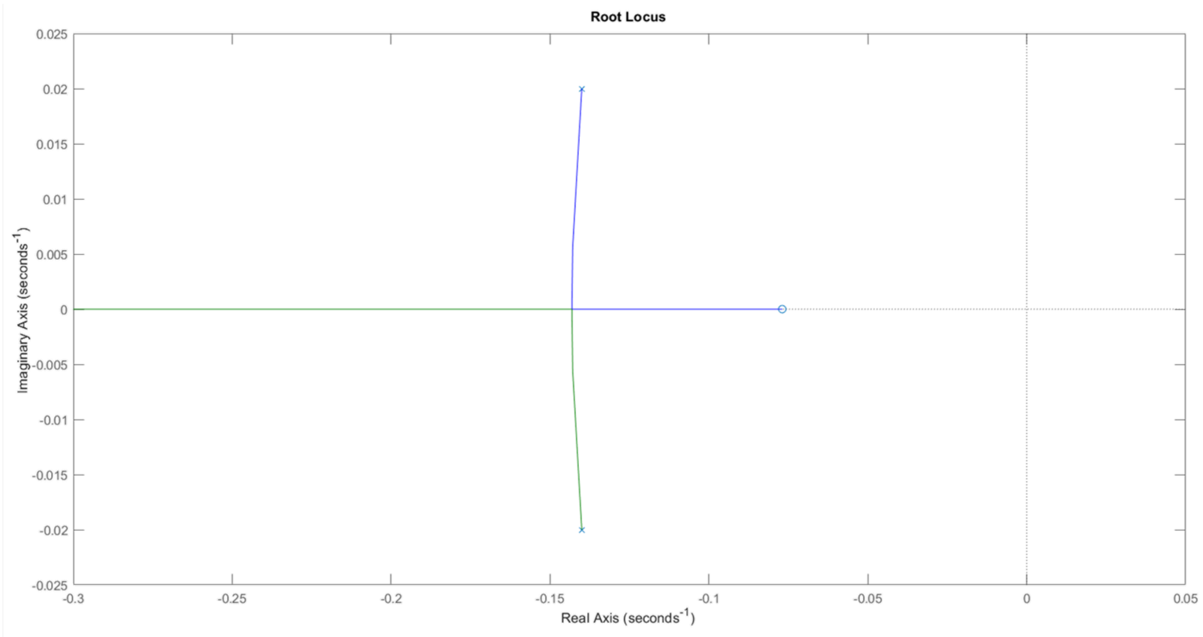

Figure 4 - Kp = 0.26 , Ki = 0.02: Root Locus

Shown in Figure 4, the poles on the imaginary axis represented that the system was underdamped. As well as this, all of the poles and zeros on the plot were to the negative side of the real axis, representing that the system will remain stable. The stability of the system was further verified using its Bode plot and Routh Hurwitz Array, shown below:

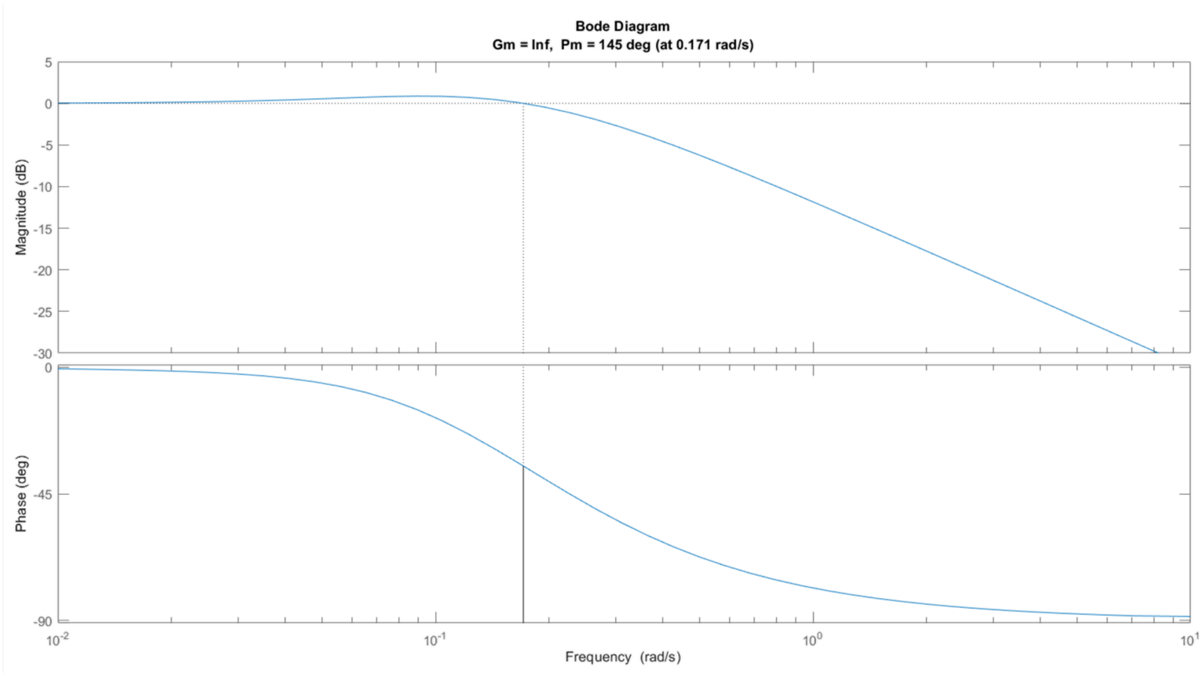

Figure 5 - Kp = 0.26 , Ki = 0.02: Bode Plot

Routh – Hurwitz Array:		
	1.00	0.02
	0.28	0
	0.02	0

There were a few overall flaws with this PI combination, notably the slow response time and lagging response to disturbances. Though MATLAB determined this combination to be within the margin of 4% overshoot, on visual inspection it could be see it was much greater than that. This was because the PI combination was tested only for the 0^o case and later iterated through multiple gradients for testing. An overshoot that large could be dangerous, especially when it stayed so far overshot for more than 30 seconds.

This controller gain combination was determined as unsuitable for use.

3.2.1.2 $K_p = 1.81$, $K_i = 1.47$

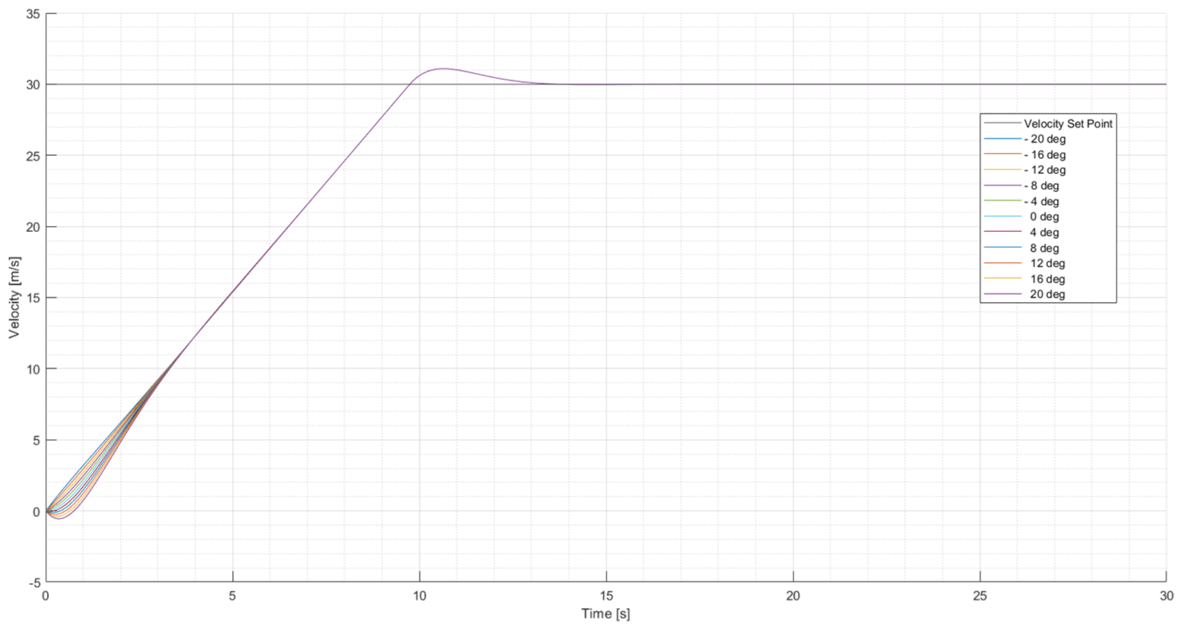

Figure 6 - $K_p = 1.81$, $K_i = 1.47$: Velocity & Time

Figure 6 above showed a slightly more responsive controller with a much more consistent behaviour between gradients. MATLAB correctly determined this controller to fall withing the 4% overshoot requirement and unlike the previous controller, regardless of the gradient, the controller always overshot by the same amount when it reached the set speed. The controller was also underdamped, but by a slightly larger degree this time. MATLAB determined $\zeta = 0.7547$. Underdamped behaviour was confirmed on inspection of the root locus plot, shown below in Figure 7:

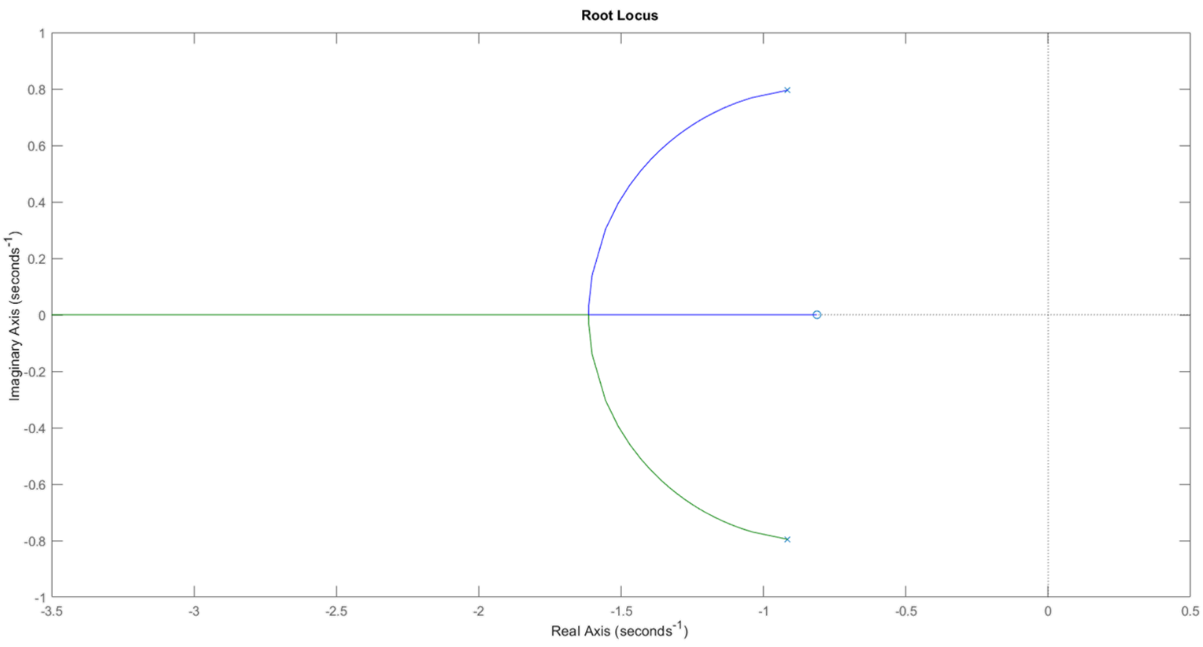

Figure 7- Kp = 1.81 , Ki = 1.47: Root Locus

Much alike Figure 4, Figure 7 had 2 roots on the imaginary axis which highlighted underdamped behaviour, confirming the value produced from MATLAB. As well as this, for the 0^{o} case, the plot was entirely on the negative side of the real axis and was therefore stable.

Stability was corroborated again using the Bode diagram and Routh Hurwitz array, shown below in Figure 8 and below:

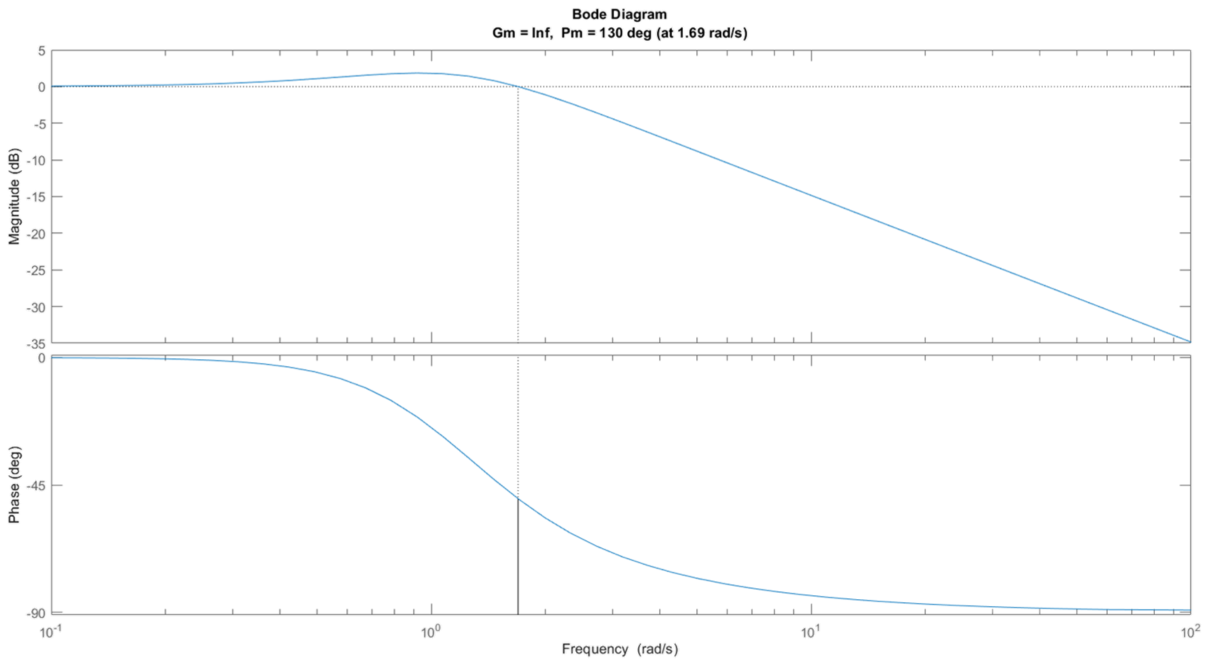

Figure 8- Kp = 1.81 , Ki = 1.47: Bode Plot

Routh – Hurwitz Array:		
	1.00	1.47
	1.83	0
	1.47	0

This controller was a great candidate for Part 1, with consistent behaviour regardless of gradient and stability and underdamped behaviour at a gradient of zero it met the selection conditions that were specified in the MATLAB selection script. As well as having desirable damping and consistency, it had a relatively snappy response time to the initial increase ramp, and when adjusting to the end of the ramp.

This controller PI combination was determined as potentially suitable for use.

3.2.1.3 $K_p = 2.38$, $K_i = 1.44$

Figure 9 below showed a controller with a similar response to the controller in Kp = 1.81 , Ki = 1.47, however the main difference here was the damping of the system response.

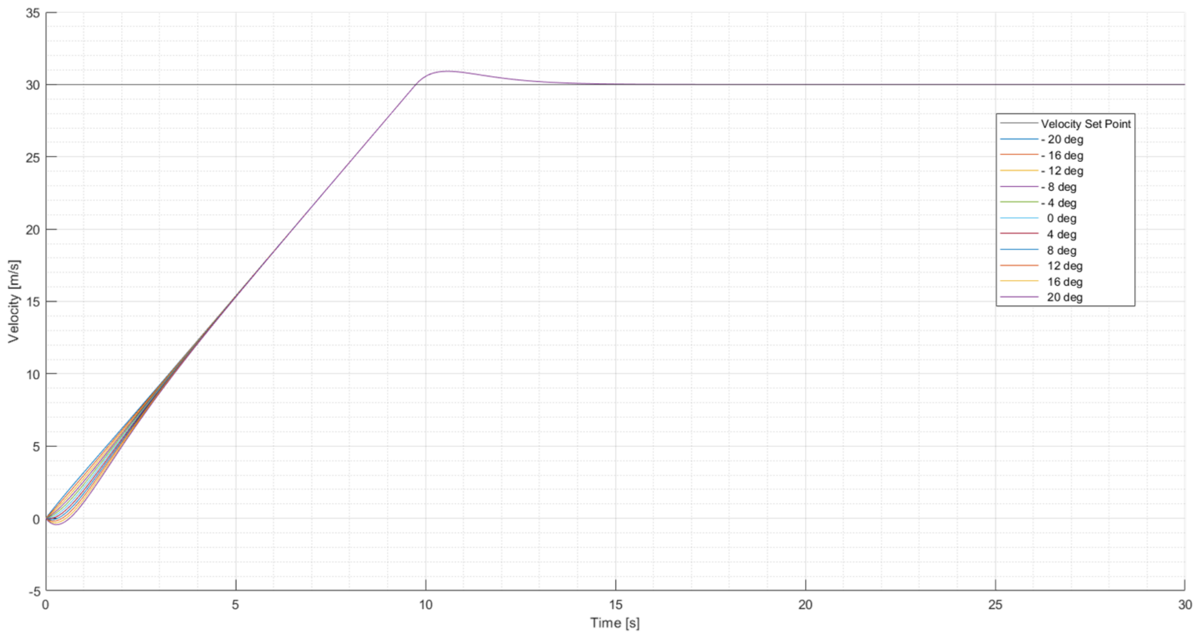

Figure 9 - $K_p = 2.38$, $K_i = 1.44$: Velocity & Time

MATLAB determined for this PI combination that the damping ratio, $\zeta = 1$, therefore the system was critically damped. Even though the damping ratio was critical, there was overshoot. This was because of how the input to the system was modelled. Due to the ramp input, the velocity critically rises to match the ramp, then critically reacts back to the plateaued value at 30 m/s when it was reached.

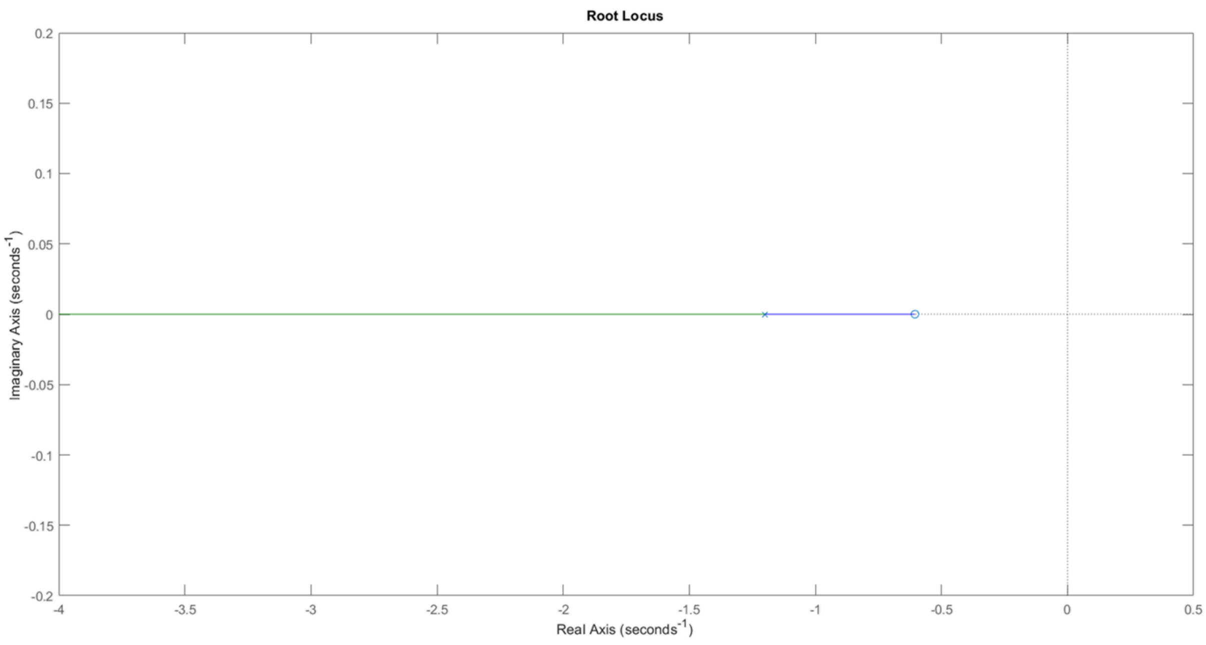

Figure 10 - $K_p = 2.38$, $K_i = 1.44$: Root Locus

Critical damping was verified when looking at the Root Locus plot for the system. The roots had lost their imaginary component, and now sat on the negative real axis.

This demonstrated overdamped and stable behaviour. Furthermore, stability was confirmed at this gradient by observing the infinite gain margin on the Bode Diagram, shown below in Figure 11:

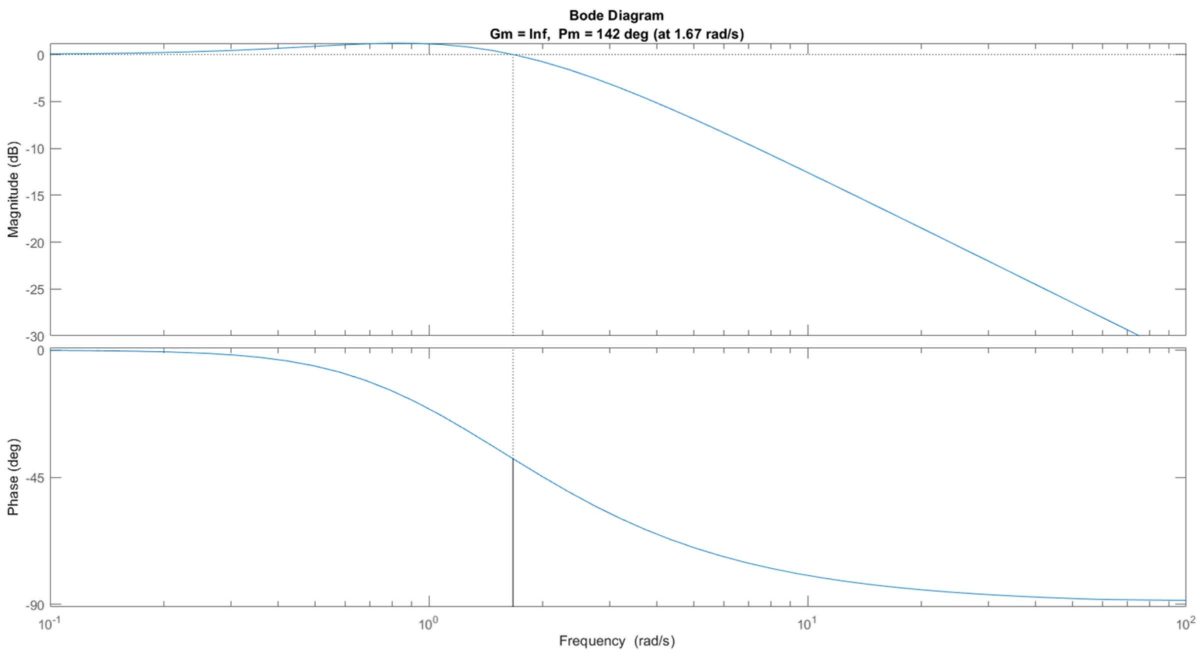

Figure 11 - Kp = 2.38 , Ki = 1.44: Bode Plot

The Routh Hurwitz Array also showed consistent positive values in the first column, further verifying stability:

Routh – Hurwitz Array:	1.00	1.44
	2.40	0
	1.44	0

This controller PI combination was deemed suitable for use. The behaviour at a series of angles showed that after the initial matching with the rising input, it consistently overshot within the desirable limit.

Preference, however, was given to the controller with underdamped behaviour, $K_p = 1.81$, $K_i = 1.47$. It was taken and used for further analysis.

3.2.2 Focussed Stability & Performance Assessment

It was decided that Controller $\mathrm{Kp} = 1.81$, $\mathrm{Ki} = 1.47$ would be carried forward for further detailed study. This controller would now be tested under a number of different conditions to understand its performance and limitations.

The transfer function of the controller was input into MATLAB previously to determine the damping ratio of the system. This transfer function did not include the effects of gradient and therefore was not suitable for analysing gradient effects on stability. The transfer function was modified to include the effects of gradient:

$$T(s) = \frac{K_d s^2 + (K_p - g\theta)s + K_i}{(1 + K_d)s^2 + (0.02 + K_p - g\theta)s + K_i}$$

This new transfer function was now suitable for further analysis.

3.2.2.1 Root Locus Differing Gradient

Gradient was again swept between -20° up to 20°, varying by 4° each step. The root locus plot of the study was recorded in MATLAB and was shown in Figure 12 below:

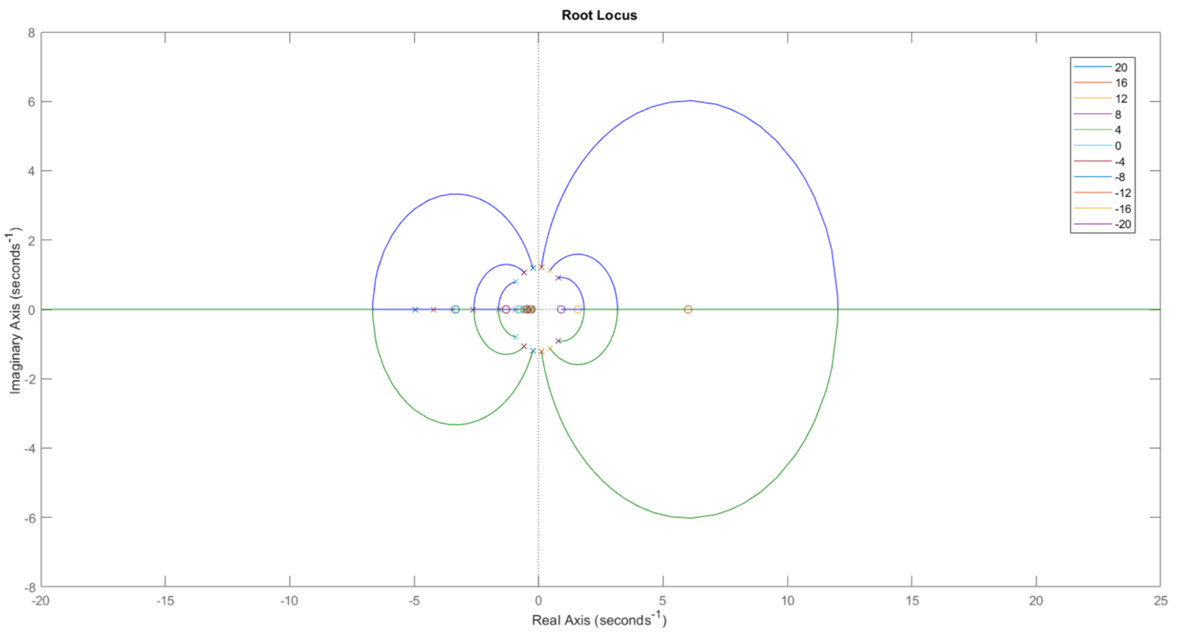

Figure 12 - Kp = 1.81 ,Ki = 1.47 Root Locus Gradient Sweep

The Root Locus plot displayed a large variance in behaviour over the range of angles.

At high up-hill angles, system behaviour turned from underdamped to overdamped as the complex component of the poles reduced to zero. It was noted however that these overdamped cases remained stable as they were on the negative side of the real axis. This was an acceptable change in controller behaviour due to the conditions, however it was unlikely that a 20° uphill section of road would require a cruise control system at all, never mind set to 30 m/s.

Looking over to the other side of the gradient range, as the downhill angle reached 12°, the controller became unstable due to its poles now lying on the positive side of the real axis. The behaviour had also remained underdamped.

Shown in Figure 13, these 3 data points were isolated, allowing them to be more easily seen:

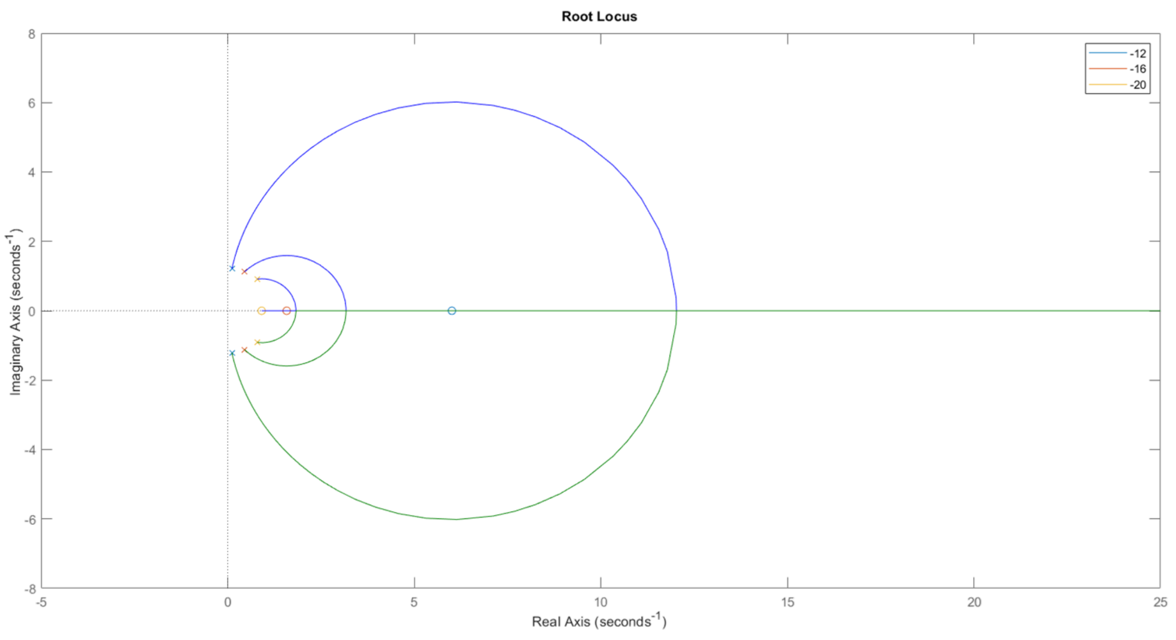

Figure 13 - Kp = 1.81 ,Ki = 1.47 Focussed Root Locus [-12, -16,-20] degrees

All the gradients also corresponded to their own individual Bode Diagrams, shown below in Figure 14. Interestingly, the Bode Diagrams reflect a much more stable setting than described in the Root Locus diagram.

3.2.2.2 Bode Differing Gradient

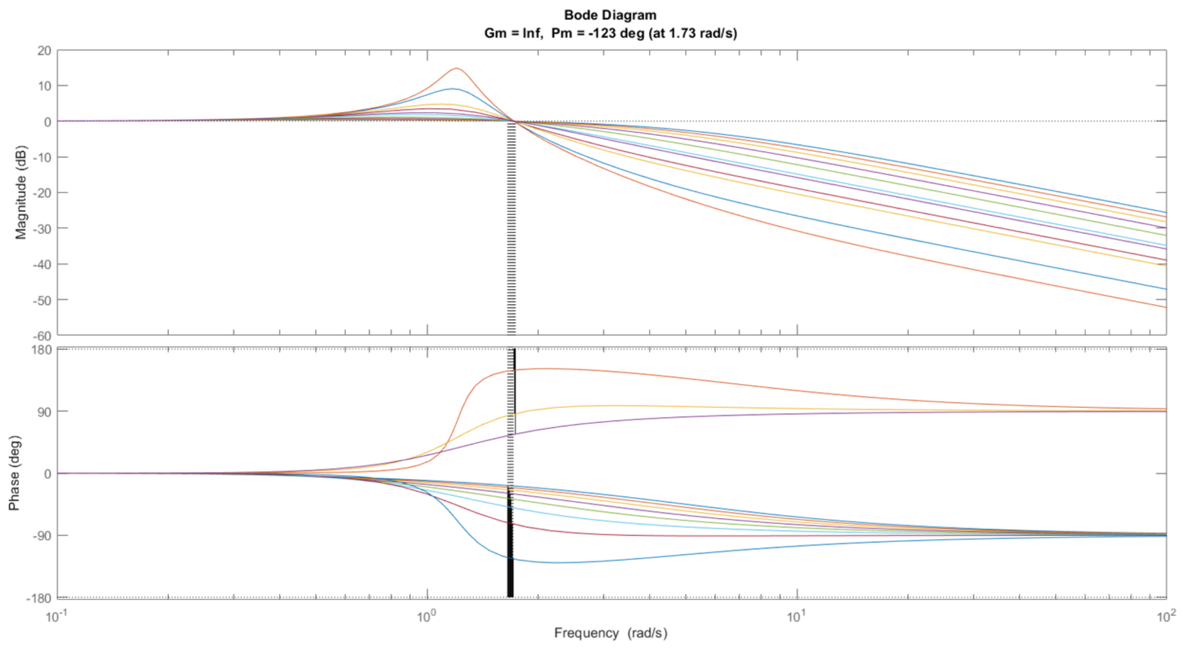

Figure 14 - Kp = 1.81 ,Ki = 1.47 Bode Diagram Gradient Sweep

As commented, the Bode diagram shown in Figure 14 above displayed an infinite gain margin for all gradients. This inferred stability as the phase margin never reached -180^o in any of the gradient cases. Looking to the Routh Hurwitz array for the 12^o case showed:

Routh – Hurwitz Array:	1.00	1.47
	-115.89	0
	1.47	0

The first column of the Routh Hurwitz array displayed a change in sign in the first column, this demonstrateed instability for the downhill 12^o case and those of greater negative gradient.

3.3 Conclusions

PI control was determined to be suitable for the car cruise controller. The derivative element changed the behaviour of the controller, though was not necessary for functionality and was therefore dropped. This was helpful when it came to using the 'optimisation script', as it eliminated a variable that had to be optimised therefore cutting computation time by around a third.

The decided controller gains of $K_p = 1.81$, $K_i = 1.47$ were simulated and their behaviour was desirable. These gains corresponded of a damping ratio of $\zeta = 0.7547$, overshoot within a safe and legal limit, and a reasonable time to decay back to the set speed. For this reason it was chosen for further performance and stability analysis.

In their performance analysis, it was found that $K_p = 1.81$, $K_i = 1.47$ responded well to gradients from -20^o to 20^o. Modelled using a ramp velocity input, it had extremely consistent overshoot characteristics. There was minimal rolling backwards, noted by a negative velocity immediately past t = 0, however this was not representative of reality as car cruise control systems were not typically engaged at zero speed.

Moving on to the stability analysis, it was found that even though through visual observation and the Bode Diagrams noting stability, the Root Locus plot and the Routh-Hurwitz Arrays showed that the controller was unstable when going down a 12^o decline and greater. Though this was undesirable, it was an unrealistic scenario. Roads that a cruise control was likely to be used on i.e. a motorway typically would not reach this steep a decline.

Further analysis and controller adjustment could improve the combination of gains to prevent instability in all cases, though this would likely have a trade off with overshoot or responsiveness. Derivative gain could also be incorporated into the controller though if using the same brute force optimisation technique, it would greatly increase the computational time.

4.0 Part 2

4.1 Methodology

Similar to Part 1, a Simulink model was created for the modelling of the control system. This time, instead of modelling the effects of the car as a simplified version of the equation of motion, all the relevant forces to the car's motion were considered separately and summed afterwards.

These forces were variant to velocity, so now not only was that being fed back to the control unit and scaled from gains, but also used to determine perturbing forces.

4.1.1 Simulink Model

4.1.1.1 Part A – Dynamic Analysis

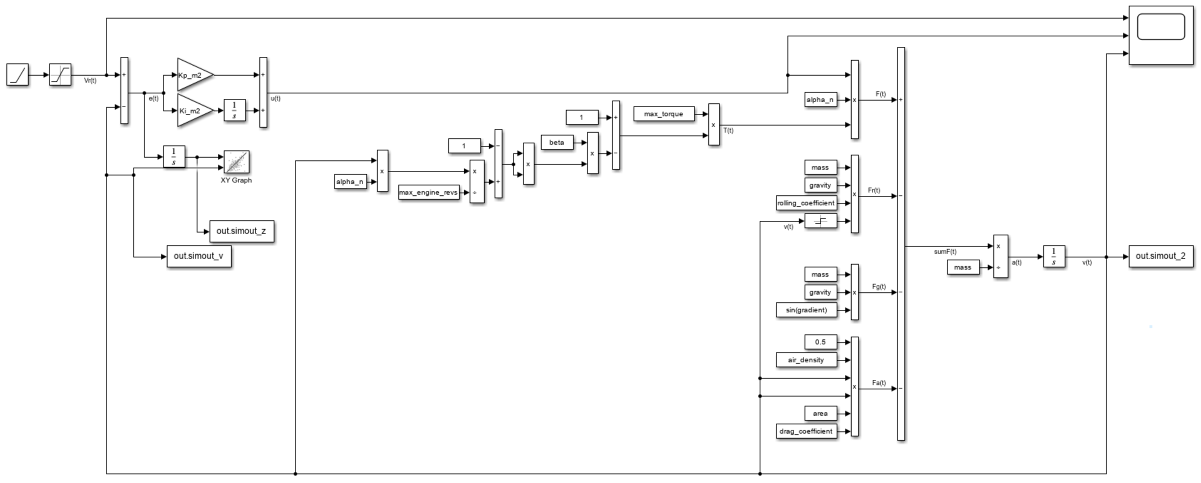

Figure 15 - Part 2A Simulink Model

Shown above in Figure 15 was the Simulink model that was created and used to simulate controller behaviour for Part 2A. Much like Part 1, a ramp input was used to simulate the input speed increase with a saturation block to cap this off at the maximum speed value. These were once again controlled within a MATLAB script which allowed for easy editing of the variables and analysis of the output variables.

4.1.1.2 Part B – Feedback Linearisation

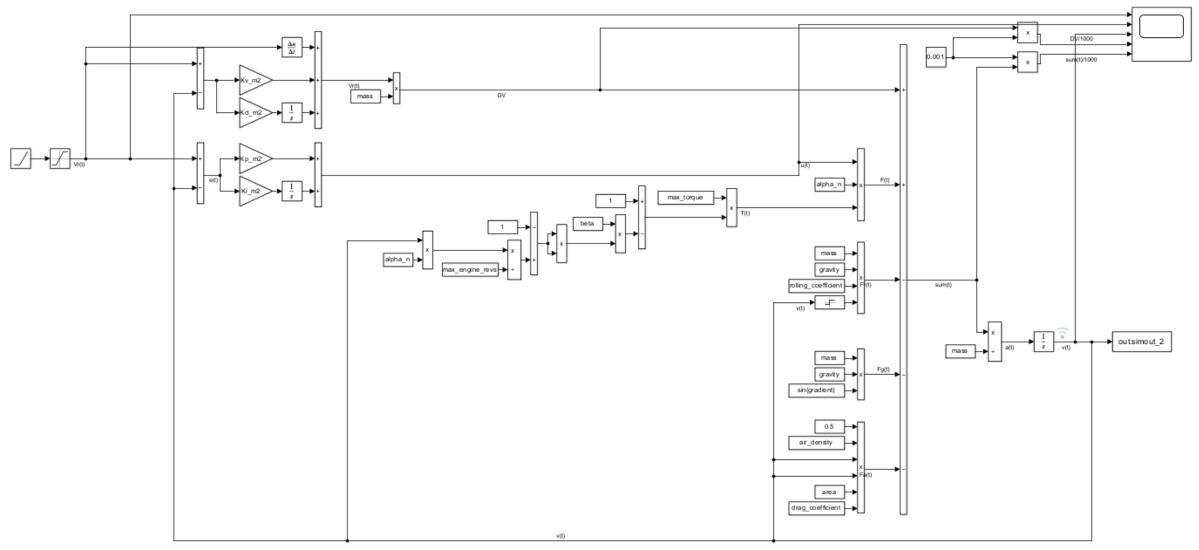

Figure 16 - Part 2B Simulink Model

Figure 16 also used the same ramp and saturation settings, as well as being integrated with its own MATLAB script in order to improve control over input variables and easily provide output data. Unlike part 1, there was no necessary iteration of gains, as these were specified and followed in the project outline documents.

The list of constants that were used in Part A were shown below in Table 1:

Variable	Description	Value	Unit
K_p	Proportional Gain	0.5	[-]
K_i	Integral Gain	0.1	[-]
α_n	Force to torque ratio	16	[m^{-1}]
β	Engine characteristic	0.4	[-]
T_m	Max engine torque	190	[Nm]
Ω_m	Max engine speed	420	[rad s^{-1}]
m	Mass	1000	[kg]
g	Gravitational acceleration	9.81	[m s^{-2}]
C_r	Rolling coefficient	0.01	[-]
C_d	Drag coefficient	0.32	[-]
ρ	Air density	1.225	[kg m^{-3}]
A	Wetted area	2.4	[m^2]
θ	Angle of road	0	[rad]

Table 1 - Part A Variables

These were used in the blocks, shown in Figure 15 and Figure 16, throughout simulations, with the exception of changing α_n for the analysis of Part A.

4.1.2 Dynamic Analysis

Dynamic analysis involved the testing of different values for α_n. Values were iterated using a `for` loop in the controlling MATLAB script. This allowed for fast iteration and the data from each to be stored, to then be displayed graphically.

By integrating controller error at a given timestep, Z was obtained. Z was recorded at each timestep for each α_n value along with velocity. Z and velocity were used to generate phase space graphs. Due to time constraints, this was the only iteration of scenarios in this project.

4.1.3 Feedback Linearised Control

Control torque was expressed in terms of displacement and its subsequent derivatives. The derived equation was shown below in 3:

$$\tau_c = m[k_v(\dot{x}_{sp} - \dot{x}) + k_d(x_{sp} - x) + \ddot{x}_{sp}] + \left[\alpha_n\left(k_p(\dot{x}_{sp} - \dot{x}) + k_i(x_{sp} - x)\right)\left(T_m\left(1 - \beta\left(\frac{\alpha_n \dot{x}}{\Omega_m} - 1\right)^2\right)\right) - \frac{1}{2}\rho A C_d|\dot{x}||\dot{x}|\right] - [mgC_r sgn(\dot{x}) + mg\sin(\theta)] \qquad 3$$

$$0 = (\ddot{x}_{sp} - \ddot{x}) + k_v(\dot{x}_{sp} - \dot{x}) + k_d(x_{sp} - x) \qquad 4$$

Accompanying 3, equation 4 showed a new controller error introduced, containing new gain values, k_v and k_d. As shown in Figure 16, this new error was implemented into the model and used the difference in the set value from the feedback value to influence the output control torque. Control torque was measured and output into MATLAB.

4.2 Results

4.2.1 Part A Behaviour

Data recorded using MATLAB was displayed on figures, shown below.

4.2.1.1 Variance of Gradient

Gradient was varied for the controller of Part A to represent the controller's behaviour on a series of inline/declines. Shown below in Figure 17, there was a considerable amount of rolling forwards or backwards immediately after t = 0. For the majority of cases, this quickly converged back to the ramping input, then overshot within a safe range.

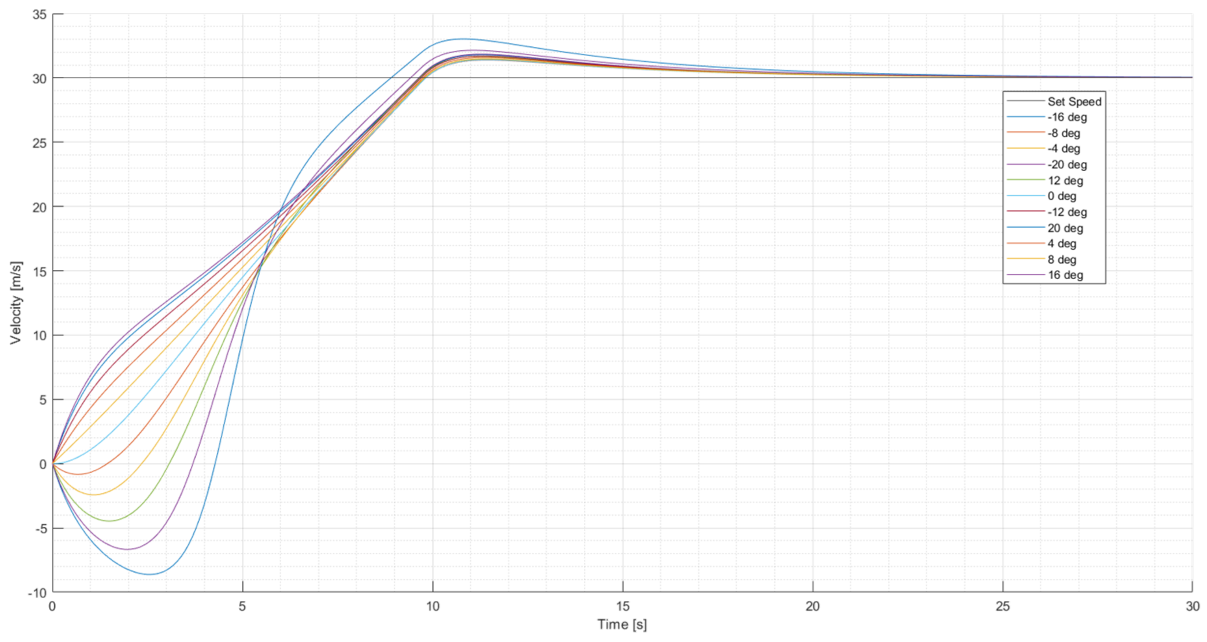

Figure 17 - Kp = 0.5, Ki = 0.1, Dynamic Analysis Gradient Sweep

The outlier to this was the case for 20^o incline, which overshot notably more than the rest of the scenarios.

4.2.1.2 Variance of α_n

Changing α_n was analogous to changing the gear of the car's engine. In doing so, velocity response characteristics were modified, shown below in Figure 18:

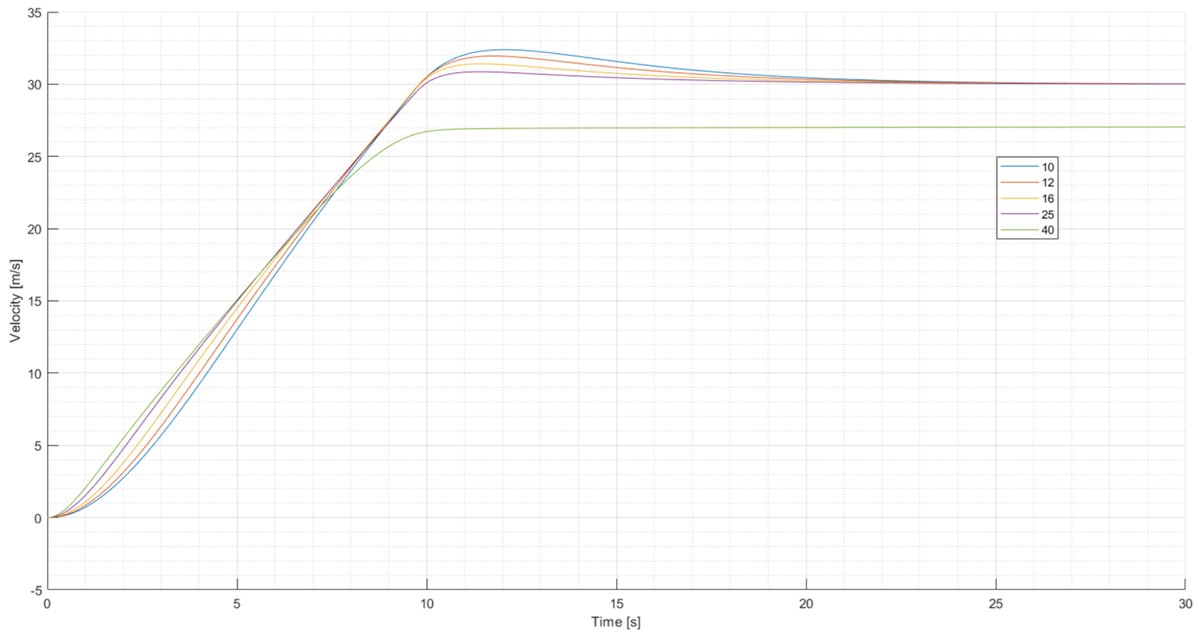

Figure 18 - Kp = 0.5, Ki = 0.1, Dynamic Analysis v_α Sweep

Notably, for the case of α_n = 40, velocity reached a steady state error, plateauing at 27 m/s. It could also be seen that for the same case, it had a higher acceleration than other gear ratios, demonstrating that a greater torque output allowed for higher acceleration with the trade-off of a reduced maximum velocity for a given engine speed.

Plotting the integral of error, Z, against time showed how the error changed over the same timeframe. This was shown below in Figure 19

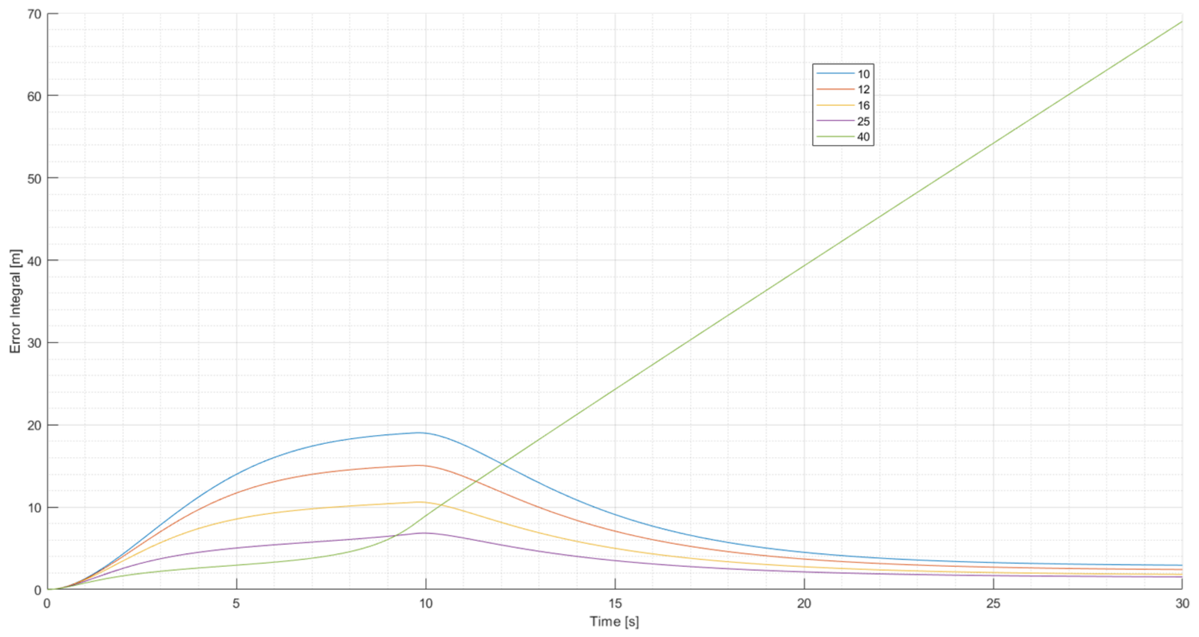

Figure 19 - Kp = 0.5, Ki = 0.1, Dynamic Analysis Error Integral Sweep

Easily visible was the steady state error which caused a linear increase in Z. As long as the gear did not change and the velocity did not increase, then this error would continue to grow. Finally showing the phase space for the controller below in Figure 20:

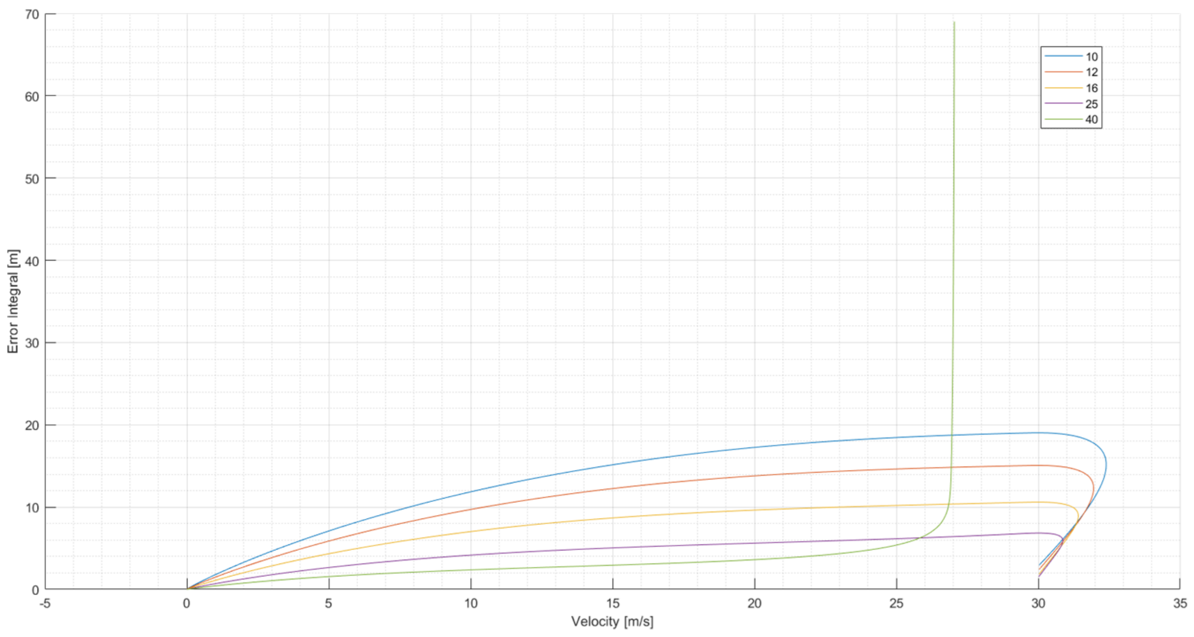

Figure 20 - Dynamic Analysis Phase Space

Once again, the $\alpha_n = 40$ could be seen increasing in error, there was no equilibrium point. For the other values of α_n, an equilibrium point was reached at 30 m/s. As each of the systems did not have oscillating perturbations, like an undulating gradient for example, the phase space did not oscillate around a point. Further modelling would show this style of behaviour.

4.2.2 Part B Behaviour

4.2.2.1 Variance of Gradient

Using the Simulink file shown in Figure 16, the following data was simulated. Visible from the velocity against time graph, this controller also suffered from rolling backwards or forwards at t = 0. The control torque-based controller did however demonstrate a desirable characteristic of minimal overshoot in a way similar to previous controllers.

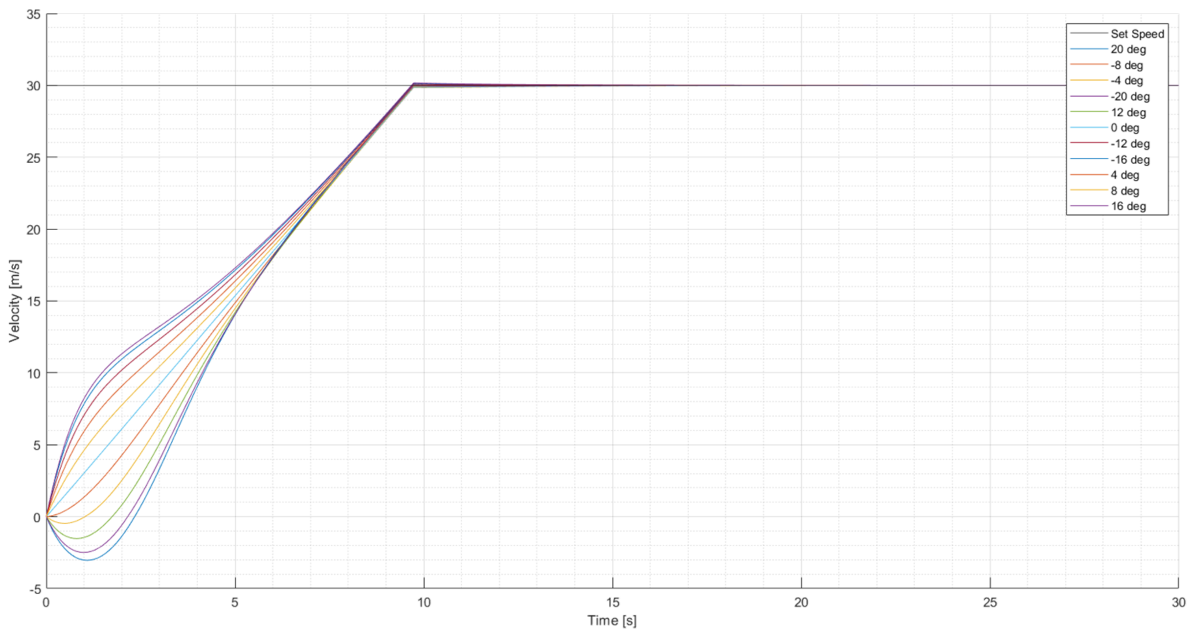

Figure 21 - Part B, Kv = 0.1, Kd = 0.1, Gradient Sweep

This was likely due to the controller being able to immediately cut the input of torque to the correct amount to control the speed of the car. This did highlight the inaccuracy of the lack of lag in engine speed as a car begins or ends acceleration.

4.2.2.2 Variance of α_n

Due to $\alpha_n = 40$ never reaching the desired value, it was difficult to include on any of the relevant graphs, because of this, it was omitted though it should be mentioned that its behaviour was similar to that shown in 4.2.1.

Displayed below in Figure 22 was the Error integral against time plot for a range of α_n values:

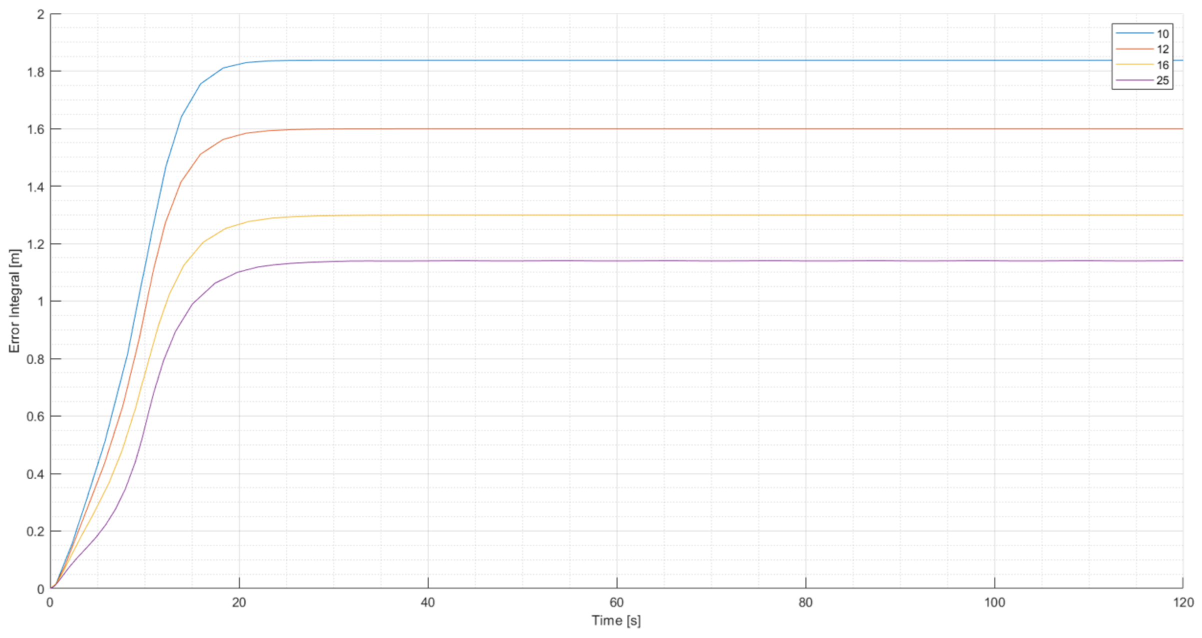

Figure 22 - Control Torque Error Integral Sweep

Similarly to 4.2.1, the integral of the error tended to a constant value over time, showing that the velocity reached the specified point. This was corroborated by data in Figure 23 that showed a velocity tied extremely tightly to the input ramp:

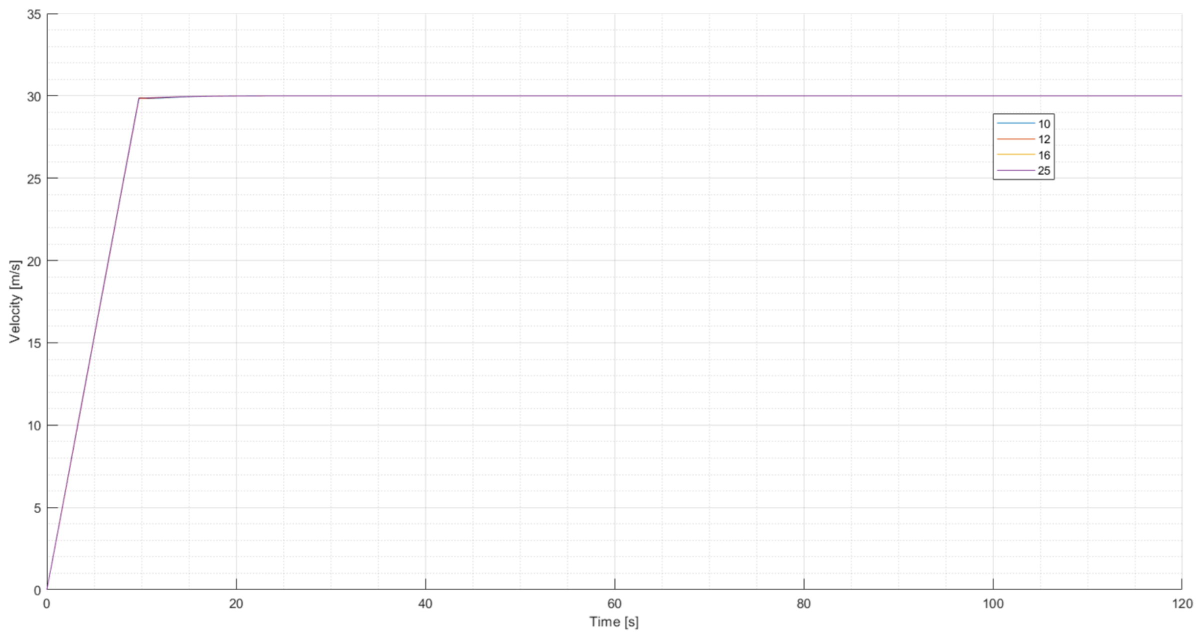

Figure 23 - Control Torque Velocity Sweep

As mentioned, velocity for the range of α_n values here was tied very tightly to the input, with little to no noticeable lag or deviation. This suggested a problem with how the solution was modelled. One method to improve on this would be again to incorporate the engine taking time to rev up and down which was not currently accounted for, as well as this the gains were not heavily optimised in any way.

The following phase space plot was produced for the single case with a range of α_n values:

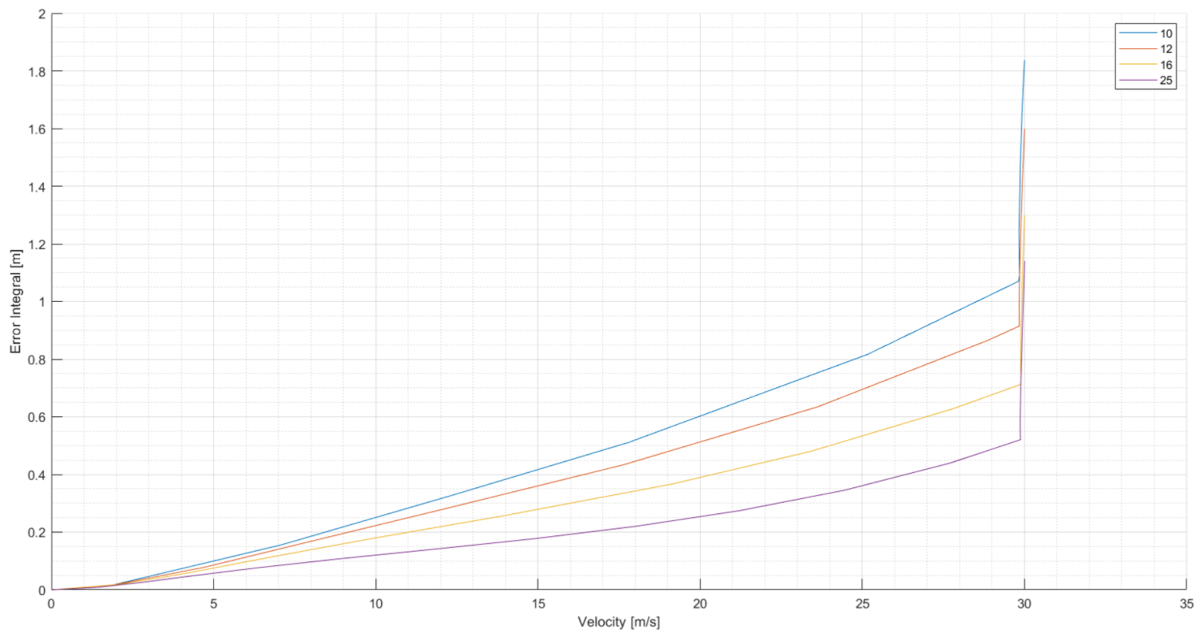

Figure 24 - Control Torque Phase Space

Once again omitting α_n = 40 due to being impractical to display, 4 equilibrium points were being reached by the solution when the velocity was reached with no further deviation.

4.3 Conclusions

To add complexity to the system, drag and resistance forces were modelled within Simulink to replace the simpler equation of motion block used in Part 1. This model more realistically represented the effects of the car's motion on itself.

Equations 3 and 4 were derived using the sum of forces. This represented the control torque, the output torque from the engine that introduced an extra error. The control torque method allowed a much more snappy response than the controller in part 1, however this was not realistic due to the time it would have taken for the engine to speed up and slow down.

Further work should be put in to modelling a more realistic behaviour for the controller, adding features like the engine spooling time and limiting the acceleration of the car to a comfortable, passenger compatible 1G where it was exceeded. As well as this, more perturbation scenarios should be modelled, including changing speeds and gradients.

5.0 Report Conclusions

PI control was determined to be suitable for the car cruise controller. The derivative element changed the behaviour of the controller, though was not necessary for functionality and was therefore dropped. This was helpful when it came to using the 'optimisation script', as it eliminated a variable that had to be optimised therefore cutting computation time by around a third.

The decided controller gains of $\underline{K_p = 1.81 , K_i = 1.47}$ were simulated and their behaviour was desirable. These gains corresponded of a damping ratio of $\zeta = 0.7547$, overshoot within a safe and legal limit, and a reasonable time to decay back to the set speed. For this reason it was chosen for further performance and stability analysis.

In their performance analysis, it was found that $\underline{K_p = 1.81 , K_i = 1.47}$ responded well to gradients from -20^o to 20^o. Modelled using a ramp velocity input, it had extremely consistent overshoot characteristics. There was minimal rolling backwards, noted by a negative velocity immediately past t = 0, however this was not representative of reality as car cruise control systems were not typically engaged at zero speed.

Moving on to the stability analysis, it was found that even though through visual observation and the Bode Diagrams noting stability, the Root Locus plot and the Routh-Hurwitz Arrays showed that the controller was unstable when going down a 12^o decline and greater. Though this was undesirable, it was an unrealistic scenario. Roads that a cruise control was likely to be used on i.e. a motorway typically would not reach this steep a decline.

Further analysis and controller adjustment could improve the combination of gains to prevent instability in all cases, though this would likely have a trade off with overshoot or responsiveness. Derivative gain could also be incorporated into the controller though if using the same brute force optimisation technique, it would greatly increase the computational time.

To add complexity to the system, drag and resistance forces were modelled within Simulink to replace the simpler equation of motion block used in Part 1. This model more realistically represented the effects of the car's motion on itself.

Equations 3 and 4 were derived using the sum of forces. This represented the control torque, the output torque from the engine that introduced an extra error. The control torque method allowed a much more snappy response than the controller in part 1, however this was not realistic due to the time it would have taken for the engine to speed up and slow down.

Further work should be put in to modelling a more realistic behaviour for the controller, adding features like the engine spooling time and limiting the acceleration of the car to a comfortable, passenger compatible 1G where it was exceeded. As well as this, more perturbation scenarios should be modelled, including changing speeds and gradients.

6.0 Suggestions for Modifications

6.1 More Written MyPlace Resources on Bode Plots

This was just a personal nuisance, I never really learned a lot of the control theory content in 3rd year due to a number of factors, I can't even remember if Bode Plots were covered in that year. I struggled to learn what they actually meant using the resources on MyPlace. This basically meant that Bode Plots were completely new, and there also weren't many resources online at all describing much more than what MyPlace did.

6.2 Submission Overlap & Assessment Weighting

I had to request for an extension this semester for a few coursework pieces, it has been a tough semester for me. Like many other folks, I am also plagued by the 'panic monkey' workflow a lot of the time of realising a deadline is coming then the productivity alarm bells ringing.

I've found for myself that deadlines are better motivators than 'just do it' and for my very subjective view I'd suggest physically splitting parts 1 and 2 into 2 different presentations at different points in the semester but keeping one report. Having an earlier part 1 presentation means people are more likely to be up to speed for the changes in part 2 and less likely to, like me, become extremely overwhelmed very quickly. It also allows for people to modify their part 1 with constructive feedback and inspiration from fellow students if it is involved in the part 2 submission in some way.

This might cause problems with the other part of this point though, submission overlap. Control was due on the same day as pressurised systems, another class where the argument of 'just don't fall behind' can be made but again – panic monkey workflow and semester has been tough.

I don't really know what to suggest for solving a conflicting submission date problem as modules will always need to issue work, and we will always need to manage how we do that work. It's more of a problem where its not easy but that's because this is intrinsically hard stuff.

6.3 Choices for control system

I was personally more than fine with the car cruise control topic, but it could be cool in future if people had the option to choose another slightly different scenario with the same – have thing, have variable, control variable, increase complexity – model.

One that comes to mind is an element of an aircraft's autopilot, this could be speed again much like the cruise control except controlling a propellor instead of wheels in the case of a plane, or another option for an aircraft, rocket, plane, missile, etc. could be pitch, yaw, and roll control and ensuring these keep to a specified heading / altitude, I could understand though if the equations responsible for control surface deflection and forces are a bit alien but then the element of choice comes in!

If all the different subtopics were roughly the same difficulty / complexity then there would be no real advantage or disadvantage choosing one over the other and for people who weren't as familiar and didn't want to be, focussing on a car's cruise control would still be there as an option, I do realise that this is easier said than done though and making sure it is suitable for the course isn't as simple as coming up with an idea and slapping it into practice.

7.0 References

[1] RAC, “How to use cruise control,” RAC, 2023. [Online]. Available: https://www.rac.co.uk/drive/advice/driving-advice/cruise-control/. [Accessed 2023].