

Coventry University

**7146CEM: Automotive Software Engineering - Design and**

**Development Coursework**

**Student Name:** Prasad Vinodkumar Umashankar

**Student ID**: 11910949

**Email Id:** [**prasadv5@uni.coventry.ac.uk**](mailto:prasadv5@uni.coventry.ac.uk)

**Course: Automotive Software Engineering**

1. **Table of Contents**

1 Table of Contents………………………………………………………………………….…………. 2

2 Terms and Abbreviations………………………………………………………………….….….. 3

3 Overview……….…..…………………………………………………………………………………... 3

3.1 PID Controller……………………………………………………………………………………….…. 3

4 Task Specification…….……………………………………………………………………….….…. 4

4.1 Implementation Process ……………………………………………….…….…….……….….. 4

4.2 PID controller using the following discrete time equation………………………... 5

4.3PID controller using Model Referencing……………………………………………………. 6

4.3.1 PID controller using Model Referencing Cruise Control……………………..…..… 6

4.3.2 PID controller using Model Referencing Motor Speed……………………………… 11

5 Code guidelines……………………………………………………………………………………… 16

6 Version Control……………………………………………………………………………………… 17

**2 Terms and Abbreviations**

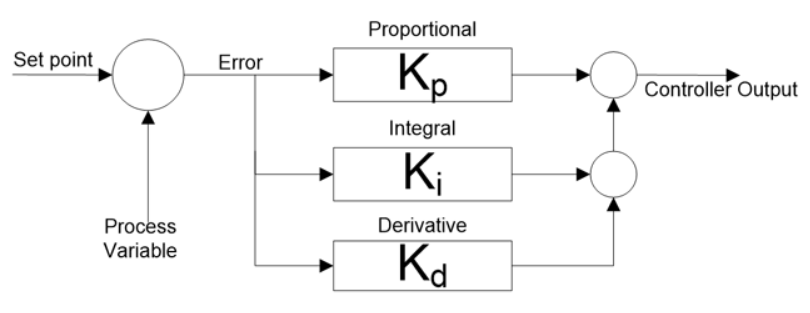
|  |  |
| --- | --- |
| **Terms** | **Definition** |
| PID | Proportional Integral Derivative controller |
| SDLC | Software Development Life Cycles |
| Kp | Proportional gain |
| Ki | Integrator gain |
| Kd | Derivative gain |
| Ts | Time Stamp |

**3 Overview**

**3.1 PID Controller**

A proportional–integral–derivative controller (PID controller or three-term controller) is a control loop mechanism employing feedback that is widely used in industrial control systems and a variety of other applications requiring continuously modulated control.

A PID controller continuously calculates an error value e ( t ) {\display style e(t)} e(t) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms. In below diagram it’s show the PID structure.



**4. Task Specification**

**4.1 Implementation Process**

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

**Design Phase:**

* **Requirement Analysis:** This phase contains detailed communication with the customer to understand their requirements and expectations. This stage is known as Requirement Gathering.
* **System Design:** This phase contains the system design and the complete hardware and communication setup for developing product.
* **Architectural Design:** System design is broken down further into modules taking up different functionalities. The data transfer and communication between the internal modules and with the outside world is clearly understood.
* **Module Design:** In this phase the system breaks into small modules. The detailed design of modules is specified, also known as Low-Level Design.

**Testing Phases:**

* **Unit Testing:** Unit Test Plans are developed during module design phase. These Unit Test Plans are executed to eliminate bugs at code or unit level.
* **Integration testing:** After completion of unit testing Integration testing is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed on the Architecture design phase. This test verifies the communication of modules among themselves.
* **System Testing:** System testing test the complete application with its functionality, inter dependency, and communication. It tests the functional and non-functional requirements of the developed application.
* **User Acceptance Testing:** UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets user’s requirement and system is ready for use in real world.

**4.2 PID controller using the following discrete time equation**

Discrete-time PID Controller Implementation based on the below equation.

Mathematical form y(k) = yp(k) + yi(k) + yd(k)

Where

yp(k) = Kpe(k)

yi(k) = yi(k − 1) + KiTse(k)

yd(k) = Kd/Ts[e(k) − e(k − 1)]

Ts = 0.01

Kp = proportional gain

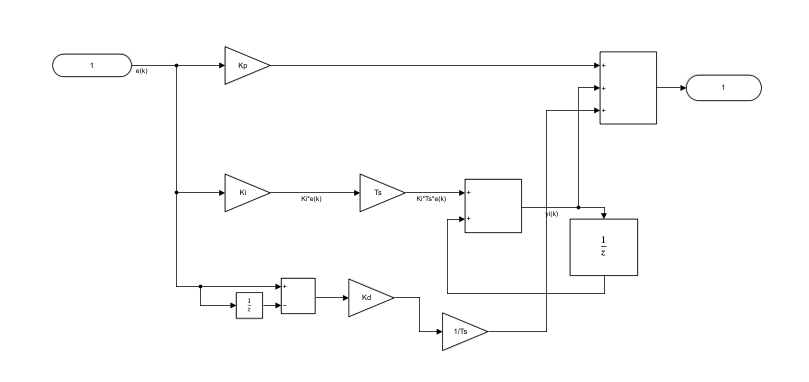
Ki = integrator gain

Kd = derivative gain

Ts = time stamp

PID (Proportional Integral Derivative Controller) is a controller for determining precision instrumentation system with the characteristics of the feedback on the proficiency level system.

Below diagram is representing of Discrete-time PID Controller Implementation



PID (Proportional Integral Derivative Controller) is a controller for determining precision instrumentation system with the characteristics of the feedback on the proficiency level system. PID control component consists of three types: Proportional, Integrative and Derivatives.

**4.3 PID controller using Model Referencing**

**4.3.1 PID controller using Model Referencing Cruise Control**

The model of the cruise control system is relatively simple. If it is assumed that rolling resistance and air drag are proportional to the car's speed, then the problem is reduced to the simple mass and damper system shown below.

Using Newton's 2nd law, the governing equation for this system becomes:

mv = u - bv

where u is the force generated between the road/tire interface and can be controlled directly. For this example, let's assume that

Input values

m = 1000 kg

b = 50 N.sec/m

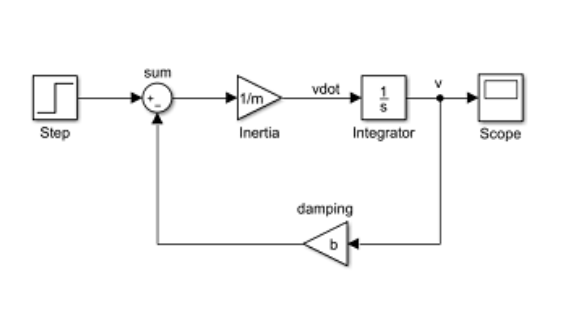
u = 500 N

This system will be modeled by summing the forces acting on the mass and integrating the acceleration to give the velocity. Insert an Integrator block and draw lines to and from its input and output terminals.

Set the block's value to "b" and rename this block to "damping".

Draw a line from the damping Gain block output to the negative input of the Sum Block.

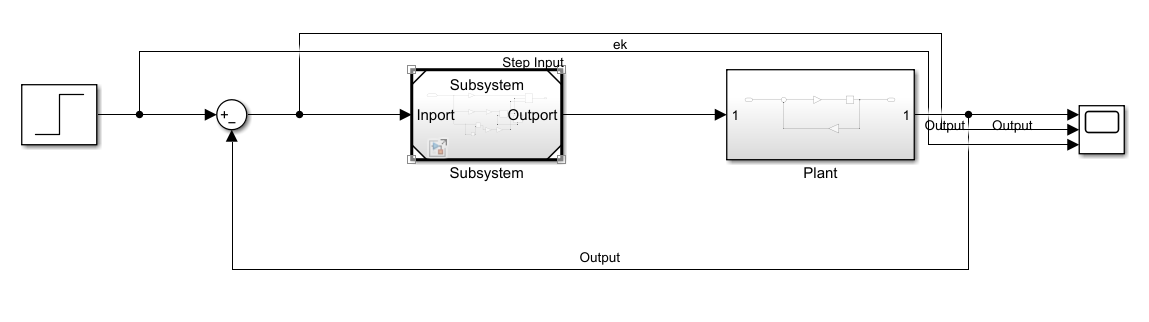
Actual plant module for cruise control as per below diagram.



The second force acting on the mass is the control input, u. We will apply a step input.

* Insert a Step block and connect it with a line to the positive input of the Sum Block.
* To view the output velocity, insert a Scope block connected to the output of the Integrator.

PID controller using Model Referencing Cruise Control. Matlab diagram in below image.



**Case 1:** To observe out put based on Kp, Ki and Kd values**.** Input values for this model as per below.

Kd = 1;

Ki = 13;

Kp = 300;

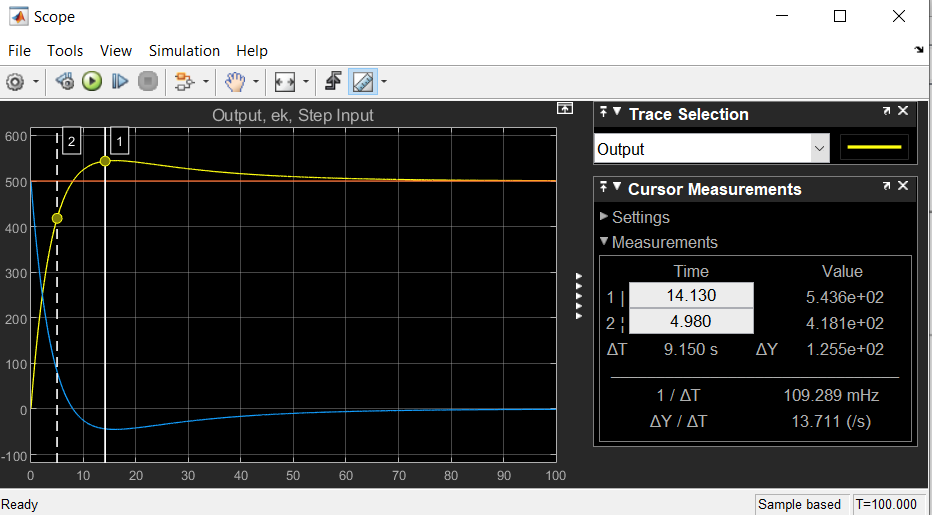
Ts = 0.01;

b = 50;

m = 1000;

u = 500;

output of this Cruise Control model as per below image.



**Case 2:** To observe output based on Kp, Ki and Kd values

Input values for this model as per below.

Kd = 1;

Ki = 20;

Kp = 300;

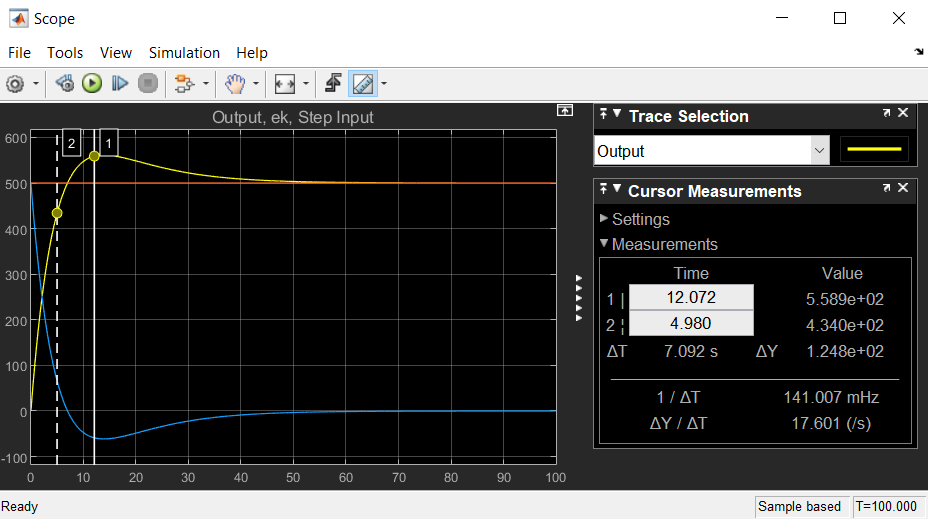
Ts = 0.01;

b = 50;

m = 1000;

u = 500;

output of this Cruise Control model as per below image.



**Case 3:**

Cruise Control:

• Rise time < 10s

• Overshoot < 10%

• Steady-state error < 1%

Input values as per below:

Kd = 1;

Ki = 10;

Kp = 200;

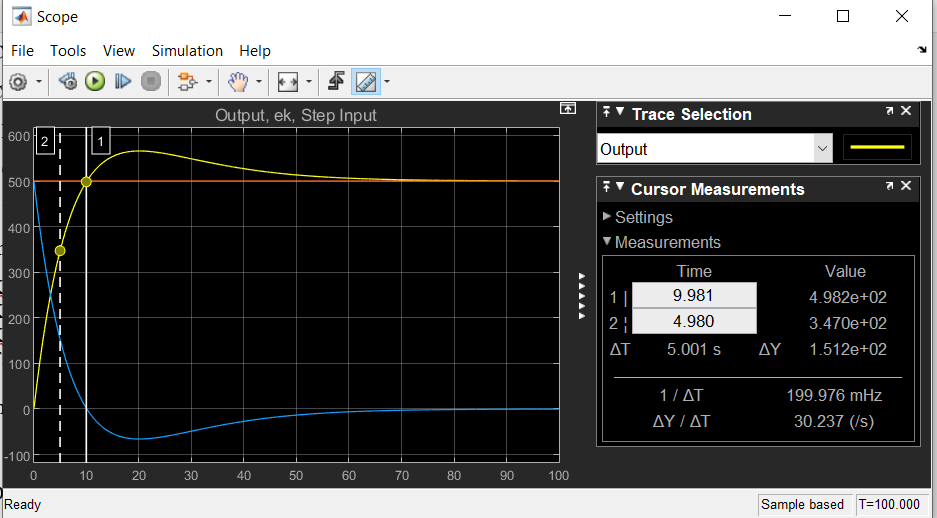
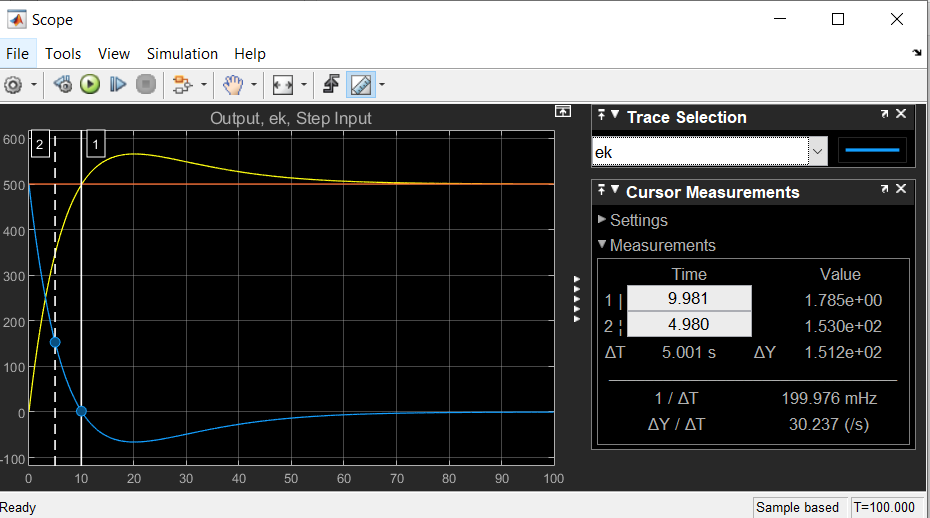
Ts = 0.01;

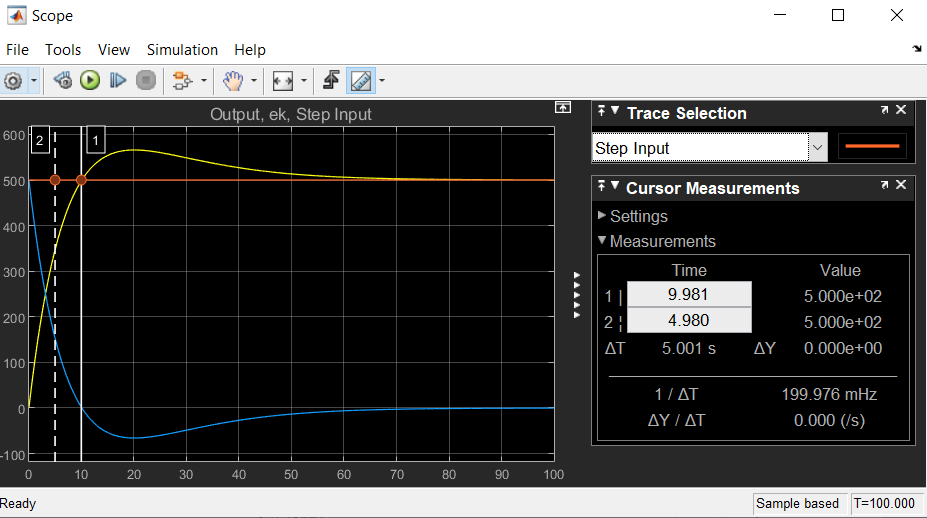
b = 50;

m = 1000;

u = 500;

Output for verify the rise time, overshoot and stedy-state error



**4.3.2 PID controller using Model Referencing Motor Speed**

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.

In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field. In this example 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.

$$  T = K_{t} i$$

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

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

In SI units, 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.

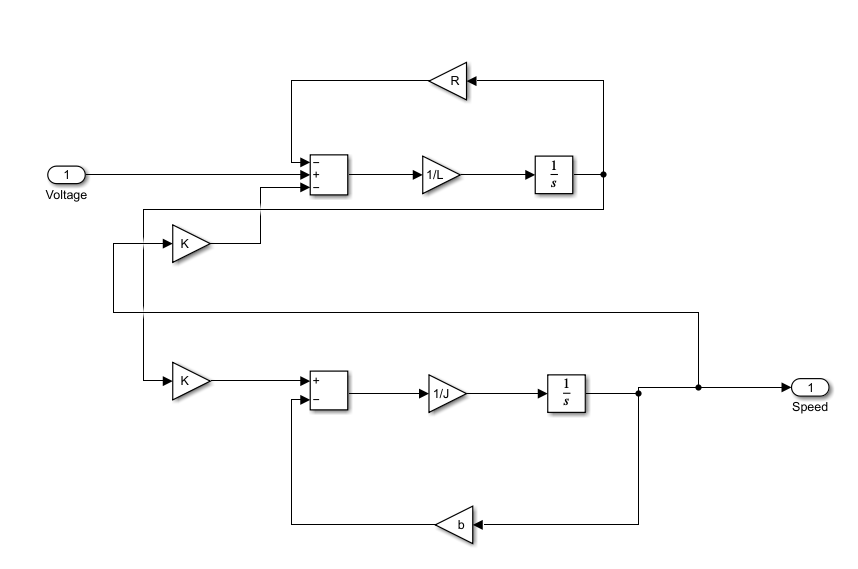
Now, we will add in the voltage terms which are represented in the electrical equation. First, we will add in the voltage drop across the armature resistance.

* Insert a Gain block above the "Inductance" block and flip it from left to right.
* Set the Gain value to "R" and rename this block to "Resistance".
* Tap a line off the current Integrator's output and connect it to the input of the "Resistance" block.
* Draw a line from the "Resistance" block's output to the upper negative input of the current equation Add block.

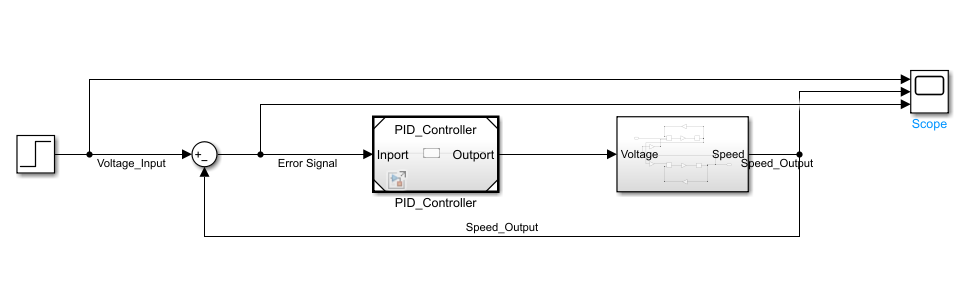
Next, we will add in the back emf from the motor.

* Insert a Gain block attached to the other negative input of the current Add block with a line.
* Edit it's value to "K" to represent the motor back emf constant and Label it "Ke".
* Tap a line off the rotational Integrator's output and connect it to the "Ke" block.
* Add In1 and Out1 blocks from the Simulink/Ports & Subsystems library and respectively label them "Voltage" and "Speed".

The final design should look like the example shown in the figure below.



Final design including PID control and Motor speed control plant model look like the example below.



Case study based on the input values and observation.

**Case 1:** To observe output based on Kp, Ki and Kd values**.** Input values for this model as per below.

J = 0.01;

K = 0.01;

Kd = 1;

Ki = 18;

Kp = 10;

L = 0.5;

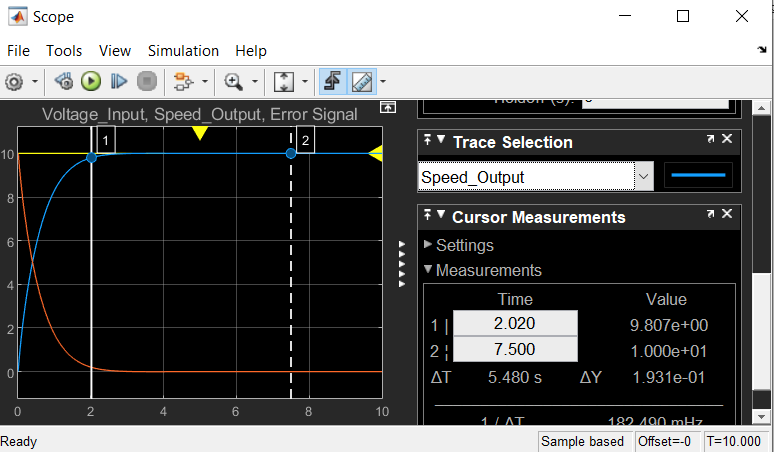
R = 1;

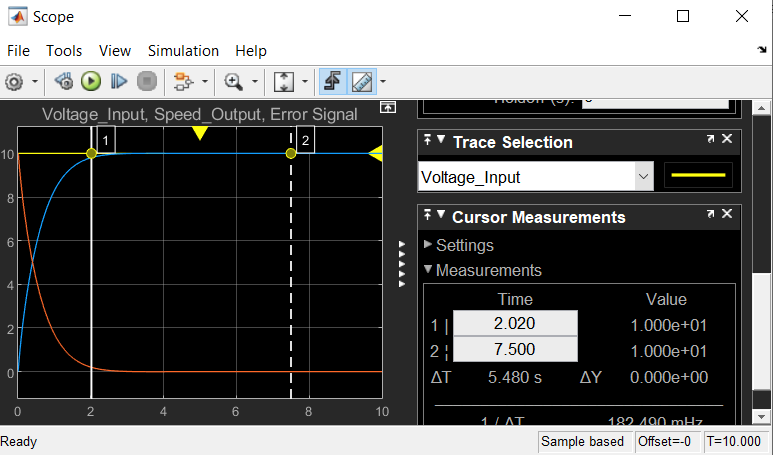
Ts = 0.01;

b = 0.1;

v = 10;

output of this Motor Speed model as per below image.





**Case 2:** To observe output based on Kp, Ki and Kd values

J = 0.01;

K = 0.01;

Kd = 1;

Ki = 18;

Kp = 10;

L = 0.5;

R = 1;

Ts = 0.01;

b = 0.1;

v = 10;

Motor Speed Control:

• Rise time < 5s

• Overshoot < 5%

• Steady-state error < 1%

**Rise Time:**

Rise time is the time taken for a signal to cross a specified lower voltage threshold followed by a specified upper voltage threshold. This is an important parameter in both digital and analog systems.

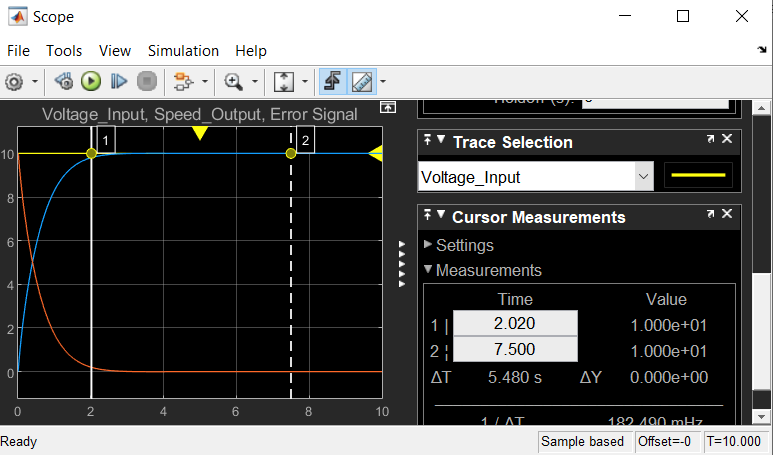
**Overshoot:**

overshoot is the occurrence of a signal or function exceeding its target.

**Steady-state error:**

Steady-state error is defined as the difference between the input (command) and the output of a system in the limit as time goes to infinity. The steady-state error will depend on the type of input as well as the system type.

output of this Motor Speed model as per below image.



**Case 3:** To observe output based on Kp, Ki and Kd values

J = 0.01;

K = 0.01;

Kd = 1;

Ki = 18;

Kp = 10;

L = 0.5;

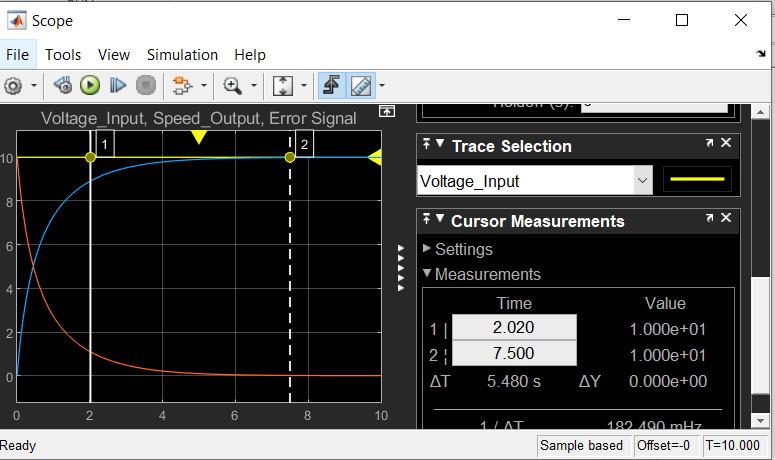
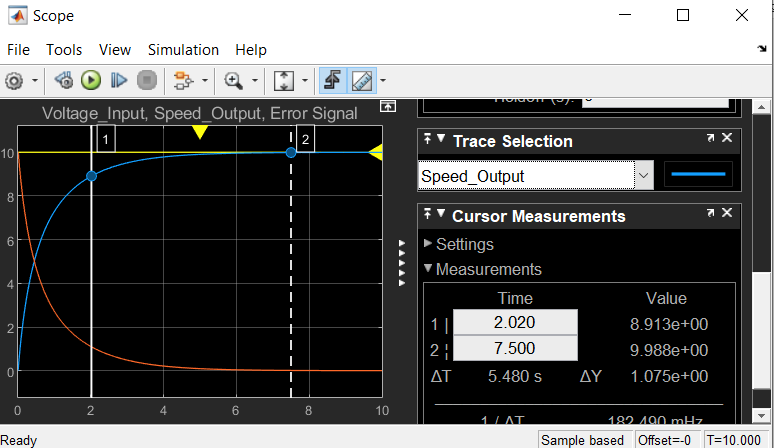
R = 1;

Ts = 0.01;

b = 0.1;

v = 10;

Output as per above Kp, ki and Kd values.

**5. Code guidelines**

The MISRA C:2012 Guidelines classify each guideline as either a rule or a directive.

Compliance with a directive cannot be determined through examination of the source code alone reviews of processes, documentation or functional requirements are needed in order to determine full compliance with directives.

And, it stresses the importance of using a tool that delivers a high degree of success in reporting and distinguishing definite violations as well as possible violations in order to avoid both [false positives and false negatives](https://www.perforce.com/blog/qac/what-are-false-positives-and-false-negatives).

Analysis tools that reliably detect possible violations can help to eliminate uncertainties through the adoption of best-practice coding techniques.

A rule violation does not mean that your code cannot be MISRA compliant. The important thing is that any rule violations are have been properly considered, authorized, and formally documented as a “deviation record”.

A deviation can apply to a single instance of a rule violation, or it can apply to a use case that occurs in multiple places throughout the code.

It is important to be able to identify every instance of the violation associated with a deviation record to support a robust review process.

The document recognizes that there are frequently encountered use cases that require deviations. And, in order to save project effort, “deviation permits” may be used to justify those deviations.

A repository of deviation permits could be agreed by the supplier and the acquirer at the outset of a project in order to save time once the project is underway.

Project-based permission settings can be used to restrict which users are able to add and remove suppressions and deviations. In this way, only those users with the appropriate level of authority can suppress diagnostics and create deviations.

This native support for a MISRA-compliant deviation process is one more reason why Helix QAC has been chosen by hundreds of organizations developing mission-critical systems worldwide.

**6. Version control**

Version control is a system that records changes to a file or set of files over time so that we can recall specific versions later. For the examples in this book, we will use software source code as the files being version controlled, though in reality we can do this with nearly any type of file on a computer. Here we are using Git version control.