Coventry University

7146CEM: Automotive Software Engineering - Design and Development

Coursework: Cruise control system and Motor speed control system

Developed by : Vignesh Babu Aravamuthan (adnanv@coventry.ac.uk)

Student ID : 11911348

Table of content

Table of figures

Table of tables

1 Introduction

This document contains information and details regarding the workflow used to create the PID controller, tuning of PID, and generation of code for cruise control and motor speed control project. GitHub is used as a version control system for the project. It is integrated with MATLAB to easily facilitate the GitHub process.

Control systems are needed when the system deals with continuously varying parameters or external disturbances such as load, friction, wind, etc, which will affect the output of the system. To maintain the stable output of the system even in the presence of external disturbances a controller is needed to control those output variations. There are a lot of controlling algorithms available for the control system, we will use only a PID controller for cruise control and motor speed.

2 Software Development Life Cycle

This section gives an overall view of the software development process used to develop the cruise control and motor speed system, starting with requirements, design, development, testing, and validation.

2.1 Requirement Gathering

This is the first stage of the V-development cycle that contains a detailed understanding of requirements and expectations for the final product.

2.1.1 PID controller

2.1.1.1 Technical Requirements

- To design the PID controller using the following equations

$$y(k) = y_p(k) + y_i(k) + y_d(k)$$

Where,

$$y_p(k) = K_p e(k)$$

$$y_i(k) = y_i(k-1) + K_i T_s e(k)$$

$$y_d(k) = \frac{K_d}{T_s}\left[e(k) - e(k-1)\right]$$

$$Ts = 0.01$$

- PID Controller block should contain discrete function blocks.

2.1.1.2 Non-Functional Requirement

- PID controller model should be designed and convert to referenced model.

2.1.2 Cruise Control

2.1.2.1 Function Requirement

The system should have the following functional requirement:

- Speed of the car should not fluctuate for the external disturbances.

2.1.2.2 Technical Requirement

The system should have the following technical requirements

S. No.	Requirements
1	Rise time < 10s
2	Overshoot < 10%
3	Stead state error <1%

Table 1 Technical Requirements for Cruise Control

2.1.3 Motor Speed

2.1.3.1 Technical Requirement

The system should have the following technical requirements

S. No.	Requirements
1	Rise time < 5s
2	Overshoot < 5%
3	Stead state error <1%

Table 2 Technical Requirements for Motor Speed Control

2.1.4 Code

Developed code for the controller should have the following requirements:

- Code should be optimized for RAM efficiency.
- To develop the code as per ISO26262.
- Code should follow MISRA C Guidelines.

2.2 Design

2.2.1 PID Controller

2.2.1.1 P Controller Design

P controller is designed by implementing the following equation in Simulink. The following is model is made as a separate subsystem to make it a modular design.

$$y_p(k) = K_p e(k)$$

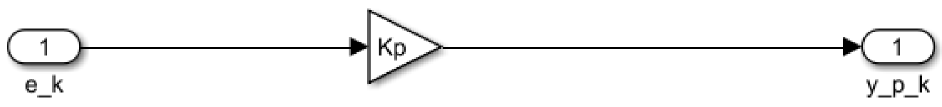


Figure 1 P Controller Implementation

2.2.1.2 I Controller Design

I controller is designed by implementing the following equation in Simulink. The following is model is made as a separate subsystem to make it a modular design.

$$y_i(k) = y_i(k-1) + K_i T_s e(k)$$

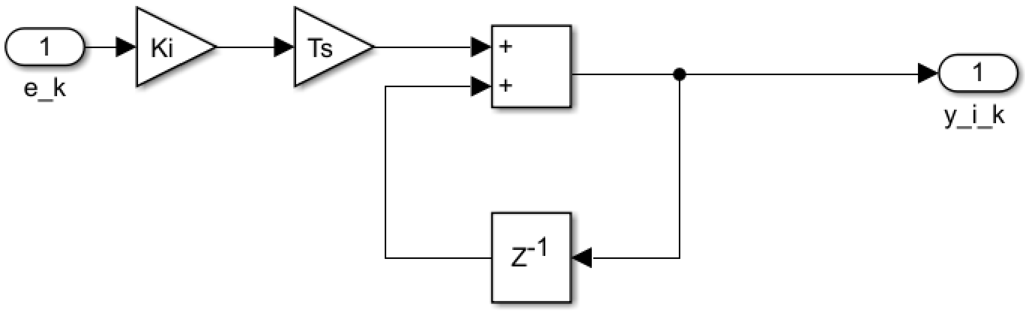


Figure 2 I Controller Implementation

2.2.1.3 D Controller Design

D controller is designed by implementing the following equation in Simulink. The following is model is made as a separate subsystem to make it a modular design.

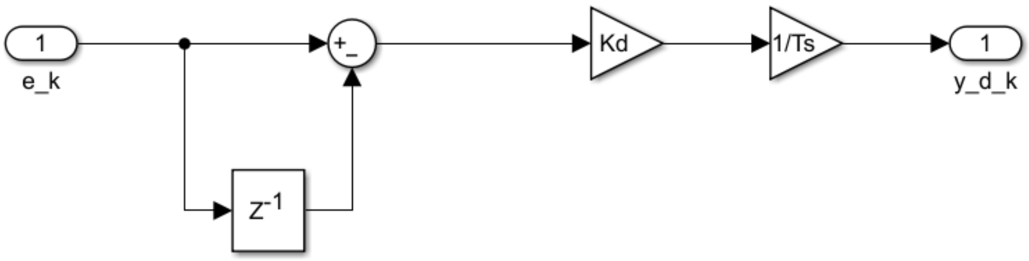


Figure 3 D controller Implementation

2.2.1.4 PID Controller Design

P, I, D controllers which are designed as a separate subsystem are integrated by implementing the following equation (refer to Figure 4). The Sampling time of the PID Controller is set to value $Ts = 0.01$.

$$y(k) = y_p(k) + y_i(k) + y_d(k)$$

This model is then converted to an atomic subsystem to make it a referenced subsystem. The referenced subsystem is useful in case of multiple definitions of the same PID controller are needed. This referenced subsystem is then used for the Cruise control model and Motor speed control.

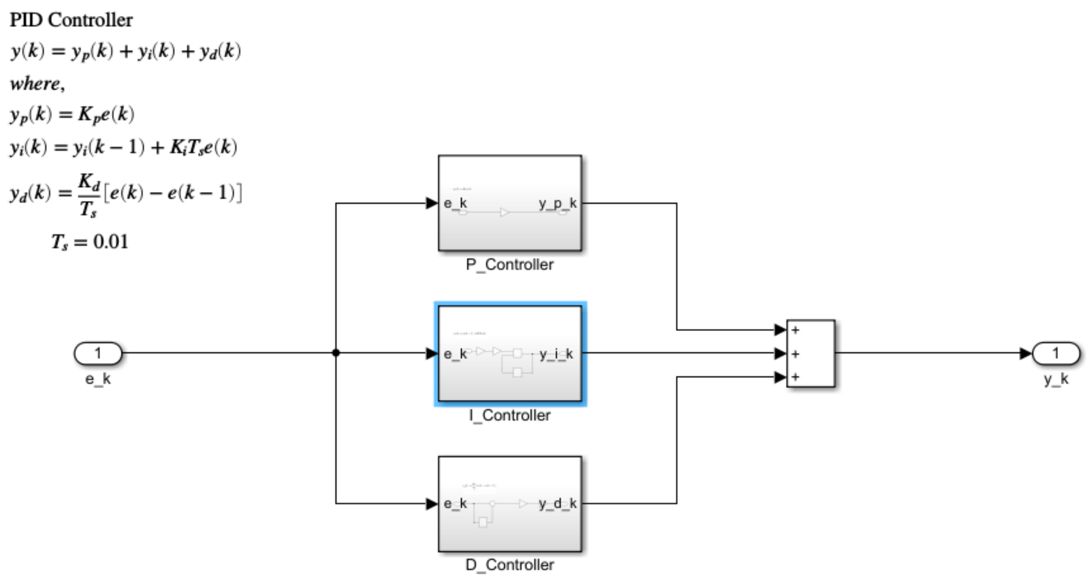
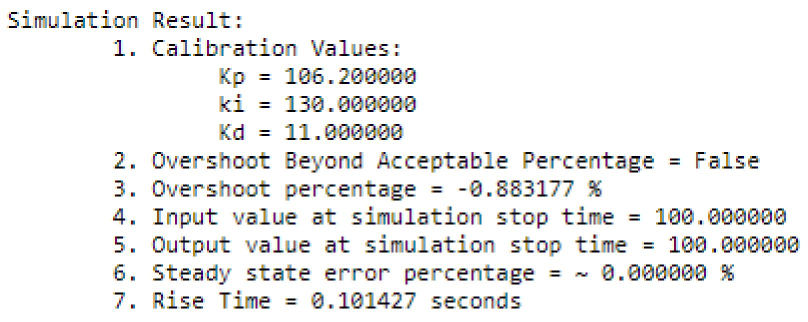


Figure 4 PID Controller Implementation

2.2.2 PID Tuning Algorithm

Script ‘PID_Turning_Script.mlx’ is created to obtain K_p, K_i and K_d values to satisfy the requirements. Script uses the trial-and-error method to find the values. The calculation for overshoot, rise time, steady-state error for the output signal is to be implemented in the script to parallelly check whether the requirements are met with corresponding K_p, K_i and K_d values.

The script will also generate a short report (refer to Figure 5) of the output signal and its properties such as rise time, overshoot percentage, output value at simulation stop time, steady-state error percentage for the corresponding K_p, K_i and K_d values.



```
Simulation Result:
        1. Calibration Values:
                Kp = 106.200000
                ki = 130.000000
                Kd = 11.000000
        2. Overshoot Beyond Acceptable Percentage = False
        3. Overshoot percentage = -0.883177 %
        4. Input value at simulation stop time = 100.000000
        5. Output value at simulation stop time = 100.000000
        6. Steady state error percentage = ~ 0.000000 %
        7. Rise Time = 0.101427 seconds
```

Figure 5 Sample Simulation Report

2.2.3 Cruise Control

2.2.3.1 *Analysis*

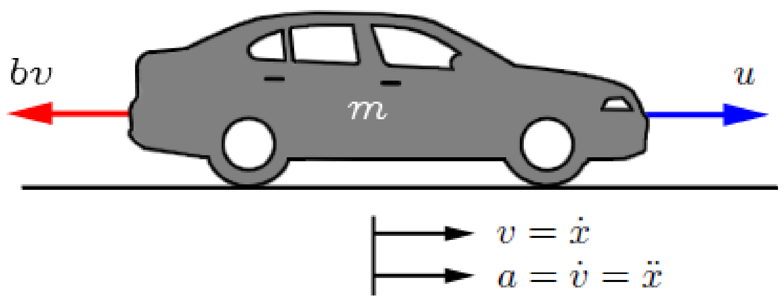


Figure 6 Cruise Control Schematic

The vehicle, of mass m, is acted on by a control force, u. The force u represents the force generated at the road/tire interface. For this simplified model we will assume that we can control this force directly and will neglect the dynamics of the powertrain, tires, etc., that go into generating the force. The resistive forces, bv, due to rolling resistance and wind drag, are assumed to vary linearly with the vehicle velocity, v, and act in the direction opposite the vehicle's motion.

2.2.3.2 *System equations*

With these assumptions, we are left with a first-order mass-damper system. Summing forces in the x-direction and applying Newton's 2nd law, we arrive at the following system equation:

$$m\dot{v} + bv = u$$

$$y = v$$

2.2.3.3 *System parameters*

Parameters of the system are assumed with the following values:

- Mass of the Vehicle (m) = 1000 Kg
- Damping Coefficient (b) = 50 $N.s/m$

2.2.4 Motor Speed

2.2.4.1 Analysis

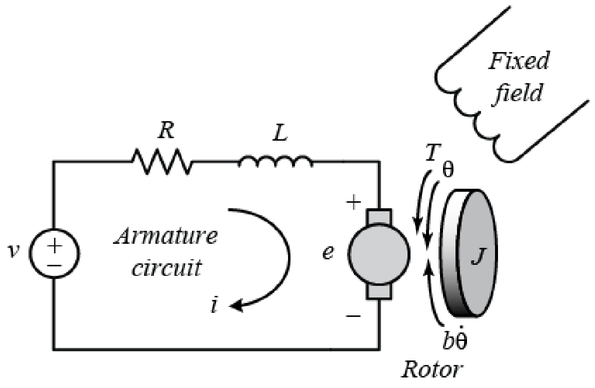


Figure 7 Motor Speed Schematic

A common actuator in control systems is the DC motor. It directly provides rotary motion and, coupled with wheels or drums and cables can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.

The input of the system is the voltage source (v) applied to the motor's armature, while the output is the rotational speed of the shaft $\dot{\theta}$. The rotor and shaft are rigid. The friction torque is proportional to shaft angular velocity.

2.2.4.2 System Equations

The torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. We will assume that the magnetic field is constant and, therefore, that the motor torque is proportional to only the armature current i by a constant factor K_t as shown in the equation below. This is referred to as an armature-controlled motor.

$$T = K_t$$

The back emf, e, is proportional to the angular velocity of the shaft by a constant factor K_e

$$e = K_e\dot{\theta}$$

The motor torque and back emf constants are equal, that is, $K_t = K_e$; therefore, we will use K to represent both the motor torque constant and the back emf constant. We can derive the following governing equations based on Newton's 2nd law and Kirchhoff's voltage law.

$$J\ddot{\theta} + b\dot{\theta} = Ki$$

$$L\frac{di}{dt} + Ri = V - K\dot{\theta}$$

2.2.4.3 System Parameters

Parameters of the system are assumed with the following values:

• Moment of inertia of the rotor (J)	=	0.01 Kgm^2
• Motor viscous friction constant (b)	=	0.1 Nms
• Electromotive force constant (K_e)	=	0.01 $\frac{v}{rad}/sec$
• Motor torque constant (k_t)	=	0.01 Nm/Amp

- Electric resistance (R) = 1 $Ohms$
- Electric inductance (L) = 0.5 H

2.3 Development and Coding

2.3.1 Cruise Control

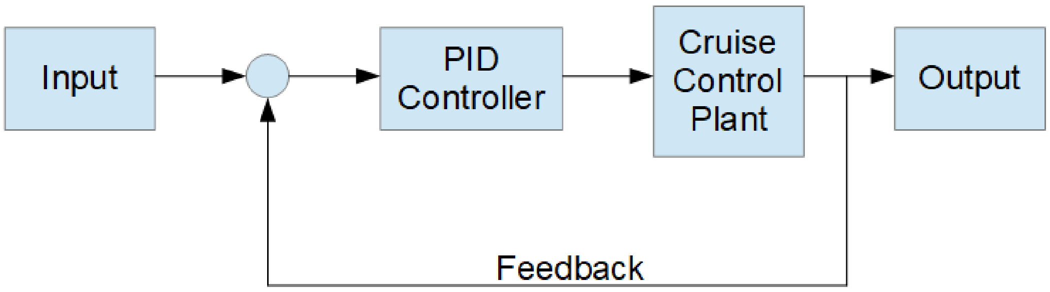


Figure 8 Cruise Control Block Diagram

The Cruise control system is planned to implement as displayed in Figure 8. Here, Input will be replaced by a 'Step input' block. For the PID controller, the referenced subsystem which is created in task 1 is used. The Cruise control plant is modelled (refer to Figure 9) using the mathematical equation defined in 2.2.3.2 above, which is derived from the analysis mentioned in 2.2.3.1 above. With this, the model is developed in Simulink (refer to Figure 10).

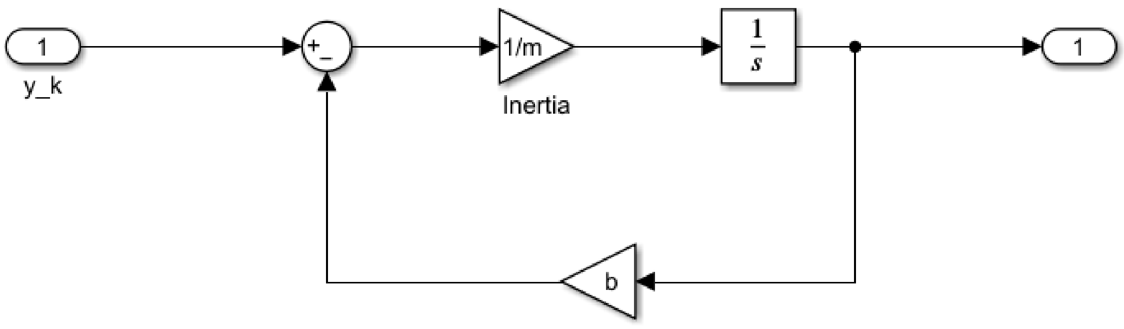


Figure 9 Cruise Plant

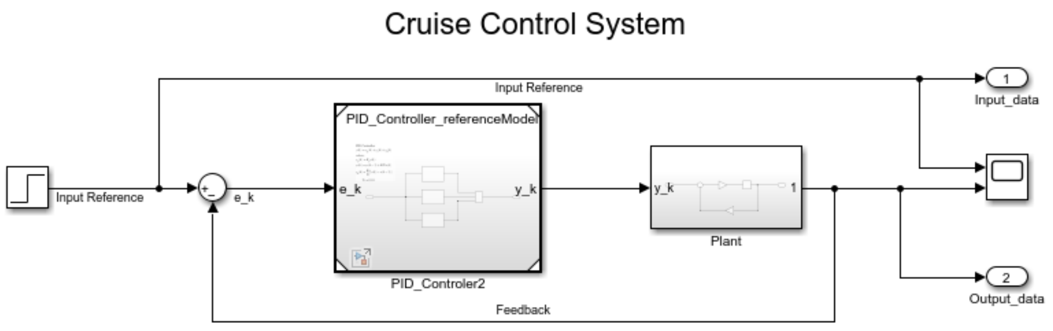
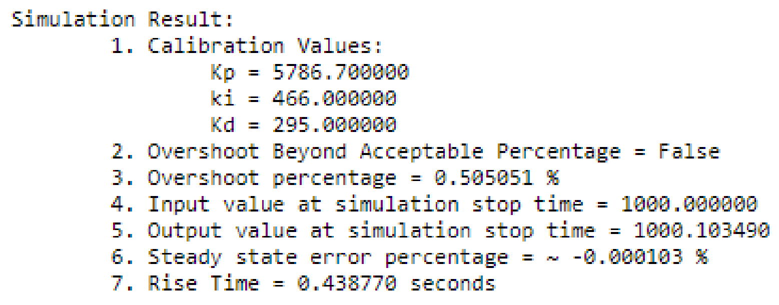


Figure 10 Cruise Control Model

Parameter b and m are needed to complete the design, so values are assumed as mentioned in 2.2.3.3 above, and saved the value in MATLAB base workspace. K_p, K_i and K_d values are tuned using the script which is developed. By trial and error method, K_i and K_d

values are fixed and varied the K_p parameter once the output signal has reached the desired value then K_p value is set as constant and varied K_i alone, by increasing K_i value it is observed that steady state error was decreasing, when the steady-state error percentage is less than the percentage mentioned in requirements of steady state error K_i values is kept constant, now to remove the overshoot created by increased K_i, K_d is increased to suppress the overshoot. Once the overshoot percentage is less than the percentage mentioned in the requirements of overshoot, calibration is stopped. Now, the PID controller is tuned to meet the requirements. Figure 11, Figure 12, Figure 13 are the simulation result, rise time plot, and overshoot plot respectively is the generated report from the ‘‘PID_Turning_Script.mlx’’ script using the Tuned PID Controller.



```
Simulation Result:
        1. Calibration Values:
                Kp = 5786.700000
                ki = 466.000000
                Kd = 295.000000
        2. Overshoot Beyond Acceptable Percentage = False
        3. Overshoot percentage = 0.505051 %
        4. Input value at simulation stop time = 1000.000000
        5. Output value at simulation stop time = 1000.103490
        6. Steady state error percentage = ~ -0.000103 %
        7. Rise Time = 0.438770 seconds
```

Figure 11 Cruise Control System Simulation Result

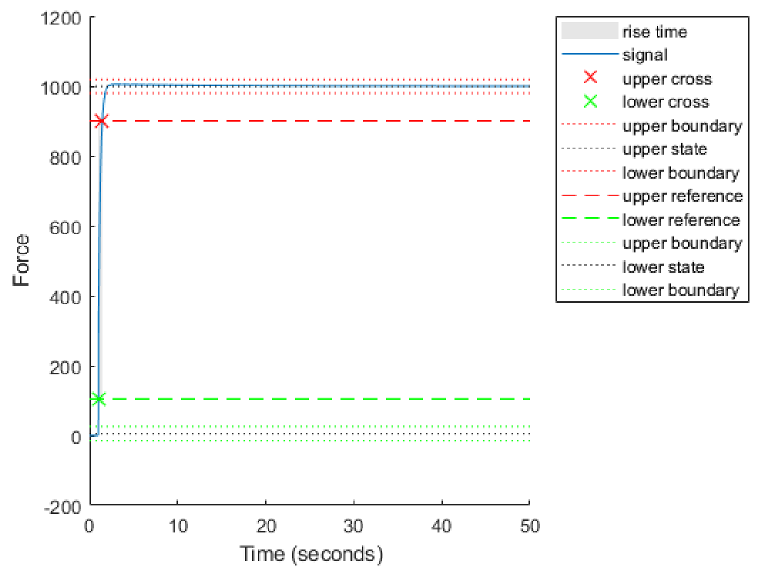


Figure 12 Cruise Control Plot for Rise Time Analysis

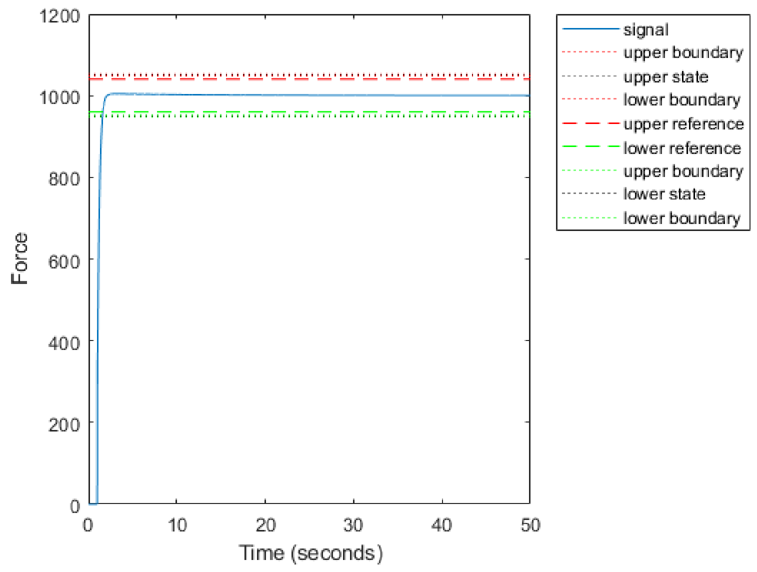


Figure 13 Cruise Control Plot for Overshoot analysis

2.3.2 Motor Speed Control

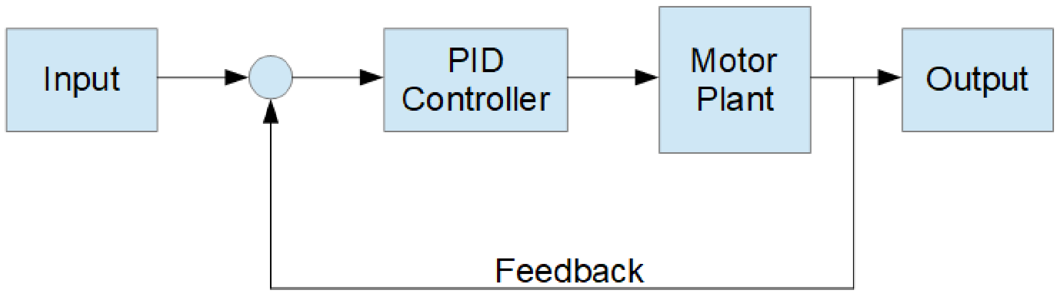


Figure 14 Motor Speed Control Block Diagram

The Motor speed control system is planned to implement as displayed in Figure 14. Here, Input will be replaced by a ‘Step input’ block. For the PID controller, the referenced subsystem which is created in task 1 is used. The Motor speed control plant is modelled (refer to Figure 15) using the mathematical equation defined in 2.2.4.2 above, which is derived from the analysis mentioned in 2.2.4.1. With this, the model is developed in Simulink (refer to Figure 16).

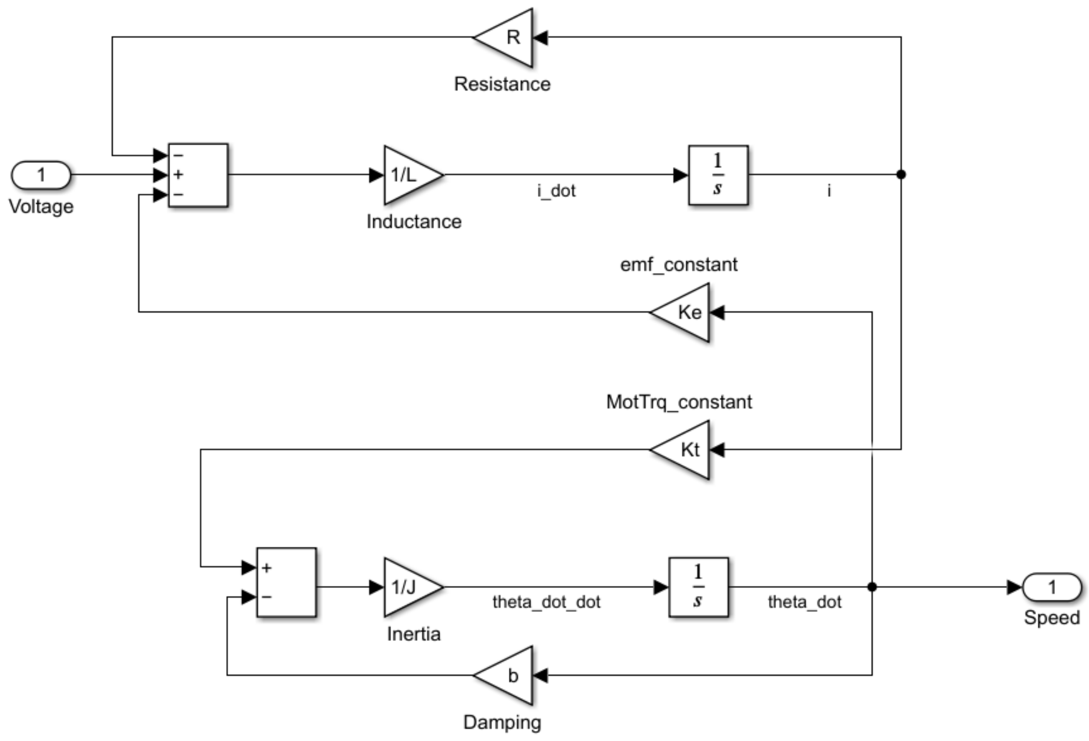


Figure 15 Motor plant

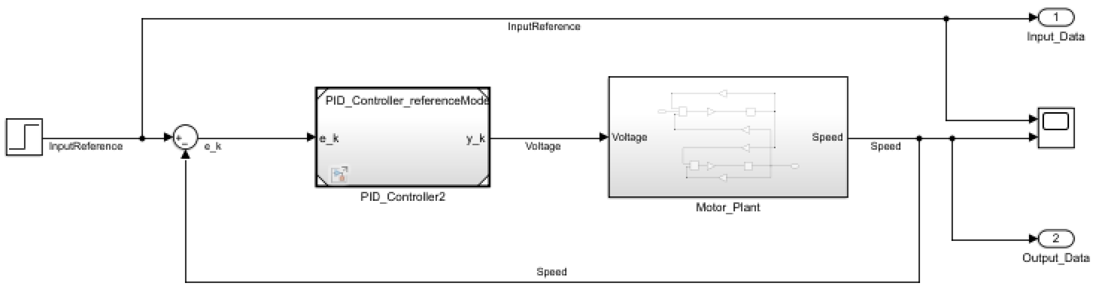
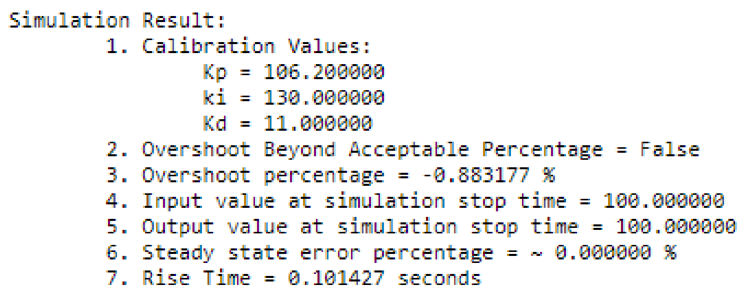


Figure 16 Motor Speed Control Model

Parameter L, R, K_e, K_t, J, and b are needed to complete the design, so values are assumed as mentioned in 2.2.4.3, and saved the value in MATLAB base workspace. K_p, K_i and K_d values are tuned using the same method and the script used in the cruise control development (refer 2.3.1 above). Figure 17, Figure 18, Figure 19 are the simulation result, rise time plot, and overshoot plot respectively, are the generated report from the ‘‘PID_Turning_Script.mlx’’ script using the Tuned PID Controller.



```
Simulation Result:
        1. Calibration Values:
                Kp = 106.200000
                ki = 130.000000
                Kd = 11.000000
        2. Overshoot Beyond Acceptable Percentage = False
        3. Overshoot percentage = -0.883177 %
        4. Input value at simulation stop time = 100.000000
        5. Output value at simulation stop time = 100.000000
        6. Steady state error percentage = ~ 0.000000 %
        7. Rise Time = 0.101427 seconds
```

Figure 17 Motor Speed Control System Simulation Result

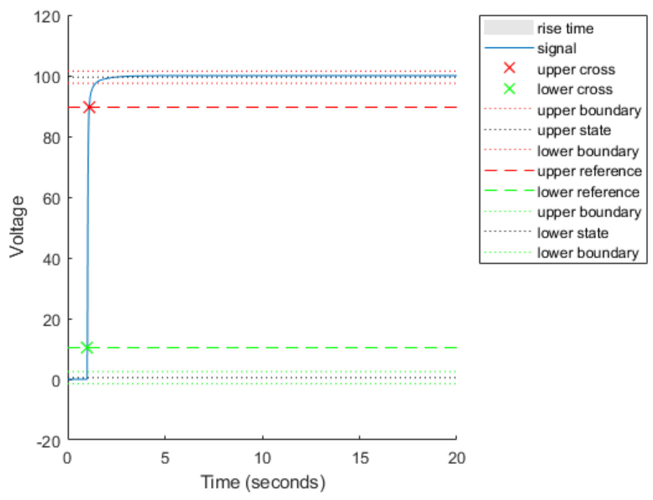


Figure 18 Motor Speed Control Plot for Rise Time Analysis

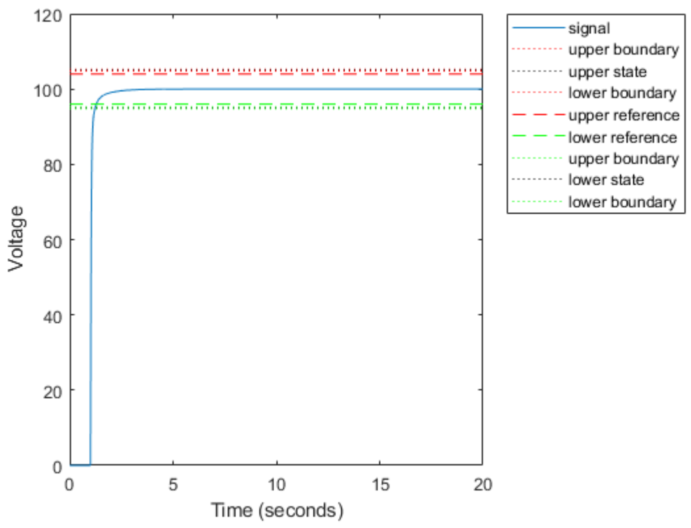
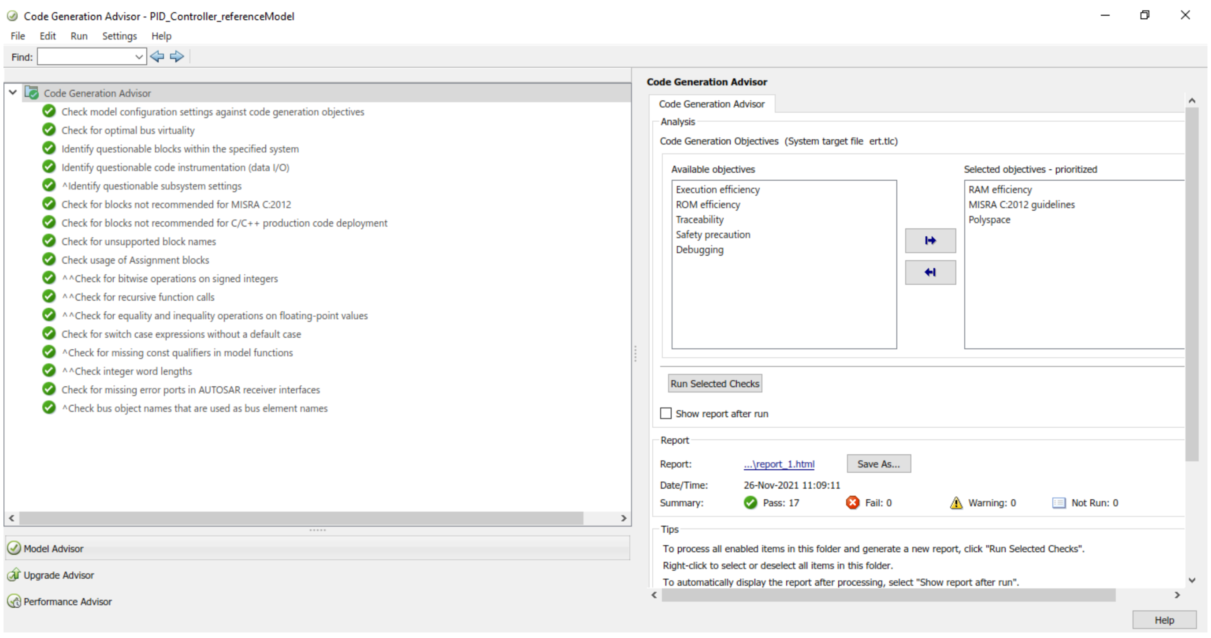


Figure 19 Motor Speed Control Plot for Overshoot Analysis

2.3.3 Code generation

Code is generated using the 'Embedded coder' application in Simulink. According to requirement code should design for 'RAM Efficiency'. All the configuration setting in the code generation is set to meet the requirement. Code generation advisor is used to check whether the generated code is for 'RAM Efficiency' (refer to Figure 20).


Code Generation Advisor - PID_Controller_referenceModel

File Edit Run Settings Help

Find:

Code Generation Advisor
- ✓ Check model configuration settings against code generation objectives
- ✓ Check for optimal bus virtuality
- ✓ Identify questionable blocks within the specified system
- ✓ Identify questionable code instrumentation (data I/O)
- ✓ ^Identify questionable subsystem settings
- ✓ Check for blocks not recommended for MISRA C:2012
- ✓ Check for blocks not recommended for C/C++ production code deployment
- ✓ Check for unsupported block names
- ✓ Check usage of Assignment blocks
- ✓ ^^Check for bitwise operations on signed integers
- ✓ ^^Check for recursive function calls
- ✓ ^^Check for equality and inequality operations on floating-point values
- ✓ Check for switch case expressions without a default case
- ✓ ^Check for missing const qualifiers in model functions
- ✓ ^^Check integer word lengths
- ✓ Check for missing error ports in AUTOSAR receiver interfaces
- ✓ ^Check bus object names that are used as bus element names

Model Advisor
Upgrade Advisor
Performance Advisor

Code Generation Advisor

Code Generation Advisor

Analysis

Code Generation Objectives (System target file ert.tlc)

Available objectives	Selected objectives - prioritized
Execution efficiency	RAM efficiency
ROM efficiency	MISRA C:2012 guidelines
Traceability	Polyspace
Safety precaution	
Debugging	

Run Selected Checks

☐ Show report after run

Report

Report: ...\report_1.html Save As...

Date/Time: 26-Nov-2021 11:09:11

Summary: ✓ Pass: 17 Fail: 0 Warning: 0 Not Run: 0

Tips

To process all enabled items in this folder and generate a new report, click "Run Selected Checks".
Right-click to select or deselect all items in this folder.
To automatically display the report after processing, select "Show report after run".

Help

Figure 20 Code Generation Advisor Report

2.4 Testing

Test harness is created for 'PID.slx'. Model is tested for coverage, execution and response to various inputs.

2.4.1 Coverage/Execution Testing

Model in Loop test is done for the referenced PID model. After the complete test, it is found that execution of the model is 100% (refer to Figure 22).

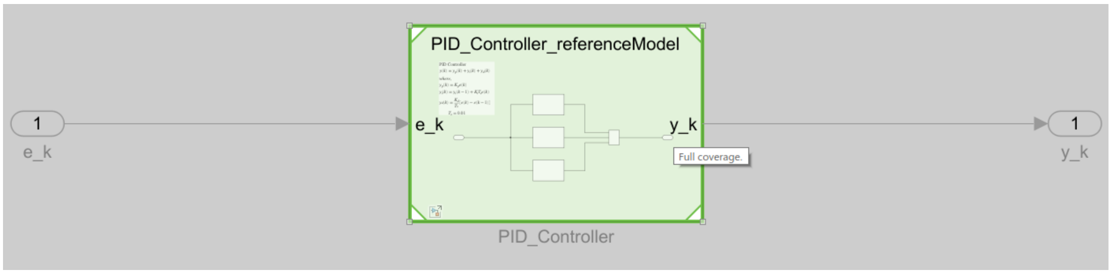
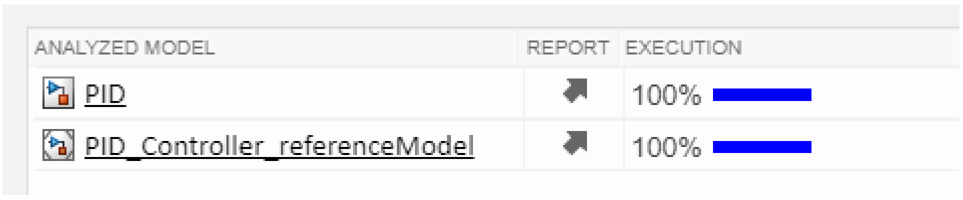


Figure 21 Coverage of the Model



ANALYZED MODEL	REPORT	EXECUTION
PID		100%
PID_Controller_referenceModel		100%

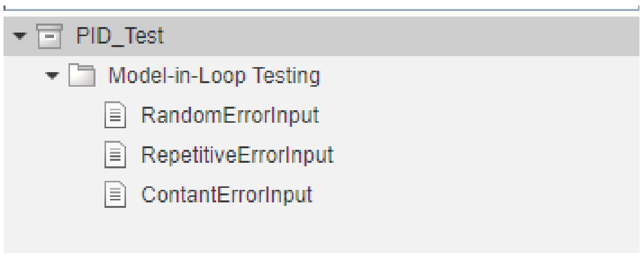
Figure 22 Execution for the Model

2.4.2 Scenario testing

Three different scenario is created in the 'Test Manager' to test the PID controller (refer to Figure 23). PID controller takes error as an input, so the input which is fed to the PID is an error signal. Scenarios are listed below:

- Random error input

- Repetitive error input
- Constant error input



PID_Test
Model-in-Loop Testing
RandomErrorInput
RepetitiveErrorInput
ContantErrorInput

Figure 23 Test cases

2.4.2.1 *Random Error Input Test*

A random error input (refer to Figure 24) is given to the PID controller to test its response, the output given by the PID is displayed in Figure 25.

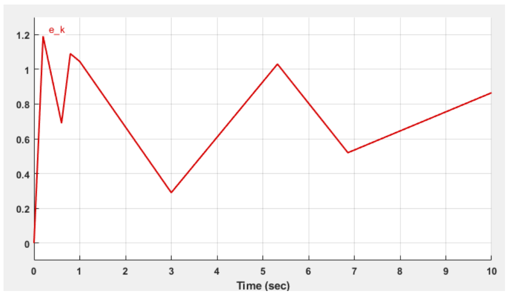


Figure 24 Random Error Input Signal

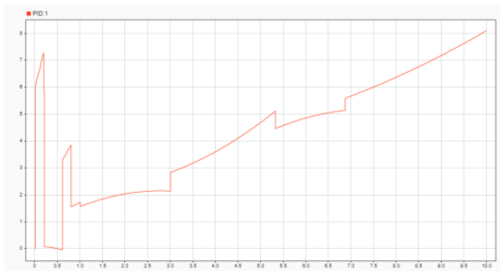


Figure 25 Output for Random Input

2.4.2.2 *Repetitive Error Input Test*

A repetitive error input (refer to Figure 26) is given to the PID controller to test its response, the output given by the PID is displayed in Figure 27.

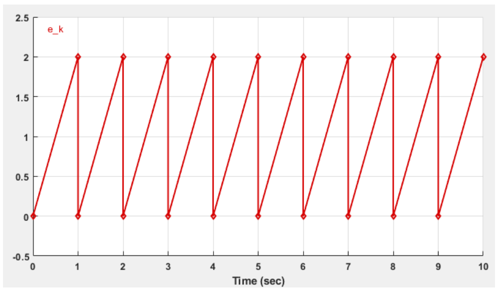


Figure 26 Repetitive Error Input Signal

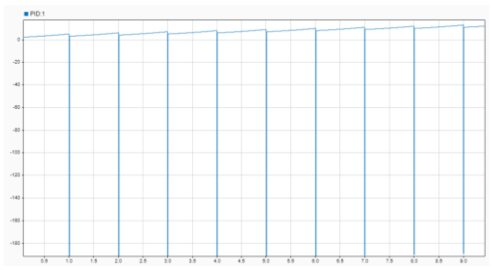


Figure 27 Output for Repetitive Input

2.4.2.3 Constant Error Input Test

A constant error input (refer to Figure 28) is given to the PID controller to test its response, the output given by the PID is displayed in Figure 29.

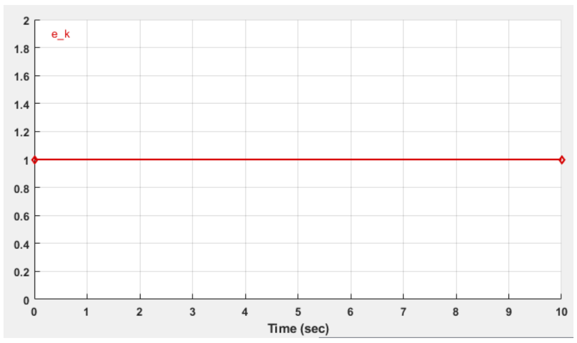


Figure 28 Constant Error Input Signal

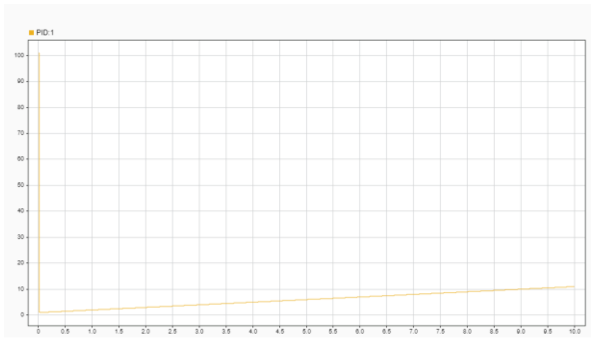


Figure 29 Output for Constant Input

2.4.3 Test Result

All the paths and statements in the model are executed fully, PID responds well to random error input, constant error input, and repetitive error input. It is observed that, whenever the error is not zero, PID controller starts to respond and outputs the signal which will mitigate the error and also satisfying the technical requirements such as rise time, overshoot and steady-state error.

S. No.	Test	Result
1	Coverage	100%
2	Execution	100%
3	MIL	3 Testcases Passed

Table 3 Test Results

2.5 Validation

2.5.1 Validation of Cruise Control Model

S. No.	Requirements	Input	Requirement Criteria	Result	Status
1	Rise time	0-1000 (Step input)	Less than 10 seconds	0.4 seconds	Accepted
2	Overshoot	0-1000 (Step input)	Less than 10%	0.5%	Accepted
3	Stead state error	0-1000 (Step input)	Less than 1%	0%	Accepted

Table 4 Cruise Control Model Validation

2.5.2 Validation of Motor Speed Model

S. No.	Requirements	Input	Requirement Criteria	Result	Status
1	Rise time < 5s	0-100 (Step input)	Less than 5 seconds	0.1 Seconds	Accepted
2	Overshoot < 5%	0-100 (Step input)	Less than 5 %	0 %	Accepted
3	Stead state error <1%	0-100 (Step input)	Less than 1 %	0 %	Accepted

Table 5 Motor Speed Model Validation

3 Advantages and Justification of the SDLC model used

3.1 Advantages

- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed. Since the requirements are very clear for cruise and motor speed control systems V-Shaped model is used.
- Proactive defect tracking – that is defects are found at an early stage.
- Simple and easy to use
- With this development technique while testing if the model is not performing as intended, then it is easy to make a change in the design, and then to start the development process again followed by testing. This iteration helps to mitigate bugs and adapts to varying requirement changes.

3.2 Real World Application

During testing when it is discovered that the model is not getting full coverage or execution, that means there are some unreachable codes, so either we can redesign to cover those paths or we can remove those unreachable code if not needed. These changes can be easily implemented if V-model development cycle is followed.

4 GitHub Workflow

GitHub is integrated with MATLAB, and all the versioning process is done within MATLAB itself. Versioning the project helped to revert back to the previous version/commit when there is major bug is introduced in the current version. It also provided a cloud back for the local repository. In case of local hardware failure all the data will be lost in local repository, we call always checkout the latest commit which pushed to the cloud and can easily resume the work. The link for the repository is as follows:

https://github.com/vigneshbabu0717/coursework_7146CEM_vigneshbabu_11911348.git

5 Conclusion

Cruise control and Motor speed control system is developed with V-development cycle as SDLC. Requirements are gathered from the task given and using the requirements, a design for both systems is created. After the design, actual development is done in Simulink. Testing is done to the controller immediately after its development. Once the PID controller performed as intended, then only the PID controller is integrated to Cruise control and Motor speed control systems. Finally, validated the output of the developed system with requirements to check whether all the requirement criteria are satisfied or not.

6 References

Control Tutorials for MATLAB & Simulink. (n.d.). Retrieved from https://ctms.engin.umich.edu/CTMS/index.php?example=CruiseControl&section=SystemModeling

Control Tutorials for MATLAB & Simulink. (n.d.). Retrieved from https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling