VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Oludayo John Oguntoyinbo

PID CONTROL OF BRUSHLESS DC MOTOR AND ROBOT TRAJECTORY PLANNING AND SIMULATION WITH MATLAB®/SIMULINK®

Technology and Communication
2009

ACKNOWLEDGEMENTS

My sincere gratitude goes first to my Creator. Though it has been such a journey academically, He has been my Sufficiency and my Help in times of need. *"He knows me well"*

Mr and Mrs Oguntoyinbo, my parents, deserve all the credit for their support both in kind and in cash; they have been there all through my life voyage prayerfully for me. Their patience is greatly appreciated.

To my supervisor, his view that everything is simple still amazes me even when the "stuffs" are serious nuts to crack. I am grateful for his service.

To my technical friends, Ifeta Adekunle and Frej, your technical supports are worthy of appreciation. You made a part of my story.

To my friends and *personal* friend, you are worthy of my kudos.

Thank you!

Oludayo Oguntoyinbo

December, 2009

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Programme of Information Technology

ABSTRACT

Author	Oludayo John Oguntoyinbo
Title	PID Control of Brushless DC Motor and Robot Trajectory Planning Simulation with MATLAB®/SIMULINK®
Year	2009
Language	English
Pages	90 + 0 Appendices
Name of Supervisor	Liu Yang

This report presents a PID model of a brushless dc (BLDC) motor and a robot trajectory planning and simulation. A short description of the brushless dc motor is given. For this work, mathematical models were developed and subsequently used in getting the simulation parameters. The PID model is accomplished with the use of MATLAB®/SIMULINK®. The operational parameters of the specific BLDC motor were modelled using the tuning methods which are used to develop subsequent simulations. The best PID parameters were thereafter used for the robot trajectory and simulation over a football pitch model.

Keywords PID, BLDC motor, MATLAB/SIMULINK

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND SOME TERMS

BLDC	Brushless Direct Current
PID	Proportional, Integral and Derivative
MATLAB	MATrix LABoratory
M-file	MATLAB text editor file
mdl	Simulink model extension
Nyquist Diagram	
Bode Plot	
Root Locus	
State-space equation	
System response	
Routh-Hurwitz	
Ziegler-Nichols	

1 INTRODUCTION

The use of the general type dc motors has its long history. It has been used in the industries for many years now. They provide simple means and precise way of control [1]. In addition, they have high efficiency and have a high starting torque versus falling speed characteristics which helps high starting torque and helps to prevent sudden load rise [2]. But with such characteristics, the dc motors have some deficiencies that needed to be attended to which gave rise to design of some other alternative types of dc motors. For example, the lack of periodic maintenance, *mechanical wear outs*, acoustic noise, *sparkling*, *brushes* effects are some of the problems that were needed to overcome the defects in dc motors. As a result, emphatic studies have been made on synchronous dc motors with brushless commutators. So, current researches have been tailored towards developing brushless direct current motors, which are fast becoming alternatives to the conventional dc motor types. The ***B***rush***L***ess ***D***irect ***C***urrent (BLDC) motors are gaining grounds in the industries, especially in the areas of appliances production, aeronautics, medicine, consumer and industrial automations and so on.

The BLDC are typically permanent synchronous motors, they are well driven by dc voltage. They have a commutation that is done mainly by electronics application.

Some of the many advantages of a brushless dc motor over the conventional “brushed dc motors are highlighted below [3]:

1. Better speed versus torque characteristics
2. High dynamic response
3. High efficiency
4. Long operating life
5. Noiseless operation
6. Higher speed ranges
7. Low maintenance (in terms of brushes cleaning; which is peculiar to the brushed dc motors).

Another vital advantage is that the ratio of torque delivered to the size of the motor is higher, and this contributes to its usefulness in terms of space and weight consideration.

The BLDC motors come in different phases, for example, single phase, double-, and triple- types. In depth discussion would not be made in this regards, but the most commonly used of all these is the three phase type.

For this purpose, a brief perspective will be considered on how the BLDC motors could be compensated in terms of control and stability. Therefore, this report would presents a theoretical background of DC and BLDC motors, design of simple model of basic DC motors tailored towards developing a BLDC motor model. In addition, a brief introduction of a very essential tool of stability determinant would also be discussed under “PID auto-tuning”. Thereafter, a MATLAB®/SIMULINK® model of the BLDC motor would also be reported accordingly.

The PID controller is applied in various fields of engineering, and it is also a very important tool in telecommunication system. If there is a system and stability is desired, then PID could be very useful.

A simple systematic approach to these tasks is given in chapter format as given below. The chapters 2 and 3 present the “DC motor and design concepts” while chapter 4 gives a brief introduction into the Brushless DC motor and its model concept. It also elaborates the basic concept of their mathematical representations in simple format. The particular BLDC motor used is a maxon motor and chapters 5 – 7 present the whole modelling idea of this specific motor and the open loop response analysis was also included as part of the pre-analysis needed for the subsequent control.

Also, the idea of the PID (Proportional-Integral-Derivative) controller and its design concepts, control mechanism and tuning methods are presented under chapters 8 and 9.

Chapters 10 – 12 present the work done on the robot trajectory planning and simulation. The chapter 10 was used to elaborate the required standard football pitch layout model; chapter 11, for the analysis and computation for the robot four-wheeled motors and the chapter 12 gives the planning stages and corresponding coding schemes.

The results analysis and discussion is presented under the 13th chapter; and finally the chapter fourteen focuses on the conclusion, challenges and recommendation and possible improvement needed in future works.

2 DC MOTOR

2.1 DC motors

A brief illustration and mathematical representation of DC motors will be discussed in the section based on the general concepts of electromagnetic induction.

The DC motors are made of a number of components; some of which are [1]:

1. Frame
2. Shaft
3. Bearings
4. Main field windings (Stator)
5. Armature (Rotor)
6. Commutator
7. Brush Assembly[1]

The most important part of these components that needs detail attention is the main field and the rotating windings (the stator and the rotor respectively).

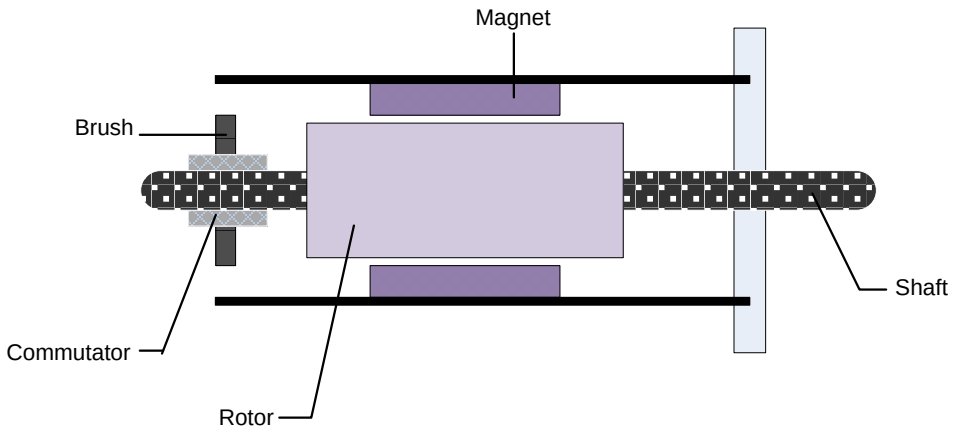

Figure 2.1 – Sectional illustration of a DC motor [2]

As shown in figure 2.1, the stator is formed by the metal carcass with a permanent magnet enclosure which a magnetic field inside the stator windings. At one of the

[1] This is a major difference between the DC and the BLDC motors

ends is the brush mountings and the brush gear which are used for electrical contacts with the armature (the rotor).

The field windings are mounted on the poles pieces to create electromagnetism. The strength of this electromagnetic field is determined by the extent of interaction between the rotor and the stator. Also, the brushes serve as the contact-piece for the commutator to provide electrical voltage to the motor. Consistent dirt on the commutator causes disruption in the supply of dc voltage, which creates a number of maintenance applications. This sometimes could lead to corrosion and sometimes sparks between the carbon made brushes and the commutator.

One of the major challenges is the control of the speed (speed precision); but this could be done by varying the applied voltage. Varying the supply voltage might involve the use of a variable resistor (or a rheostat) which will be connected in tandem with the armature to form a series connection. But this kind of arrangement is not efficient enough as a result of power dissipation. In recent times, solid state electronics has made its implication in this regard through the use of controlled rectifiers and choppers. This arrangement could be efficient as they are used for highly efficient varying dc voltage. In most cases, the most commonly used device is the thyristor (this allows for voltage variation by varying the firing angle of the thyristor in question) [4]. Consider the simple arrangement in figure 2.2.

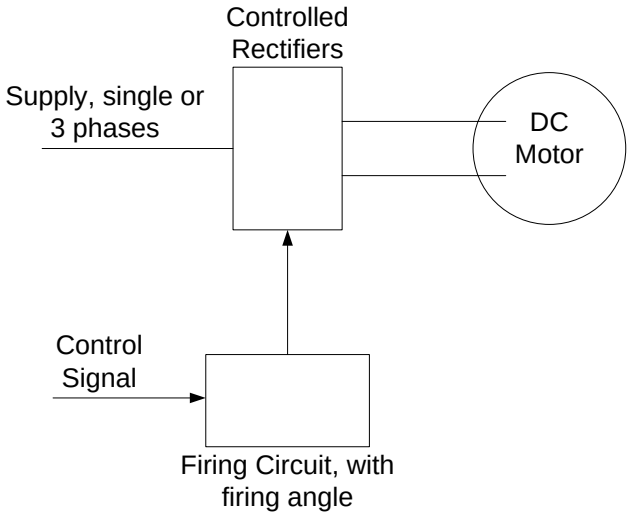

Figure 2.2 – A dc motor operation with a thyristor arrangement using the thyristor firing angle to vary the dc voltage [4].

3 DC MOTOR MODEL

3.1 Mathematical model of a typical DC motor

A typical dc motor equivalent circuit is illustrated as shown in the circuit shown below in figure 3.1 and figure 3.2:

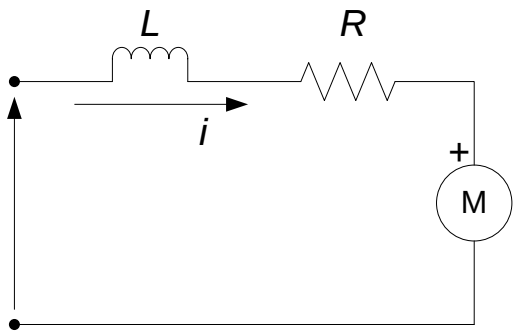

Figure 3.1 – A typical DC motor equivalent electrical circuit.

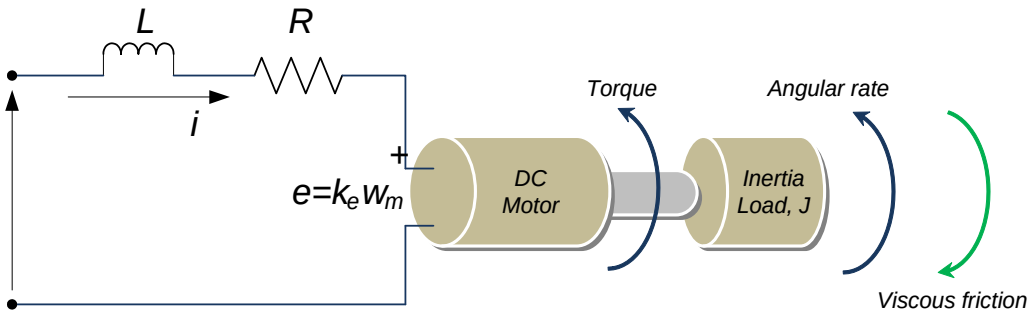

Figure 3.2 – A typical DC motor electromechanical system arrangement.

The basic component represented are the armature resistance, *R* and the armature inductance *L;* in addition, there is the back emf, *e*. From the in figure 3.1 and figure 3.2 above, the following equations are used to describe the relationship of operation.

Using the Kirchhoff's Voltage Law, *KVL*, the following equation 3.1 is obtained:

$$V_s = Ri + L\frac{di}{dt} + e \tag{3.1}$$

At steady state (DC state of zero-frequency), $V_s = Ri + e$.

Therefore, for the non steady-state, equation 3.1 is rearranged to make provision for the back emf, as shown in equation 3.2 below:

$$e = -Ri - L\frac{di}{dt} + V_s \tag{3.2}$$

Where,

$V_s =$ the DC Source voltage

$i =$ the armature current

Similarly, considering the mechanical properties of the dc motor, from the Newton's second law of motion, the mechanical properties relative to the torque of the system arrangement in figure 3.1 and figure 3.2 would be the product of the inertia load, *J* and the rate of angular velocity, ω_m is equal to the sum of all the torques; these follow with equation 3.3 and 3.4 accordingly.

$$J\frac{d\omega_m}{dt} = \sum T_i \tag{3.3}$$

$$T_e = k_f\omega_m + J\frac{d\omega_m}{dt} + T_L \tag{3.4}$$

Where,

$T_e =$ the electrical torque

$k_f =$ the friction constant

$J =$ the rotor inertia

$\omega_m =$ the angular velocity

$T_L =$ the supposed mechanical load[2],

Where the electrical torque and the back emf could be written as:

$$e = k_e\omega_m \text{ and } T_e = k_t\omega_m \tag{3.5}$$

Where,

$k_e =$ the back emf constant

$k_t =$ the torque constant

Therefore, re-writing equations 3.2 and 3.3, the equation 3.6 and 3.7 are obtained,

$$\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{3.6}$$

[2] this could be assumed to be zero for analysis sake

$$\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L \tag{3.7}$$

Using Laplace transform to evaluate the two equations 3.6 and 3.7, the following are obtained appropriately (all initial conditions are assumed to be zero):

For equation 3.6,

$$\mathcal{L}\left\{\frac{di}{dt} = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s\right\} \tag{3.8}$$

This implies,

$$si = -i\frac{R}{L} - \frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{3.9}$$

For equation 3.7,

$$\mathcal{L}\left\{\frac{d\omega_m}{dt} = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L\right\} \tag{3.10}$$

This implies,

$$s\omega_m = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m + \frac{1}{J}T_L \tag{3.11}$$

At no load (for $T_L = 0$); equation 3.11 becomes:

$$s\omega_m = i\frac{k_t}{J} - \frac{k_f}{J}\omega_m \tag{3.12}$$

From equation 3.12, *i* is made the subject for a substitute into equation 3.9.

$$i = \frac{s\omega_m + \frac{k_f}{J}\omega_m}{\frac{k_t}{J}} \tag{3.13}$$

$$\left(\frac{s\omega_m + \frac{k_f}{J}\omega_m}{\frac{k_t}{J}}\right)\left(s + \frac{R}{L}\right) = -\frac{k_e}{L}\omega_m + \frac{1}{L}V_s \tag{3.14}$$

Equation 3.14 becomes:

$$\left\{\left(\frac{s^2J}{k_t} + \frac{sk_f}{k_t} + \frac{sRJ}{k_tL} + \frac{k_fR}{k_tL}\right) + \frac{k_e}{L}\right\}\omega_m = \frac{1}{L}V_s \tag{3.15}$$

And equation 3.15 finally resolved to 3.16:

$$V_s = \left\{\frac{s^2JL + sk_fL + sRJ + k_fR + k_ek_t}{k_t}\right\}\omega_m \tag{3.16}$$

The transfer function is therefore obtained as follows using the ratio of and the angular velocity, ω_m to source voltage, V_s.

That is,

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2JL + sk_fL + sRJ + k_fR + k_ek_t} \tag{3.17}$$

From these, the transfer function could be derived accordingly as follows:

That is,

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2JL + (RJ + k_fL)s + k_fR + k_ek_t} \tag{3.18}$$

Considering the following assumptions:

1. The friction constant is small, that is, k_f tends to 0, this implies that;
2. $RJ \gg k_fL$, and
3. $k_ek_t \gg Rk_f$

And the negligible values zeroed, the transfer function is finally written as;

$$G(s) = \frac{\omega_m}{V_s} = \frac{k_t}{s^2JL + RJs + k_ek_t} \tag{3.19}$$

So by re-arrangement and mathematical manipulation on "*JL*", by multiplying top and bottom of equation 3.19 by:

$$\frac{R}{k_ek_t} \times \frac{1}{R}$$

Equation 3.20 is obtained after the manipulation,

$$G(s) = \frac{\frac{1}{k_e}}{\frac{RJ}{k_ek_t} \cdot \frac{L}{R} \cdot s^2 + \frac{RJ}{k_ek_t} \cdot s + 1} \tag{3.20}$$

From equation 3.13, the following constants are gotten,

The mechanical (time constant),

$$\tau_m = \frac{RJ}{k_ek_t} \tag{3.21}$$

The electrical (time constant),

$$\tau_e = \frac{L}{R} \tag{3.22}$$

Substituting the equations 3.21 and 3.22 into equation 3.20, it yields;

$$G(s) = \frac{\frac{1}{k_e}}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \tag{3.23}$$

4 BRUSHLESS DC MOTOR AND MODEL CONCEPT

One of the major differences between the DC motor and the BLDC is implied from the name. The conventional DC motor has brushes that are attached to its stator while the “brushless” DC motor does not. Also, unlike the normal DC motor, the commutation of the BLDC could be done by electronic control [3]. Under the BLDC motor, the stator windings are energised in sequence for the motor to rotate. More also, there is no physical contact whatsoever between the stator and the rotor. Another vital part of the BLDC is the hall sensor(s); these hall sensors are systematically attached to the rotor and they are used as major sensing device by the Hall Effect sensors embedded into the stator [3]. This works based on the principle of Hall Effect.

The BLDC motor operates in many modes (phases), but the most common is the 3-phase. The 3-phase has better efficiency and gives quite low torque. Though, it has some cost implications, the 3-phase has a very good precision in control [6]. And this is needful in terms of control of the stator current.

4.1 Mathematical model of a typical BLDC motor

Typically, the mathematical model of a Brushless DC motor is not totally different from the conventional DC motor. The major thing addition is the phases involved which affects the overall results of the BLDC model. The phases peculiarly affect the resistive and the inductive of the BLDC arrangement. For example, a simple arrangement with a symmetrical 3-phase and “wye” internal connection could give a brief illustration of the whole phase concept.

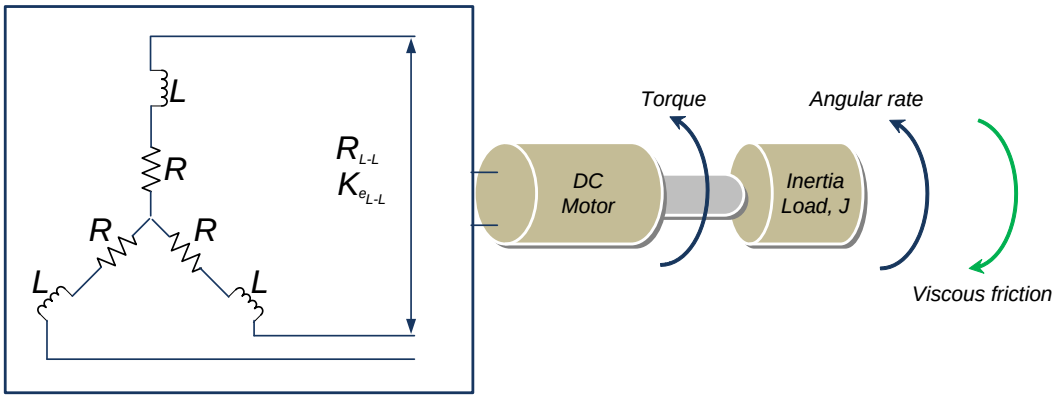

Figure 4.1 – Brushless DC motor schematic diagram

So from the equations 3.20 – 3.22, the difference in the DC and BLDC motors will be shown.

This difference will affect primarily the mechanical and electrical constants as they are very important parts of modelling parameters.

For the mechanical time constant (with symmetrical arrangement), equation 3.21 becomes:

$$\tau_m = \sum \frac{RJ}{K_e K_t} = \frac{J \sum R}{K_e K_t} \tag{4.1}$$

The electrical (time constant),

$$\tau_e = \sum \frac{L}{R} = \frac{L}{\sum R} \tag{4.2}$$

Therefore, since there is a symmetrical arrangement and a three phase, the mechanical (known) and electrical constants become:

Mechanical constant,

$$\tau_m = \frac{J.3R}{K_e K_t} \tag{4.3}$$

Electrical constant,

$$\tau_e = \frac{L}{3.R} \tag{4.4}$$

Considering the phase effects,

$$\tau_m = \frac{3.R_{\emptyset}.J}{\left(K_{e(L-L)}/\sqrt{3}\right).K_t} \tag{4.5}$$

Equation 4.5 now becomes:

$$\tau_m = \frac{3.R_{\emptyset}.J}{K_e.K_t} \tag{4.6}$$

Where K_e is the phase value of the EMF (voltage) constant;

$$K_e = K_{e(L-L)}/\sqrt{3}$$

Also, there is a relationship between K_e and K_t; using the electrical power (left hand side) and mechanical power (right hand side) equations; that is:

$$\sqrt{3} \times E \times I = \frac{2\pi}{60} \times N \times T$$

$$\frac{E}{N} = \frac{T}{I} \times \frac{2\pi \times 1}{60 \times \sqrt{3}}$$

$$K_e = K_t \times \frac{2\pi \times 1}{60 \times \sqrt{3}}$$

$$K_e = K_t \times 0.0605 \tag{4.7}$$

Where,

$K_e = \left[\frac{\mathrm{v-secs}}{\mathrm{rad}}\right]$: the electrical torque

$K_t = \left[\frac{\mathrm{N-m}}{\mathrm{A}}\right]$: the torque constant

Therefore, the equation for the BLDC can now be obtained as follow from equation 3.23 by considering the effects of the constants and the phase accordingly.

$$G(s) = \frac{\frac{1}{K_e}}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \tag{4.8}$$

5 MAXON BLDC MOTOR

5.1 Maxon EC 45 flat Ø45 mm, brushless DC motor

The BLDC motor provided for this thesis is the EC 45 flat Ø45 mm, brushless, 30 Watt from Maxon motors [8]. The order number of the motor is 200142. The parameters used in the modeling are extracted from the datasheet of this motor with corresponding relevant parameters used. Find below in Table 5.1 the major extracted parameters used for the modeling task.

	Maxon Motor Data	**Unit**	**Value**
	Values at nominal voltage		
1	Nominal Voltage	V	12.0
2	No load Speed	rpm	4370
3	No load Current	mA	151
4	Nominal Speed	rpm	2860
5	Nominal Torque (max. continuous torque)	mNm	59.0
6	Nominal Current (max. continuous current)	A	2.14
7	Stall Torque	mNm	255
8	Starting Current	A	10.0
9	Maximum Efficiency	%	77
	Characteristics		
10	***Terminal Resistance phase to phase***	Ω	1.20
11	***Terminal Inductance phase to phase***	mH	0.560
12	***Torque Constant***	mNm/A	25.5
13	***Speed Constant***	rpm/V	37.4
14	Speed/Torque Gradient	rpm/mNm	17.6
15	***Mechanical time constant***	ms	17.1
16	***Rotor Inertia***	gcm^2	92.5
17*	Number of phases		3

Table 5.1 – BLDC motor parameters used [8]

6 BLDC Maxon Motor Mathematical Model

The mathematical model of the BLDC motor is modelled based on the parameters from table 5.1 using the equation 4.23. This is illustrated below:

$$G(s) = \frac{\frac{1}{K_e}}{\tau_m \cdot \tau_e \cdot s^2 + \tau_m \cdot s + 1} \tag{6.1.}$$

So the values for K_e, τ_m and τ_e need to calculated to obtain the motor model.

From equation 4.4,

$$\tau_e = \frac{L}{3.R}$$

$$\tau_e = \frac{0.560 \times 10^{-3}}{3 \times 1.20}$$

$$\boldsymbol{\tau_e} = 155.56 \times 10^{-6} \tag{6.2.}$$

But τ_m is a function of R, J, K_e and K_t,

Where,

R = $R_{\emptyset}$ = 1.2 Ω;

J_{Rotor} = 92.5 gcm^2 = 9.25×10^{-6} Kgm^2;

$K_t = 25.5 \times 10^{-3}$Nm/A

$\tau_m = 0.0171$ secs

From equation 4.6, K_e could be obtained:

That is,

$$\boldsymbol{\tau_m} = \frac{3.R_{\emptyset}.J}{K_e.K_t} = 0.0171$$

$$\boldsymbol{K_e} = \frac{3.R_{\emptyset}.J}{\tau_m.K_t} = \frac{3 \times 1.2 \times 9.25 \times 10^{-6}}{0.0171 \times 25.5 \times 10^{-3}} = 0.0763\ \frac{\text{v} - \text{secs}}{\text{rad}}$$

Therefore, the G(s) becomes:

$$G(s) = \frac{13.11}{155.56 \times 10^{-6} \times 0.0171 \cdot s^2 + 0.0171 \cdot s + 1}$$

$$G(s) = \frac{13.11}{2.66 \times 10^{-6} \cdot s^2 + 0.0171 \cdot s + 1} \qquad (6.3.)$$

The G(s) derived above in the equation 6.3 is the open loop transfer function of the Brushless DC maxon motor using all necessarily sufficient parameters available.

7 OPEN LOOP ANALYSIS OF THE MAXON MOTOR MODEL

The open loop analysis would be done using the MATLAB®/SIMULINK®. And the corresponding stability analysis is given likewise to see the effect thereafter when there is closed loop system incorporation.

7.1 Open Loop Analysis using MATLAB `m-file`

With the aid of the BLDC motor parameters provided, the open loop analysis is done by considering the stability factors and making the necessary plots for this analysis. Some of the plots include the step response, root locus, nyquist diagram, and bode plot diagram.

For this, separate `m-files` were created for the constants, evaluated constants and the main files

`constants.m`

```
%
% Start of code
% Maxon flat motor parameters used in the modeling
%
% Characteristics parameters
R = 1.2;        % Ohms, Terminal Resistance phase to phase
L = 0.560e-3;   % Henrys, Terminal Inductance phase to phase
Kt = 25.5e-3;   % Nm/A, Torque constant
Ks = 37.4       % rpm/V, Speed constant
tm = 17.1e-3;   % seconds, s, Mechanical Time constant
J = 92.5e-7;    % kg.m^2, Rotor inertia, given in gcm^2
p = 3;          % Number of phases
%
% End of code
```

`evaluatedconstants.m`

```
%
% Start of code
%
% Evaluated parameters not given
%
constants
te = L/(p*R);               % seconds, s, Electrical Time constant
Ke = (3*R*J)/(tm*Kt);       % Back emf constant
% End of code
```

topenloop.m

```
%
% Start of code
%
% includes constant parameters
constants

% includes evaluated constants
evaluatedconstants

% Transfer function
G = tf([1/Ke],[tm*te tm 1]);

% Plots the Step Response diagram
figure;
step(G, 0.5);
title('Open Loop Step Response diagram');
xlabel('Time, secs')
ylabel('Voltage, volts')
grid on;

% plots the Root-locus
figure;
rlocus(G);
title('Open Loop Root Locus diagram');
grid on;

% plots the Nyquist diagram
figure;
nyquist(G);
title('Open Loop Nyquist diagram');
grid on;

% plots the Bode Plot
figure;
bode(G);
title('Open Loop Bode plot diagram 1');
grid on;

% plots the Bode Plot
figure;
bode(G,{0.1 , 100})
title('Open Loop Bode plot diagram with wider frequency spacing');
grid on;

% plots the Bode Plot
figure;
GD = c2d(G, 0.5)
bode(G,'r', GD,'b--')
title('Open Loop Bode plot diagram with discretisied response');
grid on;
% End of code
```

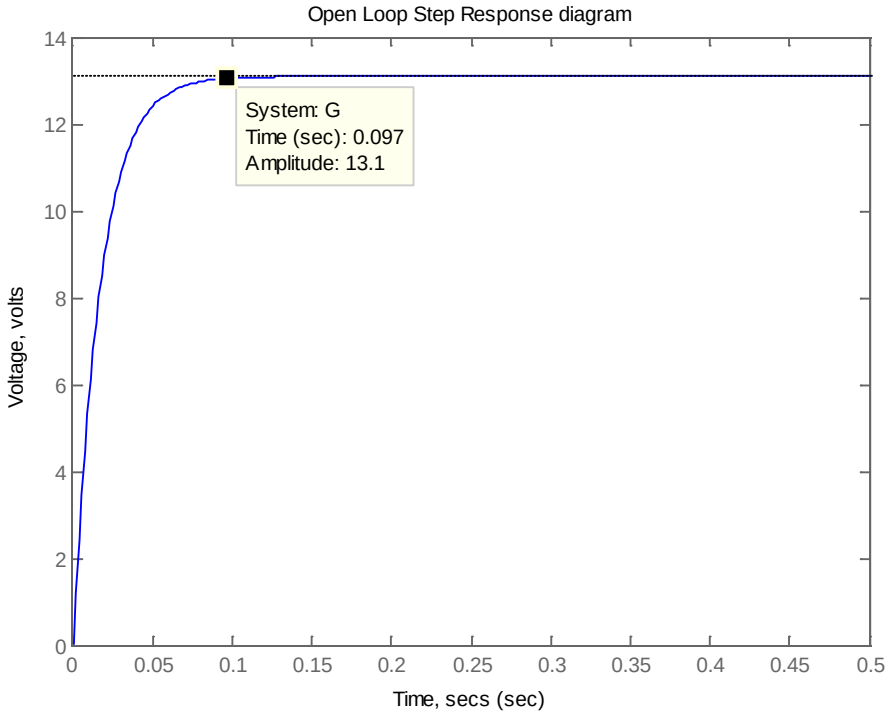

Figure 7.1 – Open Loop Step Response

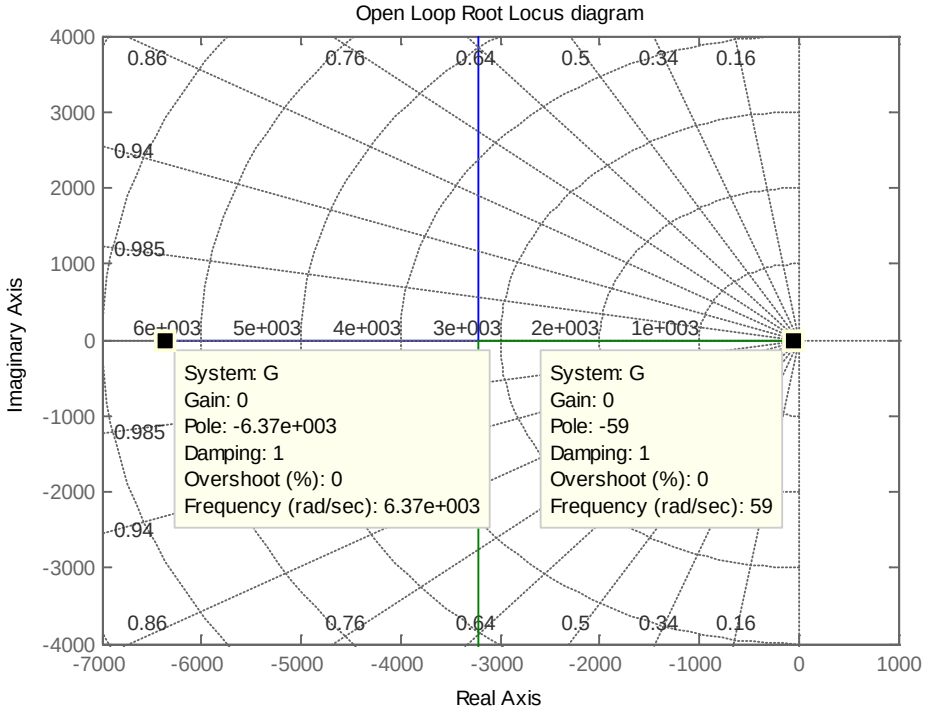

Figure 7.2 – Open Loop Step Root Locus with Gain = 0, Overshoot % = 0 and Damping = 1 for both poles

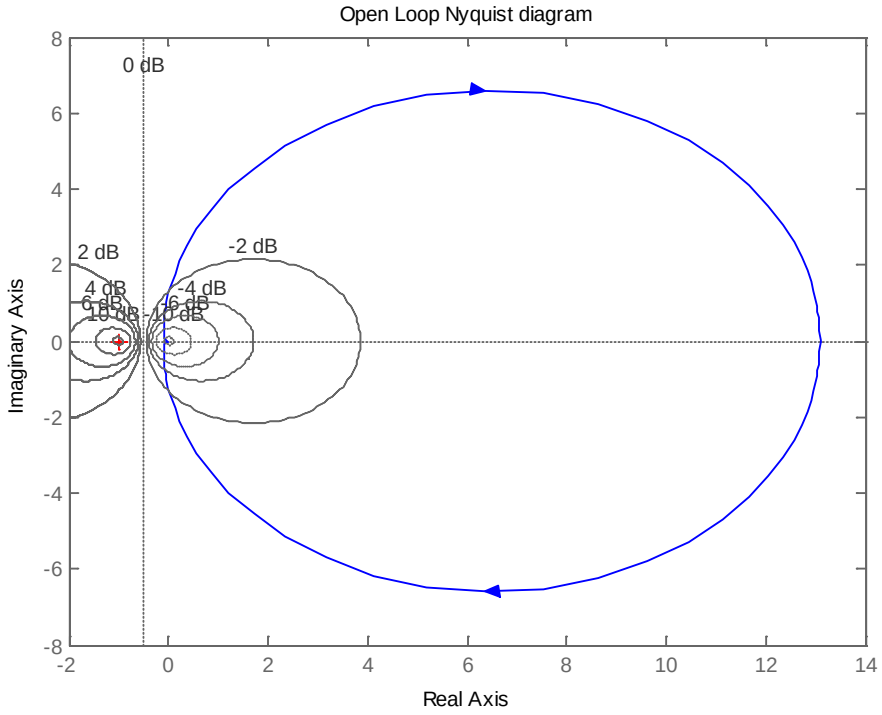

Figure 7.3 – Open Loop Step Nyquist Diagram

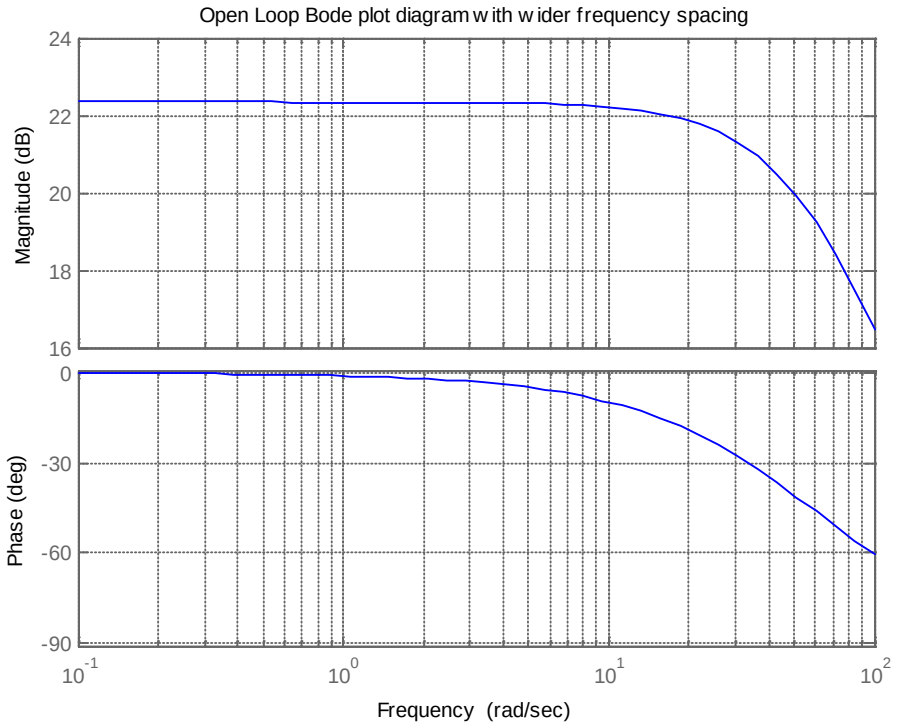

Figure 7.4 – Open Loop Step Bode Plot Diagram

7.2 Open Loop Analysis using SIMULINK

Alternatively, the open loop step response could be done by using the SIMULINK tools as shown in figure 7.5 below.

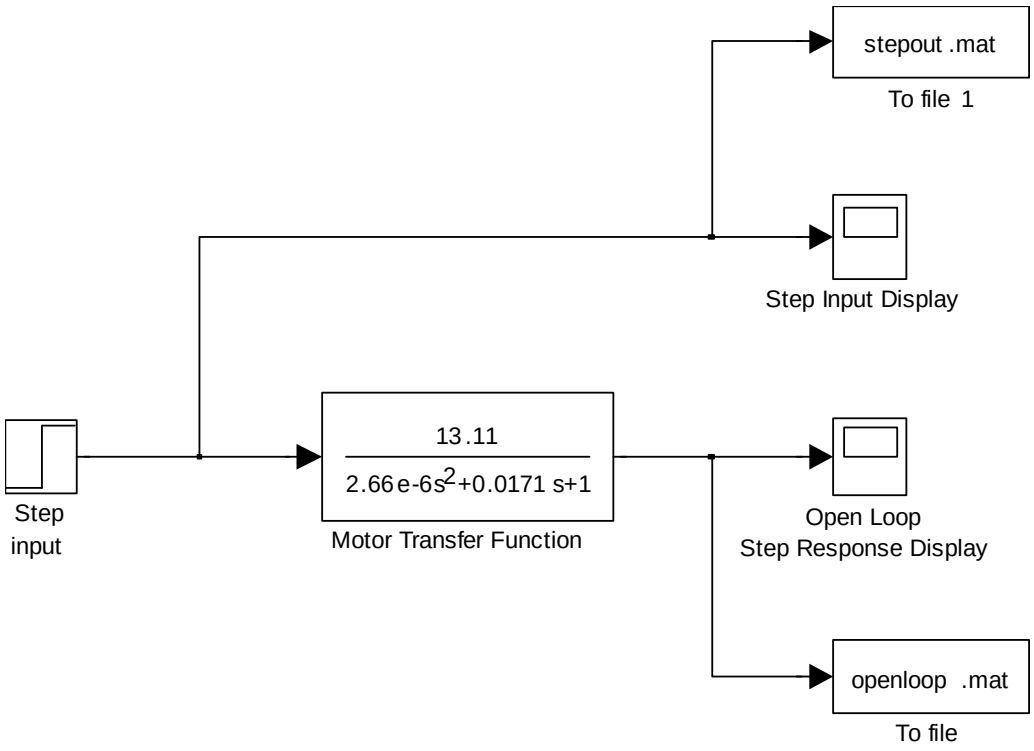

Figure 7.5 – Open loop step response simulink arrangement

From the simulation of figure 7.5 and using a step input of at t=1, the following were obtained.

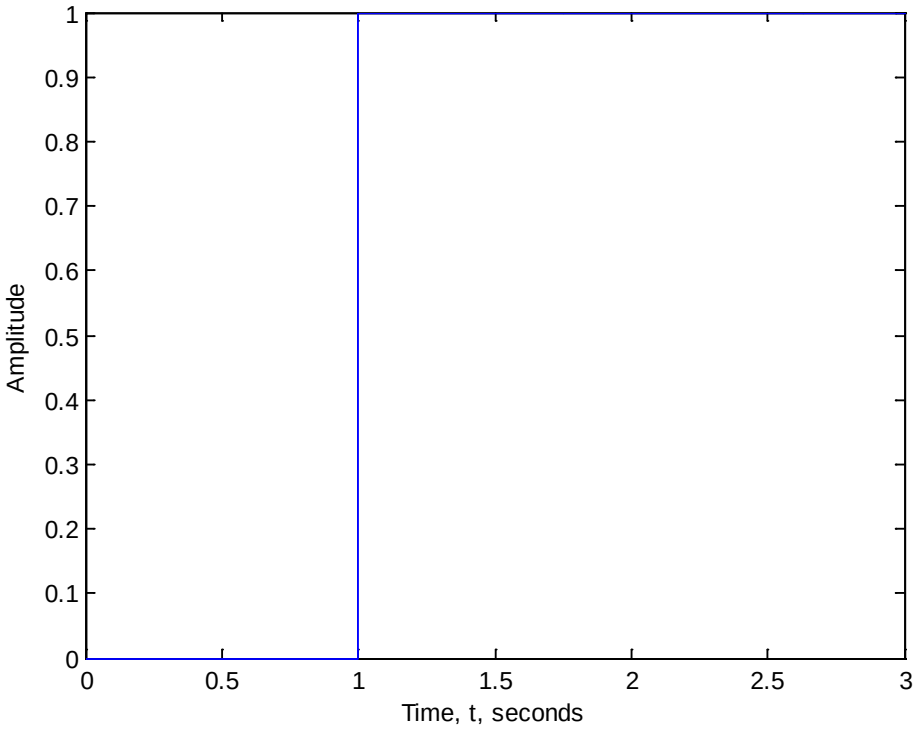

Figure 7.6 – Step input for the open loop simulink arrangement (at t=1)

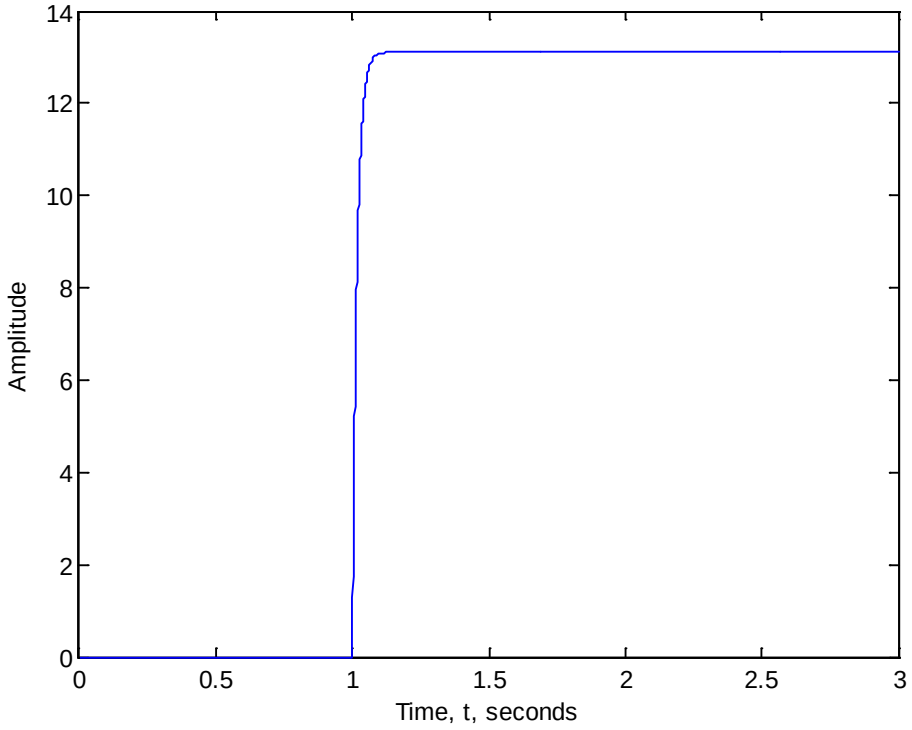

Figure 7.7 – Open loop step response output for the simulink arrangement

With the step response moved to 0.05 for a better display, a joint output of the step input and open loop step response was simulated to give figure 7.8 below. This shows the effect of the system model on the step input.

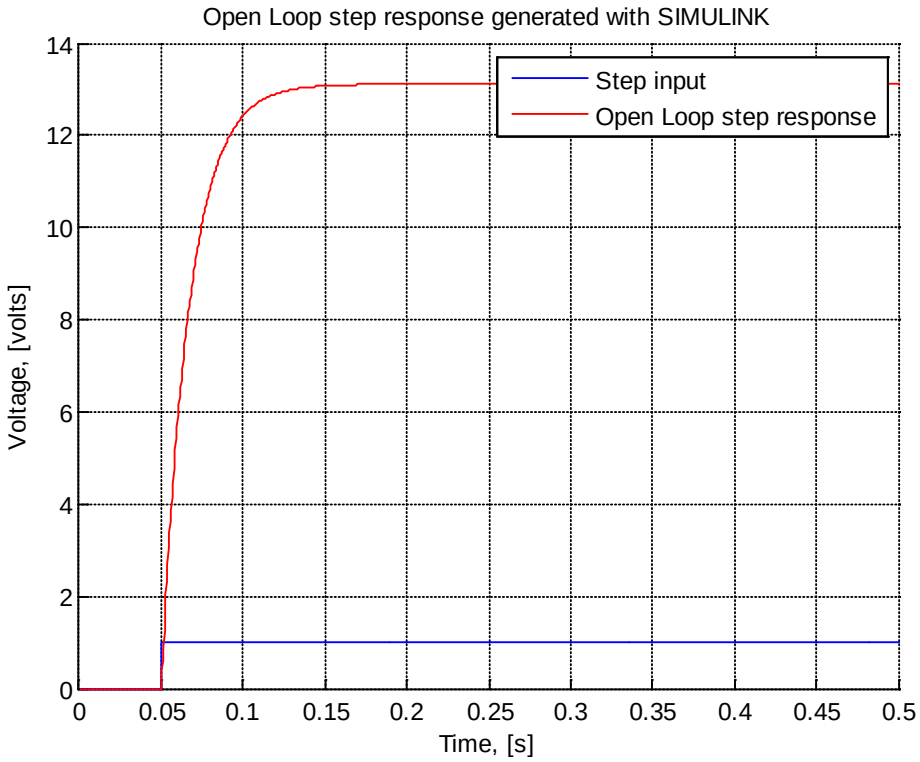

Figure 7.8 – Combined step input and open loop step response span over t=0.5 s

8 PID DESIGN CONCEPT

The Proportional-Integral-Derivative (PID) controller is about the most common and useful algorithm in control systems engineering [7]. In most cases, feedback loops are controlled using the PID algorithm. The main reason why feedback is very important in systems is to be able to attain a set-point irrespective of disturbances or any variation in characteristics of any form.

The PID controller is always designed to correct error(s) between measured process value(s) and a particular desired set-point in a system.

A simple illustration on how the PID works is given below:

Consider the characteristics parameters – proportional (P), integral (I), and derivative (D) controls, as applied to the diagram below in figure 8.1, the system, S is to be controlled using the controller, C; where controller, C efficiency depends on the P, I and D parameters [8].

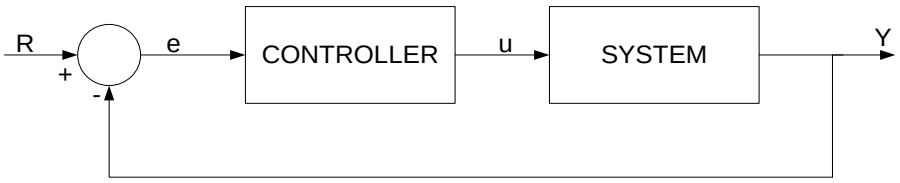

Figure 8.1 – A typical system with a controller [8]

The controller provides the excitation needed by the system and it is designed to control the overall behaviour of the system.

The PID controller has several categories of structural arrangements. The most common of these are the series and parallel structures and in some cases, there are the hybrid form of the series and the parallel structures.

The following shows the typical illustrative diagrams of common PID controller structures.

Typically, the function of the form shown in equation 8.1 is applicable in this kind of PID controller design.

$$K_P + \frac{K_I}{s} + K_D \cdot s = \frac{K_D s^2 + K_P s + K_I}{s} \qquad (8.1)$$

[8].

Where,

K_P = Proportional gain

K_I = Integral gain

K_D = Derivative gain

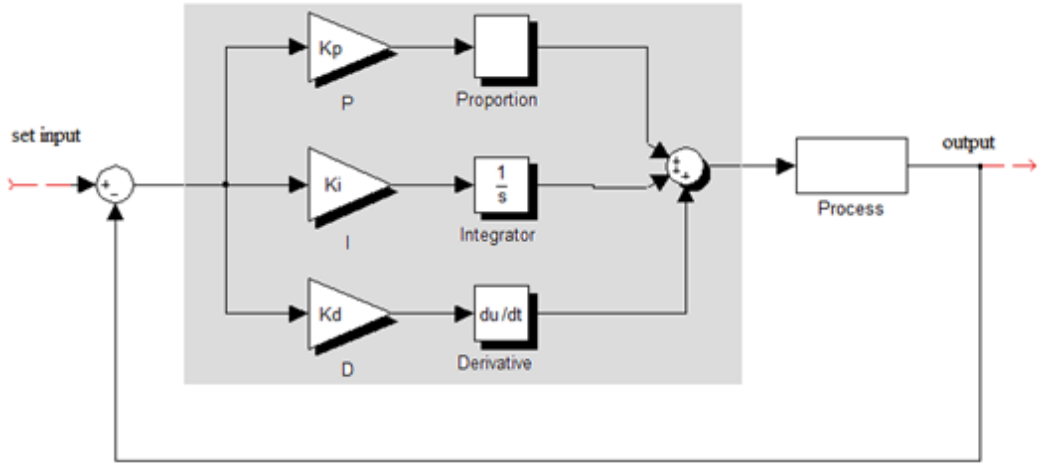

Figure 8.2 – PID parameters schematics

Considering the figure 8.1, variable, *e* is the sample error, and it is the difference between the desired input value, *R* and the actual output, *Y*. In a closed loop, *e* will be sent to the controller, and the controller will perform the integral and derivative computation on the error signal. Thereafter, the signal, *u* which is the output of the controller is now equal to the sum of [the product of proportional gain, K_P and the magnitude of the error], [the product of the integral gain, K_I and the integral of the error] and [the product of the derivative gain, K_D and the derivative of the error]. That is,

$$u = K_P e + K_I \int e dt + K_D \frac{de}{dt} \qquad (8.2)$$

The signal value, *u* is sent continuously to the plant with every corresponding new output, *Y* being obtained as the process continues. The output, *Y* is sent back and subsequently new error signal, *e* is found and the same process repeats itself on and on.

Also, it is very typical to have the PID transfer function written in several forms depending on the arrangement structure. The following equation shows one of these (a parallel structure):

$$K_P + \frac{K_I}{s} + K_D \cdot s = K_P \times \left(1 + \frac{1}{T_I \cdot s} + T_D \cdot s\right) \tag{8.3}$$

Where,

$K_P = \text{Proportional gain}$

$T_I = \text{Integral time or Reset time} = \frac{K_P}{K_I}$

$T_D = \text{Derivative time or Rate time}$

8.1 Some characteristics effects of PID controller parameters

The proportional gain K_P, will reduce the rise time and might reduce or remove the steady-state error of the system. The integral gain K_I, will eliminate the steady-state error but it might a negative effect on the transient response (a worse response might be produced in this case). And the derivative gain K_D, will tend to increase the stability of the system, reducing overshoot percentage, and improving the transient response of the system. In all, the table below will give comprehensive effects of each of the controllers on a typical closed-loop system.

Parameter	Rise time	Overshoot	Settling time	Steady-state error
K_P	↓	↑	small change	↓
K_I	↓	↑	↑	eliminate
K_D	small change	↓	↓	small change

Legend	↓	*Decrease*
	↑	*Increase*

Table 8.1 – PID controller parameter characteristics on a typical system [8]

The ability to blend these three parameters will make a very efficient and stable system. It should be noted that the relationship between the three controller parameters may not exactly be accurate because of their interdependency. Therefore, it is very possible to compute particular parameters which effects would be noticed on the other two.

8.2 PID controller design tips

Designing a PID controller might require some of the following steps to obtain a more efficient and stable system [5]:

1. It is advisable to obtain the open-loop response of the system first and subsequently determine what to improve;
2. Add a proportional gain control to improve the rising time;
3. Then, add a derivative gain to improve the overshoot percentage;
4. And perhaps, add the integral control to eliminate the steady-state error;
5. Thereafter, adjust each of the parameters might be important to achieve an overall desired performance (or output).

And most importantly, all the three PID controller parameters might not be necessarily used in some cases. In most cases, the tuning stops at the PI – control combination.

More also, it should be noted that the major goal of the PID parameters is to obtain a fast rise time with minimum overshoot and no (*almost no*) steady-state error.

9 PID CONTROLLER TUNING PARAMETERS

Under this section a critical analysis would be done on the PID tuning criteria and the parameters involved. Before a detail analysis is done, a quick look at the tuning methods is considered first and thereafter, specific tuning parameters are computed for the BLDC maxon motor. Some of the generally used tuning methods are the Trial and Error method, the Ziegler-Nichols method (1^{st}), *Improved* Ziegler-Nichols method (2^{nd}), Cohen-Coon method, Genetic Algorithms and so on. For this work, the Ziegler-Nichols tuning method would be given a priority.

9.1 The PID arrangement

As a general form, a full schematic of the PID controller arrangement with the System model arrangement is displayed in figure 9.1 as a start for the tuning procedure.

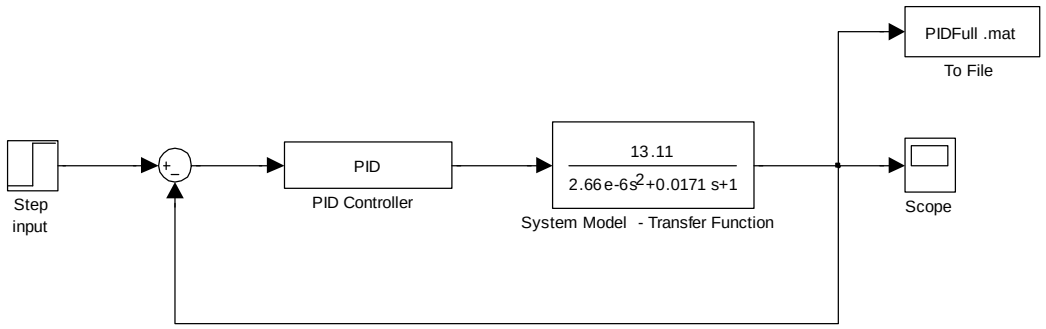

Figure 9.1 – PID Schematic for a full PID Controller with System model arrangement

The figure 9.1 is under no saturation, but the saturation is included in figure 9.2. Both figures would be used for our analysis.

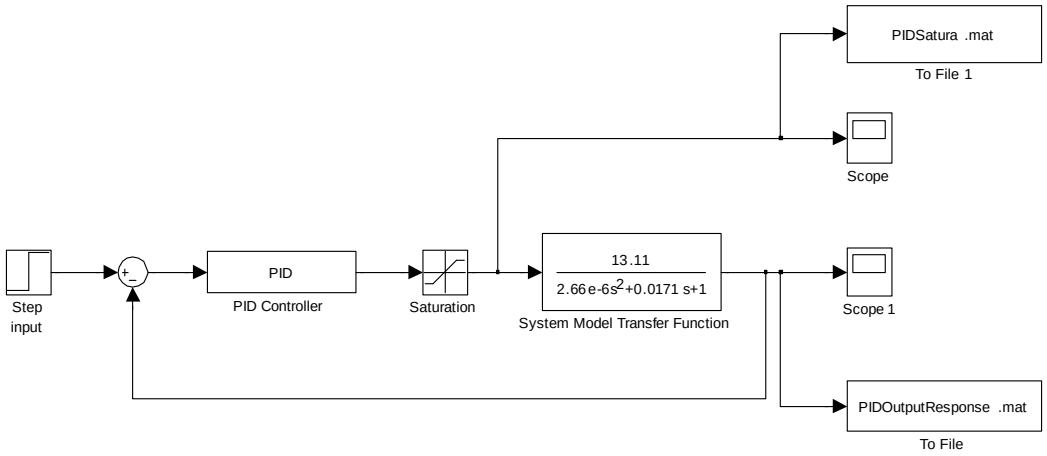

Figure 9.2 – PID Schematic for a full PID Controller (with saturation) and system model arrangement

For an initial computation, P, PI and PID would be considered in that order to observe the best part for the PID parameters to be obtained.

9.2 Trial and Error tuning methods

This method is crude but could help in getting an overview of what the PID parameters could be like and their effects on the whole system model. It is particularly time consuming because of its trial and format. But a computational stability rule was needed to set a mark for the trial and effect. This is done by using the Routh-Hurwitz stability rule as shown below. Under this, emphasis would be mainly on the PID combination.

9.2.1 The Routh-Hurwitz stability rule

From the various designs needed for this trial, a brief stability check is needed to make the trial and error at the first instance. It would be observed that the only design near the perfect (open-loop – which is without compensation or controller) is the PID. To have a more appropriate trial and error value, the following steps would be followed for only the PID structure.

From the PID controller equation 9.1,

$$K_P + \frac{K_I}{s} + K_D \cdot s = K_P \times \left(1 + \frac{1}{T_I \cdot s} + T_D \cdot s\right) \quad (9.1)$$

Similarly,

$$K_P + \frac{K_I}{s} + K_D \cdot s = \frac{K_P \cdot s + K_I + K_D.s^2}{s} \quad (9.2)$$

This is used in the m-file `tclosedloopPID_TrialError4.m` and it is convuled with the motor model.

Keeping the K_P part, with T_I and T_D set to infinity and zero respectively. A controller gain, K_C could be obtained that would sustain the oscillation output. This value serves as the ultimate gain, K_{CU}. For a proper oscillation, K_C is set to be less than K_{CU}.

Assumed the figure 9.9 below with a gain of K_{CU} and the system model:

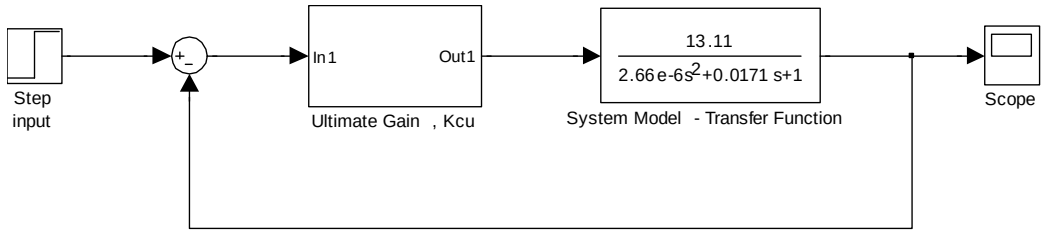

Figure 9.3 – Trial and Error PID computation diagram

By obtaining the characteristics equation of the figure 9.9, a limiting gain could be obtained just before sustained oscillation and this is assumed as the K_{CU}.

tclosedloopPID_TrialError4.m

```
% Start of code
clear
close all
% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants

num = [1/Ke];
den = [tm*te tm 1];

%Ziegler-Nichols parameter computed
Kp = 13.11;     %Proportional gain
Ki = 0%1310.6;          %Integral gain
Kd = 0%0.0763;          %Derivative gain
% For the PID equation
numc = [Kd Kp Ki];
denc = [1 0];

% convule "num with numc" and "den with demc"
numa = conv(num, numc);
dena = conv(den, denc);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac, denac] = cloop(numa, dena);
% Plotting the new step-response
t = 0:0.00001:0.5;
step(numac, denac, t);      % across 0.01 seconds timing
title('Closed loop step response for ZN - Kp, Ki and Kd');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
%grid on;

% New G1 for overall closed loop trasnfer function
G1 = tf(numac, denac);
% End of code
```

Therefore, we have:

$$1 + K_{CU} \cdot G(s) = 0 \tag{9.3}$$

$$1 + K_{CU} \cdot \frac{13.11}{2.66 \times 10^{-6} \cdot s^2 + 0.0171 \cdot s + 1} = 0 \tag{9.4}$$

Equation 9.4 becomes,

$$2.66 \times 10^{-6} \cdot s^2 + 0.0171 \cdot s + 1 + 13.11 \cdot K_{CU} = 0 \qquad (9.5)$$

So for stability purposes, K_{CU}'s range of values could be obtained by using the Routh-Hurwitz condition of stability. This is computed below:

s^2	2.66×10^{-6}	$1 + 13.11 \cdot K_{CU}$
s^1	0.0171	0
s^0	$\mathbf{1 + 13.11 \cdot K_{CU}}$	$-$

According to Routh-Hurwitz condition, the obtained characteristics equation 9.5 should be spread into column as shown above and the s^0 is evaluated as follows (because it has the assumed unknown K_{CU} which would be evaluated):

$$\mathrm{s}^0(\text{1st row}) = -\frac{\begin{vmatrix} 2.66 \times 10^{-6} & 1 + 13.11 \cdot K_{CU} \\ 0.0171 & 0 \end{vmatrix}}{1 + 13.11 \cdot K_{CU}}$$

$$\begin{aligned} &\mathrm{s}^0(\text{1st row}) \\ &= -\frac{(2.66 \times 10^{-6} \times 0) - (1 + 13.11 \cdot K_{CU})(0.0171)}{0.0171} \\ &= 1 + 13.11 \cdot K_{CU} \end{aligned}$$

For stability sake, the 1st column after the s-column must not have any sign change (that is, no change from + to – or – to +). Therefore, $\mathrm{s}^0(\text{1st row})$, must be greater than zero.

This implied that,

$$1 + 13.11 \cdot K_{CU} > 0$$

Then,

$$13.11 \cdot K_{CU} > -1$$

$$K_{CU} > \frac{-1}{13.11} = -0.0763$$

This implies that K_{CU} has its main value in the positive range. With a rough trial and error tuning, K_P, can be fixed to full value of the system model numerator, which is 13.11. The K_I and K_D were set initially to zero to see the effect of the K_P on the system. This resulted into the figure K_I about the inverse of 0.0763 = 13.106, and K_D = 0.0763. After this,

9.2.2 Proportional control

Based on the M-file – "`tclosedloopP.m`", the following figure 9.3, figure 9.4, figure 9.5 and figure 9.6 were obtained as an improvement to the open-loop system. By making an initial raw guess of the value of K_P just before applying the Routh-Hurwitz condition.

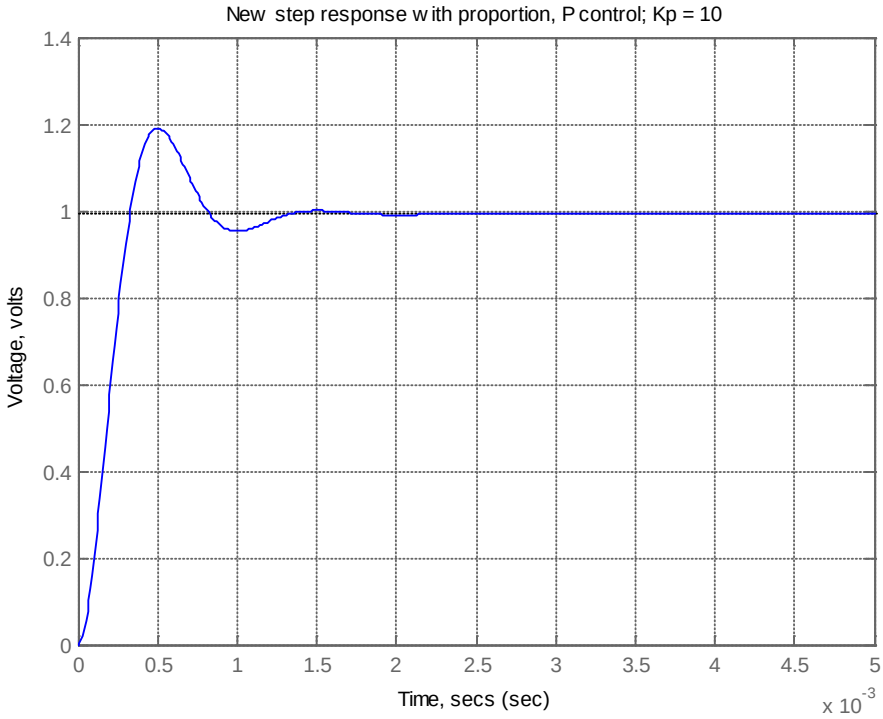

Figure 9.4 – Proportional controller gain effect on the system

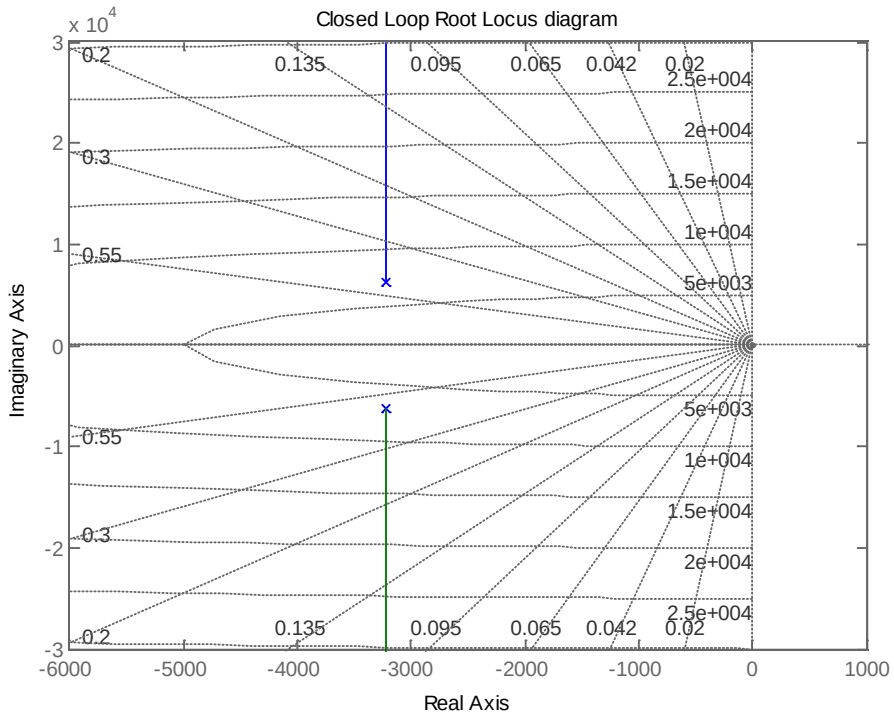

Figure 9.5 – Root locus diagram for the proportional controller gain effect

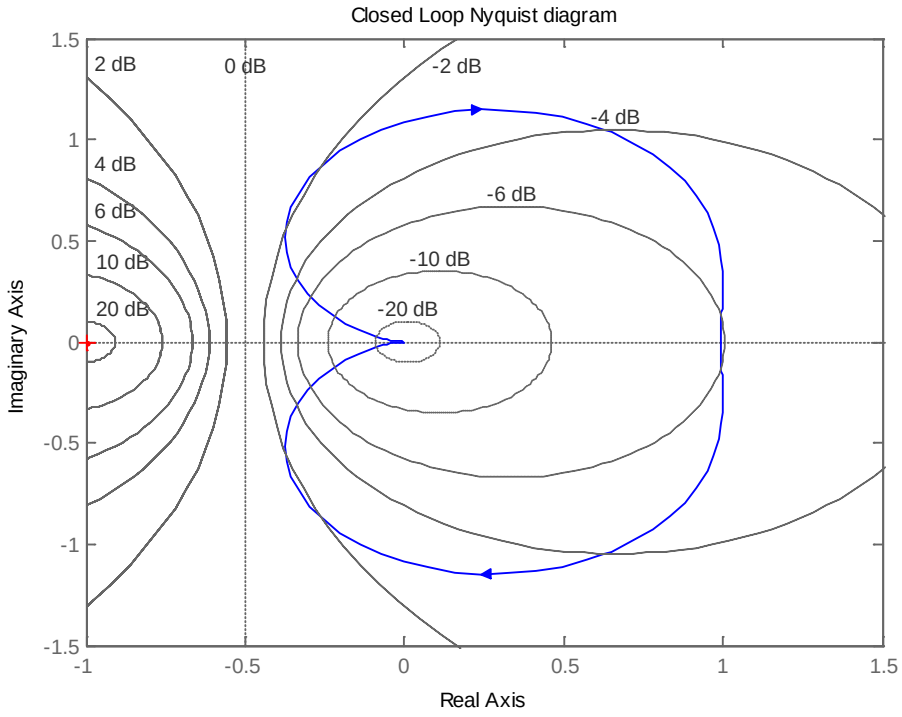

Figure 9.6 – Nyquist diagram for the proportional controller gain effect

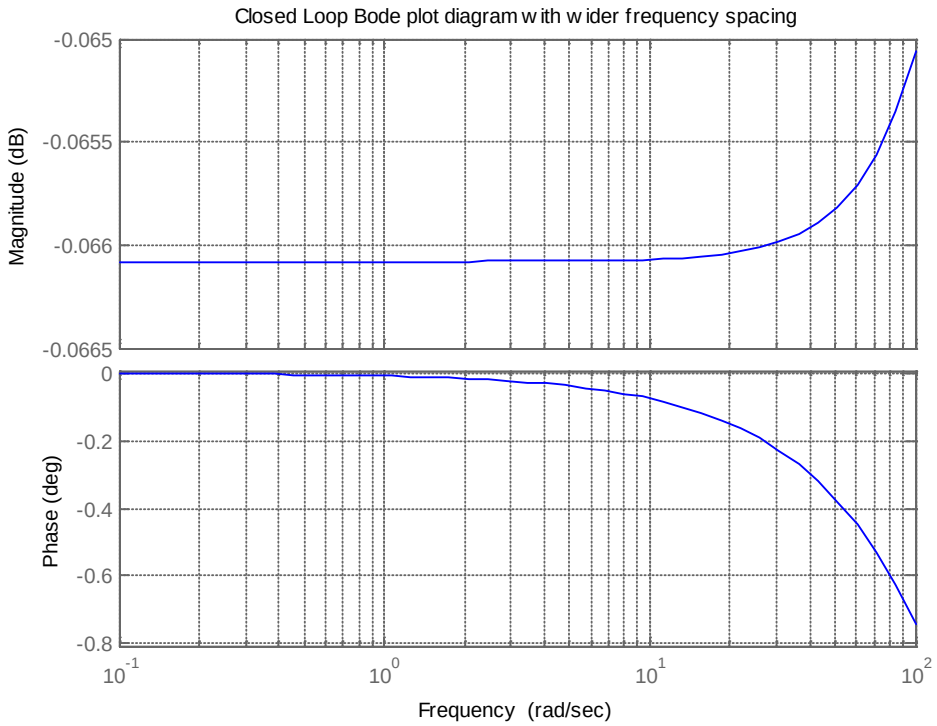

Figure 9.7 – Bode plot for the proportional controller gain effect

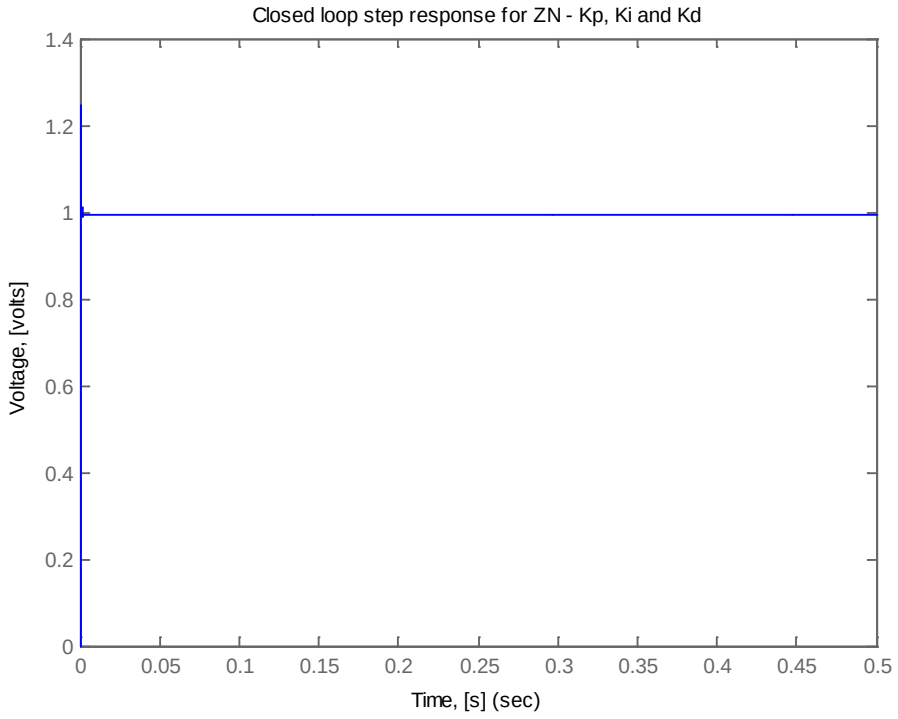

Figure 9.8 – Trial and error value used for the P parameters output, with K_I and K_D set to zero

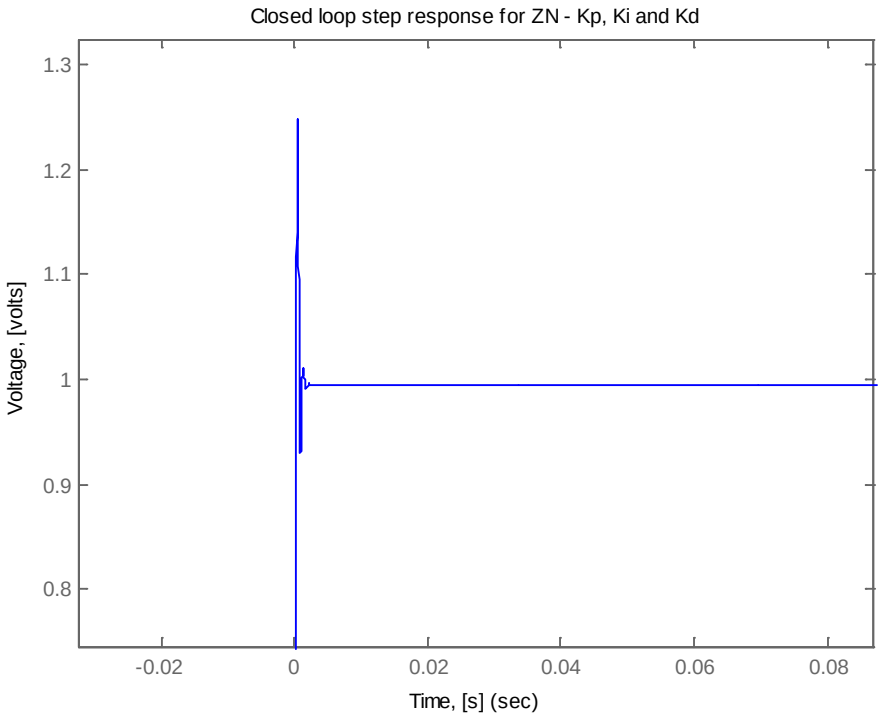

Figure 9.9 – Trial and error value used for the P parameters output, with K_I and K_D set to zero (zoomed display)

The above figure 9.3 – 9.7 show how the proportional controller has reduced the rising time and the steady-state error, the overshoot is reasonably increased but the settling time is also decreased slightly. The subsequent figures show the effects of the trial and error method of tuning applied. The detail analysis would be under the results and analysis section.

9.2.3 Proportional-Integral control

To improve on effect of the K_P, an additional K_I was also set based on the Routh-Hurwitz condition used above. This is implemented with the same m-file – "`tclosedloopPID_TrialError4.m`", the following figures 9.10 – 9.11 was obtained as an added improvement. To make a more visible on the step response, the integral parameter was scaled by 1000 to see its effects, that is, K_I= 1310.6. And another "supposed" improvement was also obtained (figures 9.12 – 9.13).

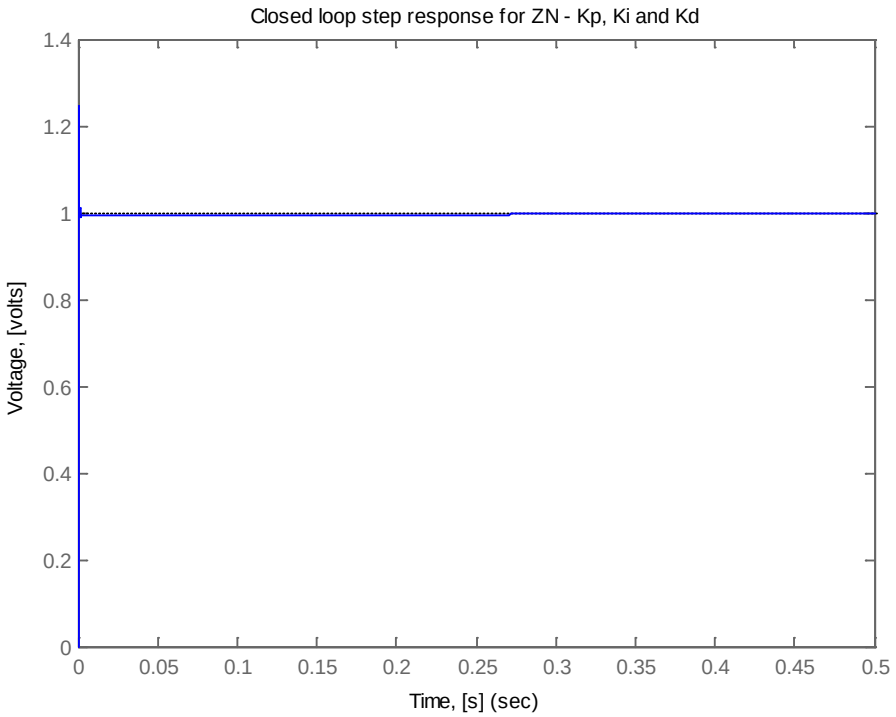

Figure 9.10 – Trial and error values used for the PI parameters output

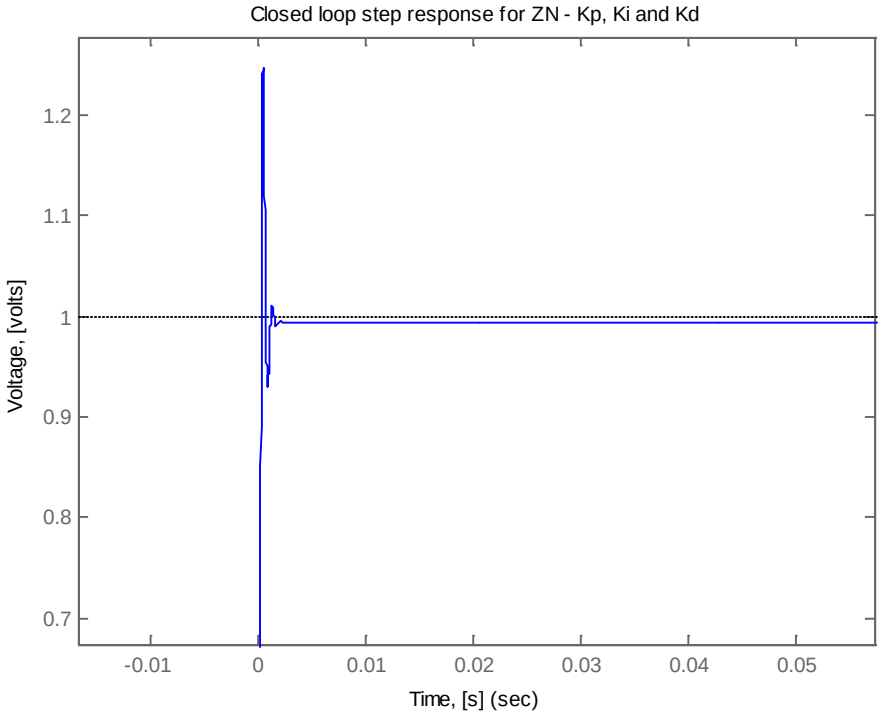

Figure 9.11 – Trial and error values used for the PI parameters output with Kd=0 (zoomed)

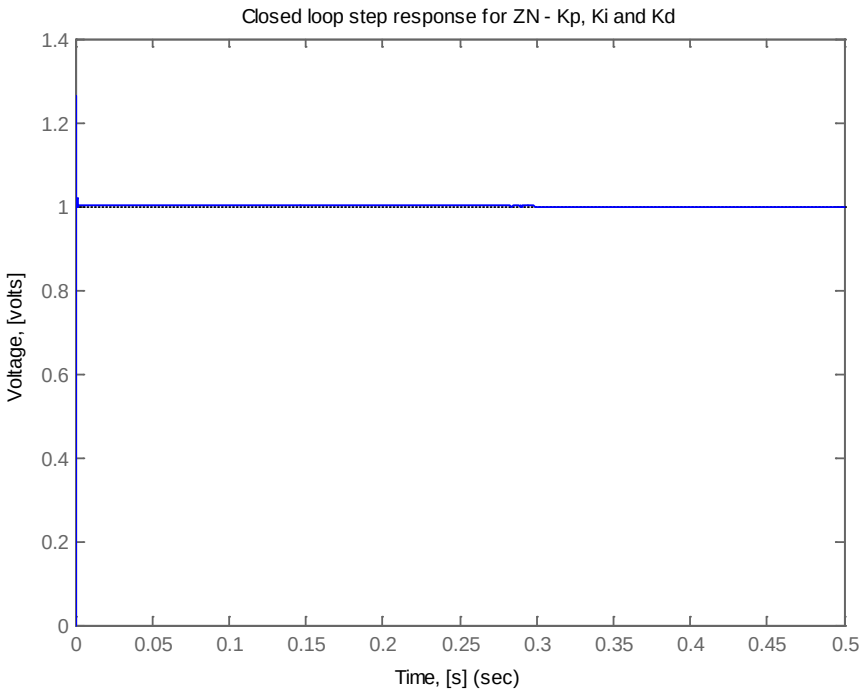

Figure 9.12 – Trial and error values used for the PI parameters output with Ki multiplied 1000 and Kd=0

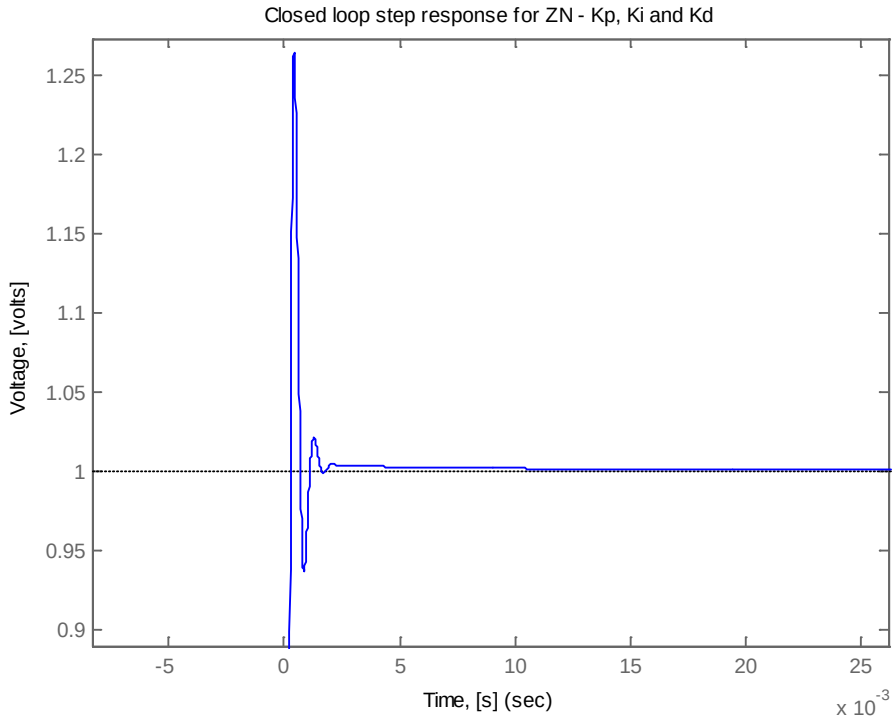

Figure 9.13 – Trial and error values used for the PI parameters output with Ki multiplied 1000 and Kd=0 (zoomed)

9.2.4 Proportional-Integral-Derivative control

But for a more critical assessment of the trial and error method, the M-file – "`tclosedloopPID_TrialError4.m`", was used to obtain a more *perfect* output for the system response as shown in the following figure 9.8. Though, all the PID parameters might not be needed sometimes, but it needful to examine it to check the effect and the difference from the other P and PI combinations. For the implementation of the PID guessed parameters based in the trial and error, the K_I and K_D were set to 1310.6 and 0.0763 respectively. On the first trial the figure – was obtained.

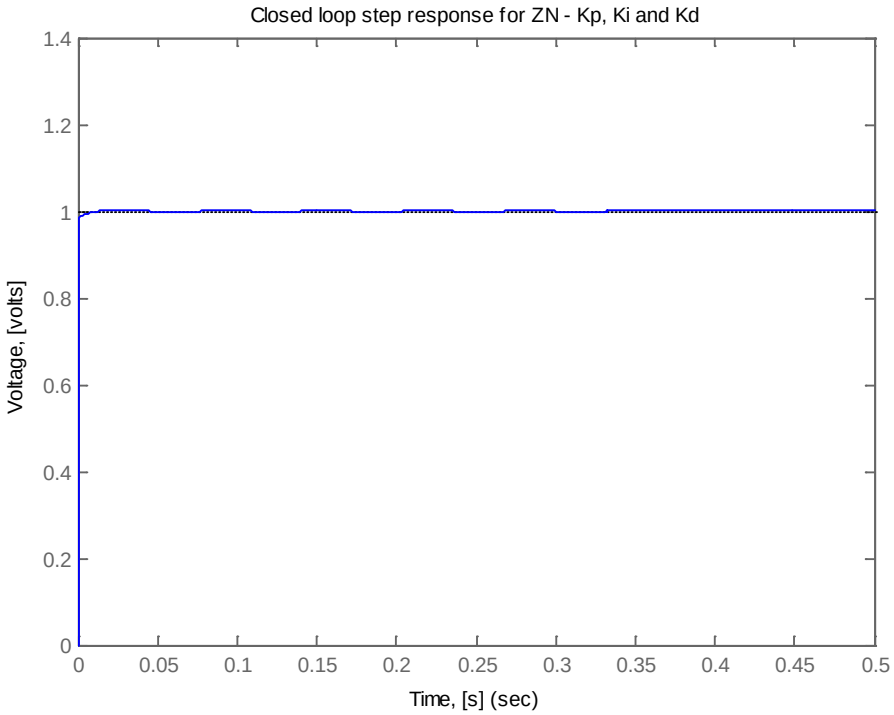

Figure 9.14 – Trial and error method for PID – control effect on the system response (first trial with Kd set at 0.0763)

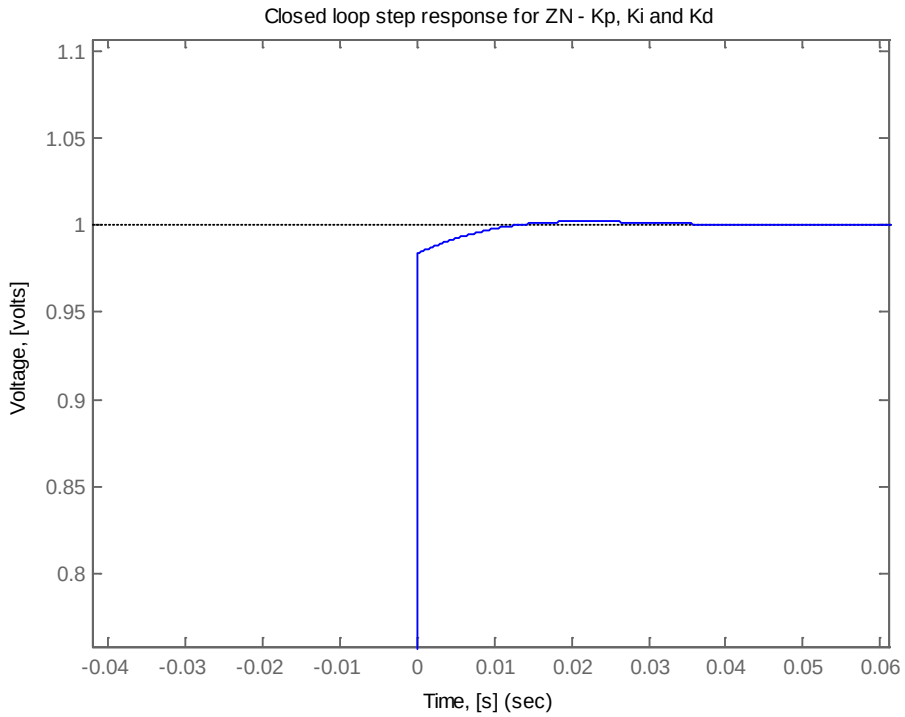

Figure 9.15 – Trial and error method for PID – control effect on the system response (first trial with Kd set at 0.0763, zoomed)

The trial and error gave a reasonable level comfort but it is time consuming and requires extra techniques to be able to have guesses that are appropriate and *near* efficient.

For an overall assessment of the P, PI and PID parameters effect, the following figure was generated for appropriate comparison effects using the `UpdatedPPIPID_TrialError.m`.

	PID Type	K_P	K_I	K_D
1.	P	13.11	0	0
2.	PI	13.11	1310.6	0
3.	PID	13.11	1310.6	0.0763

Table 9.1 – Results of the Trial and Error method for PID controller parameters

UpdatedPPIPID_TrialError.m

```
% Start of code
clear
close all

% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants
num = 1/Ke;
den = [tm*te tm 1];

%----P starts
% assumed Kp = 13.11
Kp1 = 13.11;
numa1 = Kp1 * num;
dena1 = den;

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac1, denac1] = cloop(numa1, dena1);
%----P ends

%----PI Starts
%Trial and Error tuning parameter Kp and Ki
Kp2 = 13.11;
Ki2 = 1310.6;

% For the PI equation
numc2 = [Kp2 Ki2];
denc2 = [1 0];

% convule "num with numc" and "den with demc"
numa2 = conv(num, numc2);
dena2 = conv(den, denc2);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac2, denac2] = cloop(numa2, dena2);
%----PI ends

%----PID Starts
%Trial and Error parameter guessed with support of RH
Kp3 = 13.11;    %Proportional gain
Ki3 = 1310.6;   %Integral gain
Kd3 = 0.0763;    %Derivative gain
% For the PID equation
numc3 = [Kd3 Kp3 Ki3 ];
denc3 = [1 0];
```

UpdatedPPIPID_TrialError.m (contd.)

```
% convule "num with numc" and "den with demc"
numa3 = conv(num, numc3);
dena3 = conv(den, denc3);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac3, denac3] = cloop(numa3, dena3);
%----PID ends

% Plotting the new step-response
t = 0:0.00001:0.01;

% New G1 for overall closed loop transfer function
G1 = tf(numac1, denac1);

G2 = tf(numac2, denac2);

G3 = tf(numac3, denac3);

% Plots the Step Response diagram
figure;
hold on
step(G1, t);
hold on
step(G2, t);
hold on
step(G3, t);
legend('P', 'PI', 'PID');
title('Closed Loop PID Trial and Error step response generated for
```

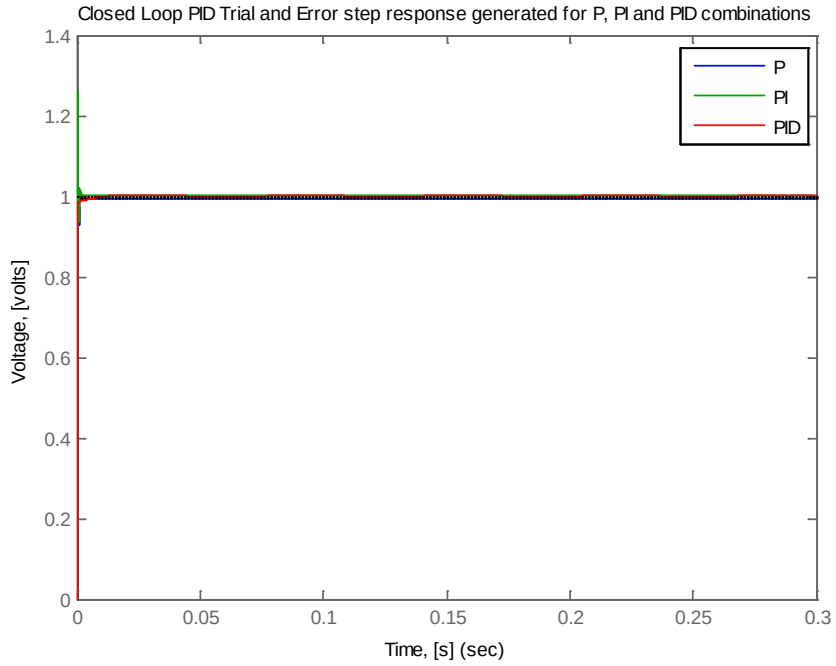

Figure 9.16 – Trial and error method for P, PI and PID – control effect on the system response (t-max=0.3s)

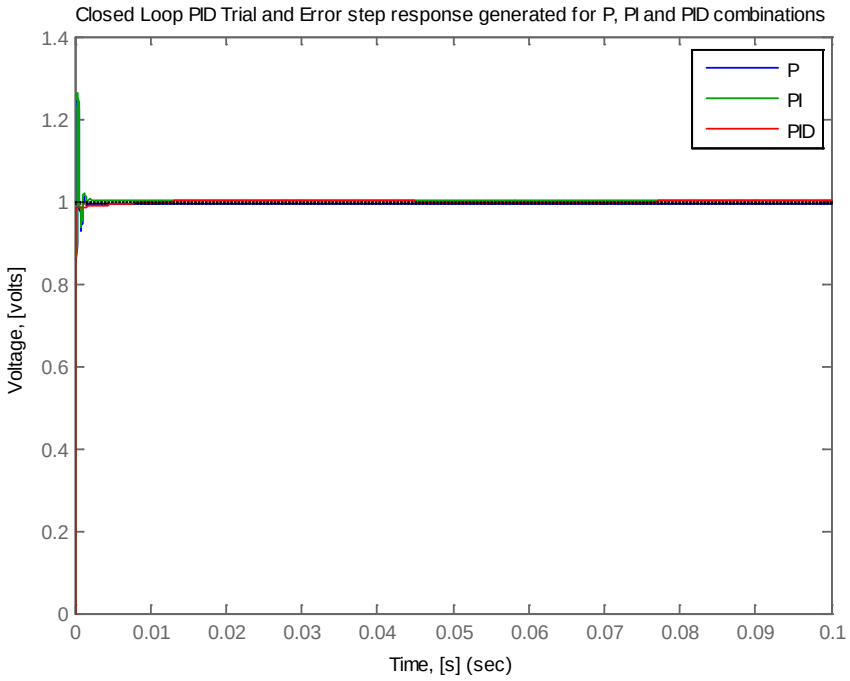

Figure 9.17 – Trial and error method for P, PI and PID – control effect on the system response (t-max=0.1s)

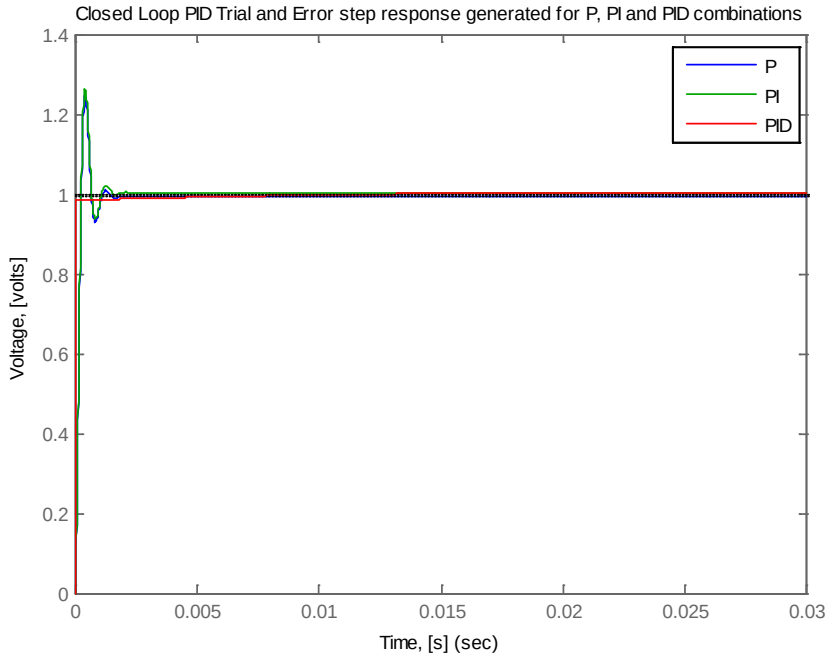

Figure 9.18 – Trial and error method for P, PI and PID – control effect on the system response (t-max=0.03s)

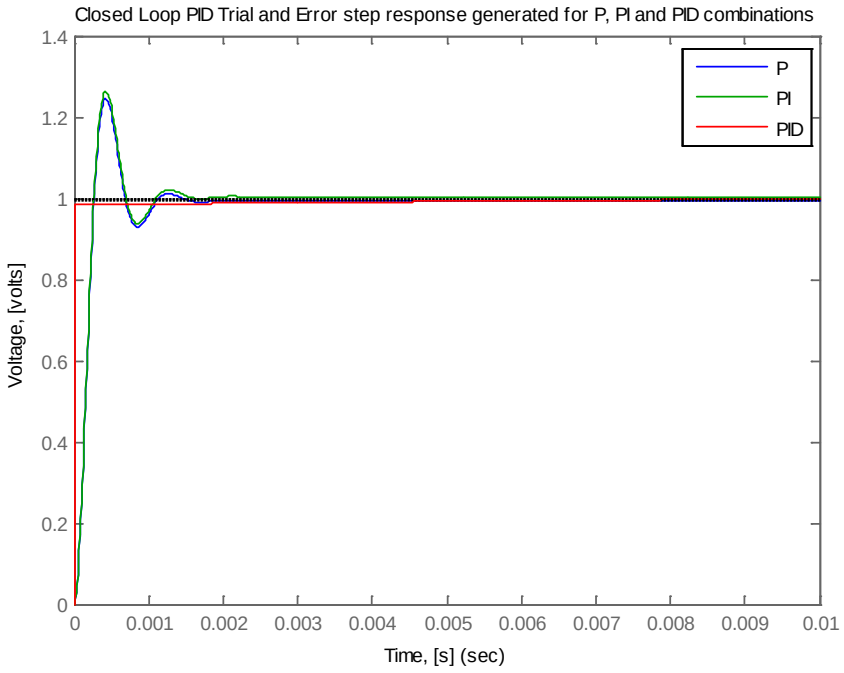

Figure 9.19 – Trial and error method for P, PI and PID – control effect on the system response (t-max=0.01s)

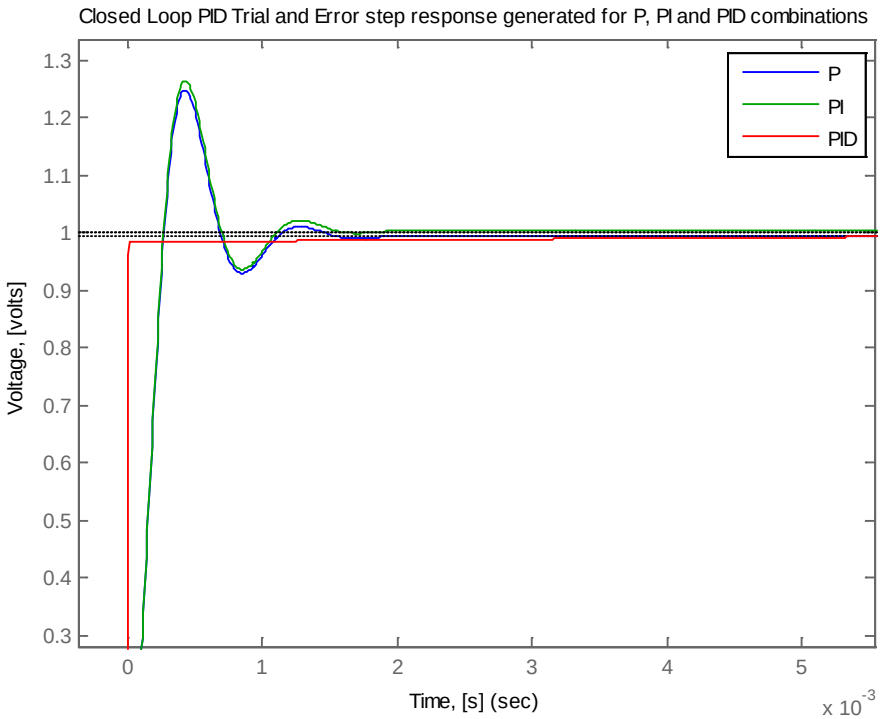

Figure 9.20 – Trial and error method for P, PI and PID – control effect on the system response (1st zooming)

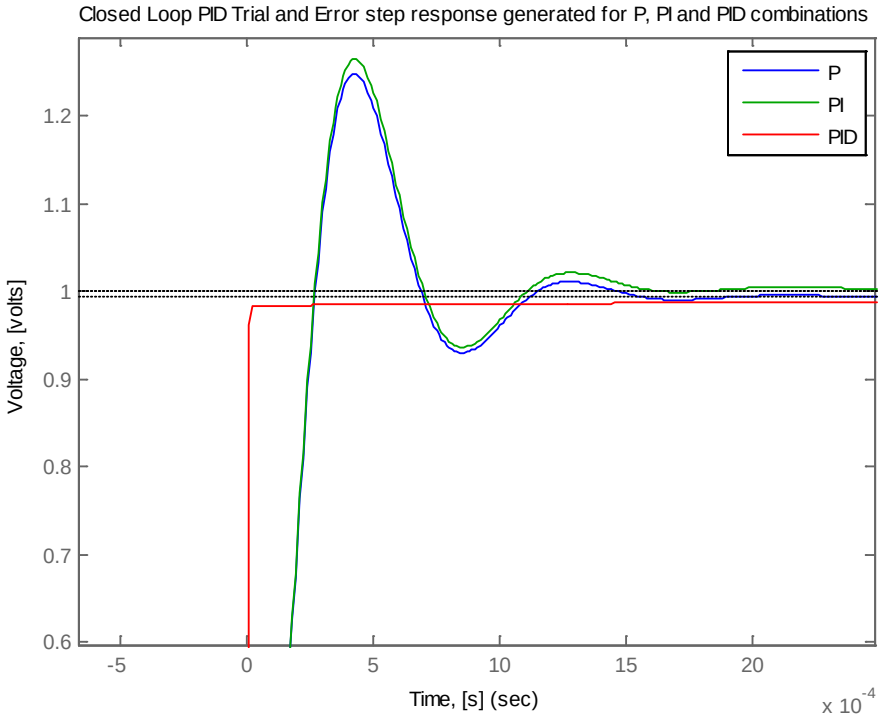

Figure 9.21 – Trial and error method for P, PI and PID – control effect on the system response (2nd zooming)

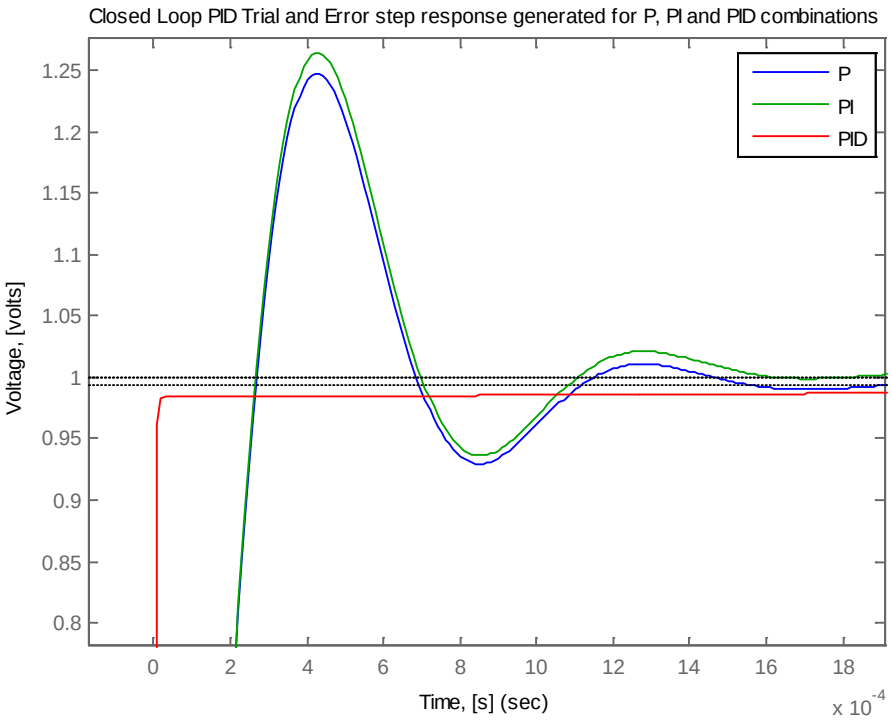

Figure 9.22 – Trial and error method for P, PI and PID – control effect on the system response (3rd zooming)

9.3 Ziegler-Nichols tuning methods

The Ziegler-Nichols method used was done based on obtaining the open loop transfer function and thereafter obtaining the necessary parameter values needed for the various evaluation of the P, PI and PID parameters. The steps taken involve the files `topenloop.m` used in conjunction with the `openloop.mdl` model. So, for the Ziegler-Nichols method analysis the m-file `topenloop_zn.m` was used accordingly.

The open loop step response is characterized by two main parameters, the L (delay time parameter) and T (time constant). These two parameters are computed by drawing tangents to the open loop step response at its point of inflections (basically two points. The inflection points are particularly done so that there would be an intersection with the vertical (voltage axis, which correlates with the steady-state value) and horizontal (time axis) axes.

Based on the Ziegler-Nichols, the following were derived to obtain the control parameters based on the required model:

	PID Type	K_P	$T_I = \frac{K_P}{K_I}$	$T_D = \frac{K_D}{K_P}$
1.	P	$\frac{T}{L}$	∞	0
2.	PI	$0.9\times \frac{T}{L}$	$\frac{L}{0.3}$	0
3.	PID	$1.2\times \frac{T}{L}$	$2 \times L$	$0.5\times L$

Table 9.2 – Ziegler-Nichols PID controller parameters model [10]

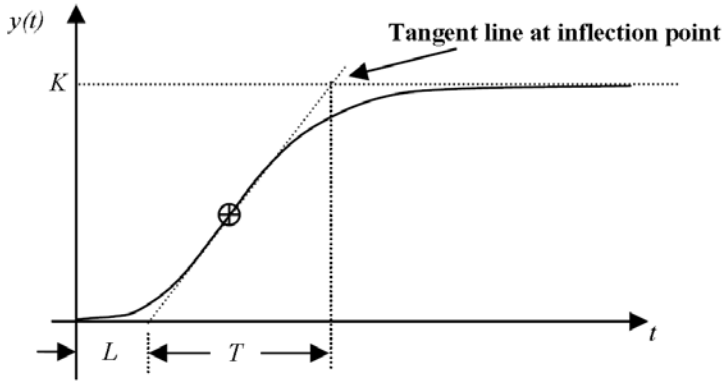

Figure 9.23 – Ziegler-Nichols step response tuning method [10]

From the figure 9.23, the target is on how to evaluate the two parameters (L and T) needed. This is done as follows with the illustration.

topenloop_zn.m

```
%
% Start of code
%
% includes constant parameters
clear
close all

%motor constants
constants

% includes evaluated constants
evaluatedconstants

% Transfer function
G = tf([1/Ke],[tm*te tm 1]);

% Plots the Step Response diagram
figure;
step(G, 0.5);
title('Open Loop Step Response diagram');
xlabel('Time, secs')
ylabel('Voltage, volts')
%grid on;

format long
load openloop.mat
coeff_x=polyfit([6 10 12],openloop(2,[6 10 12]),1)
coeff_y=polyfit([700:900],openloop(2,[700:900]),1)

for n=1:100
    zn_line_x(n)=coeff_x(1)*n+coeff_x(2);
end

for n=1:900
    zn_line_y(n)=coeff_y(1)*n+coeff_y(2);
end

figure(2)
hold on
plot(openloop(2,:),'red')
plot(zn_line_x);
plot((zn_line_y), 'green');
legend('1step response','line');
grid on
axis([0 400 0 14]);
l=length(openloop(2,:))
L_samples=roots(coeff_x)

%inflecton_point=intersect(zn_line_x,zn_line_y)
[a,b,c]=intersect(zn_line_x,zn_line_y)

% End of code
```

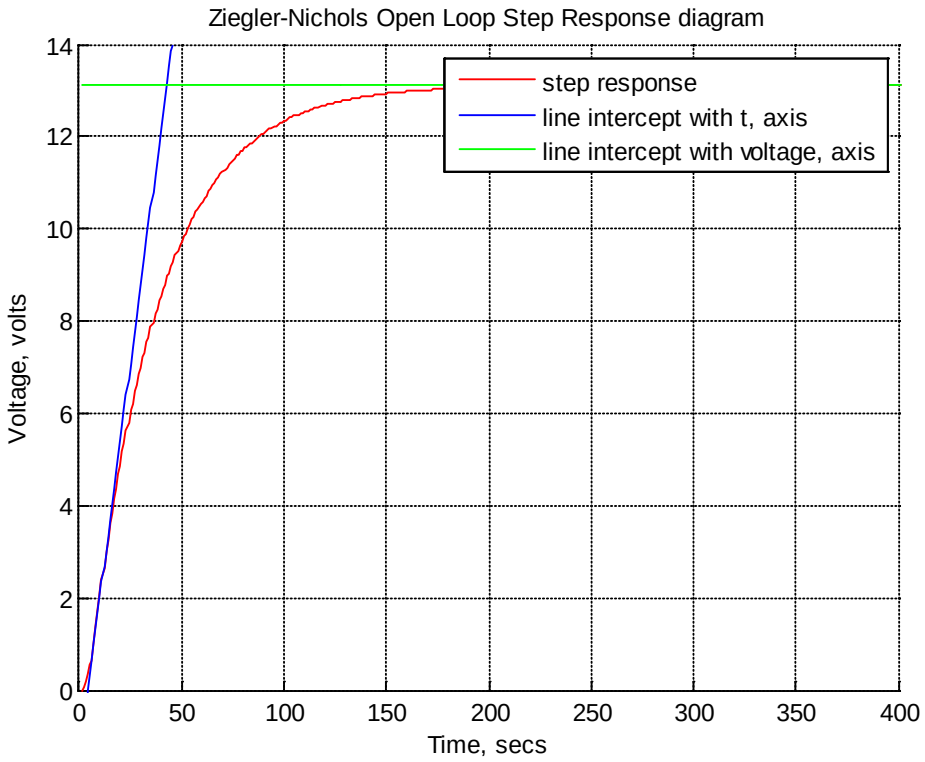

Figure 9.24 – Ziegler-Nichols open step response plot computation

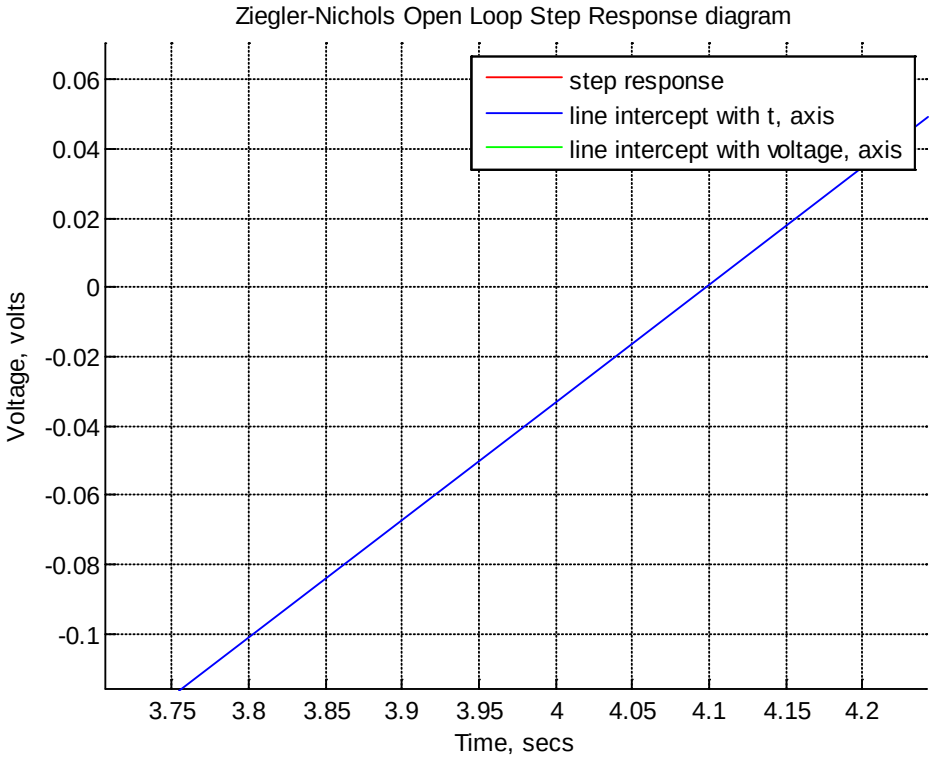

Figure 9.25 – Ziegler-Nichols open step response horizontally zoomed

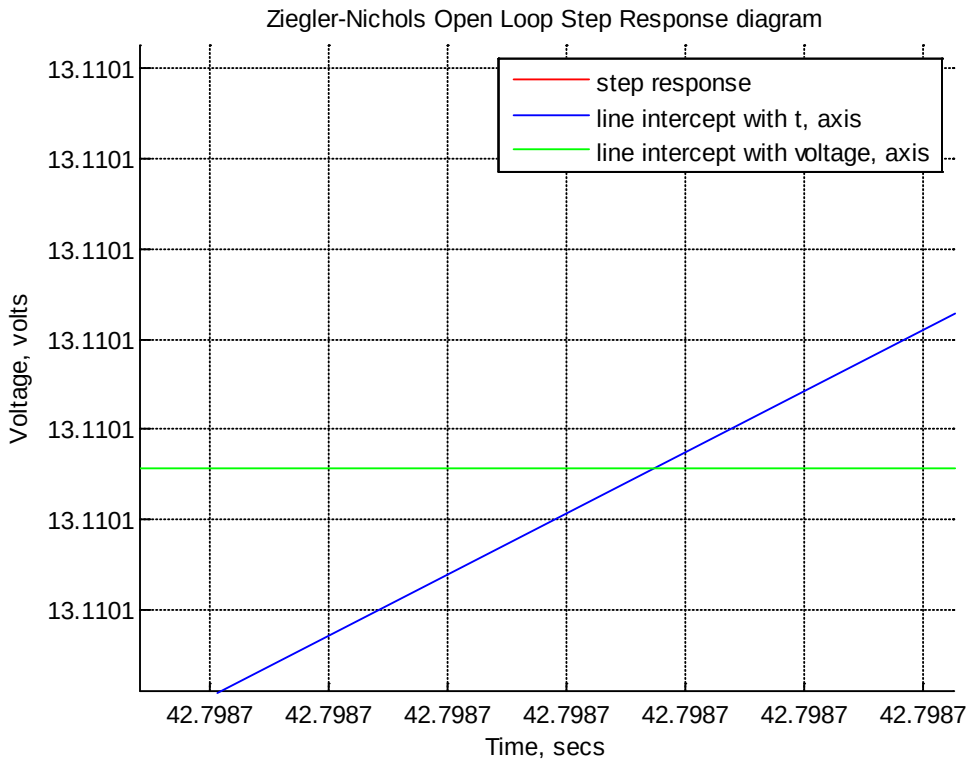

Figure 9.26 – Ziegler-Nichols open step response vertically zoomed

Therefore, from the figure 9.24, figure 9.25 and figure 9.26, the values of the L and T could be computed as follows:

An assumed sample rate of 1000 was used for the `topenloop_zn.m` plots

Point of interception of the horizontal line ≈ 4.1 (voltage = 0)

Coordinate of the point of interception of the two lines ≈ (T^*, K) = (42.7987, 13.1101);

Where,

T^* is horizontal trace of the interception on the tangent lines drawn

L = 4.1;

K = 13.1101;

T = T^* – L = 4.1 = 42.7987 - 38.6987 ≈ 38.70

This implies that we have:

L = 0.0041;

K = 13.1101;

T = 0.0387

With the above computation, the P, PI and PID computation was done to get the best suited parameters combination desired.
So the updated table 9.1 would be table 9.2 shown below:

	PID Type	K_P	$T_I = \frac{K_P}{K_I}$	$T_D = \frac{K_D}{K_P}$
1.	P	9.439	∞	0
2.	PI	8.495	0.0137	0
3.	PID	11.327	0.0082	0.00205

Table 9.3 – Results of the Ziegler-Nichols method for PID controller parameters

From table 9.2, the following parameters are obtained based on the equation format (from equation 7.3 above) to become equation 9.1 below:

For P only,

$$K_P + \frac{K_I}{s} + K_D \cdot s = 9.439 + \frac{K_I}{s} + K_D \cdot s \tag{9.1}$$

For PI only,

$$K_P + \frac{K_I}{s} + K_D \cdot s = 8.495 + \frac{620.07}{s} + K_D \cdot s \tag{9.2}$$

For PID only,

$$K_P + \frac{K_I}{s} + K_D \cdot s = 11.327 + \frac{1381.34}{s} + 0.0232 \cdot s \tag{9.3}$$

Using the figure 9.1 (above) and m-file `tclosedloopP_zn.m`, `tclosedloopPI_zn.m` and `tclosedloopPID_zn.m`, the outputs of the various PID combinations could be obtained as given below:

tclosedloopP_zn.m

```
%start of code
clear
close all

% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants

num = 1/Ke;
den = [tm*te tm 1];

% assumed Kp = 10
Kp = 9.439;
numa = Kp * num;
dena = den;

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed transfer function in
this case
[numac, denac] = cloop(numa, dena);

% Plotting the new step-response
t = 0:0.00001:0.005
step(numac, denac, t);      % across 0.01 seconds timing
title('Closed step response with proportion, P control; Kp =
9.439');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
grid on;

% New G1 for overall closed loop trasnfer function
G1 = tf(numac, denac);

% plots the Root-locus
figure;
rlocus(G1);
title('Closed Loop Root Locus diagram');
grid on;

% plots the Nyquist diagram
figure;
nyquist(G1);
title('Closed Loop Nyquist diagram');
grid on;

% plots the Bode Plot
figure;
bode(G1,{0.1 , 100})
title('Closed Loop Bode plot diagram with wider frequency
spacing');
grid on;

%end of code
```

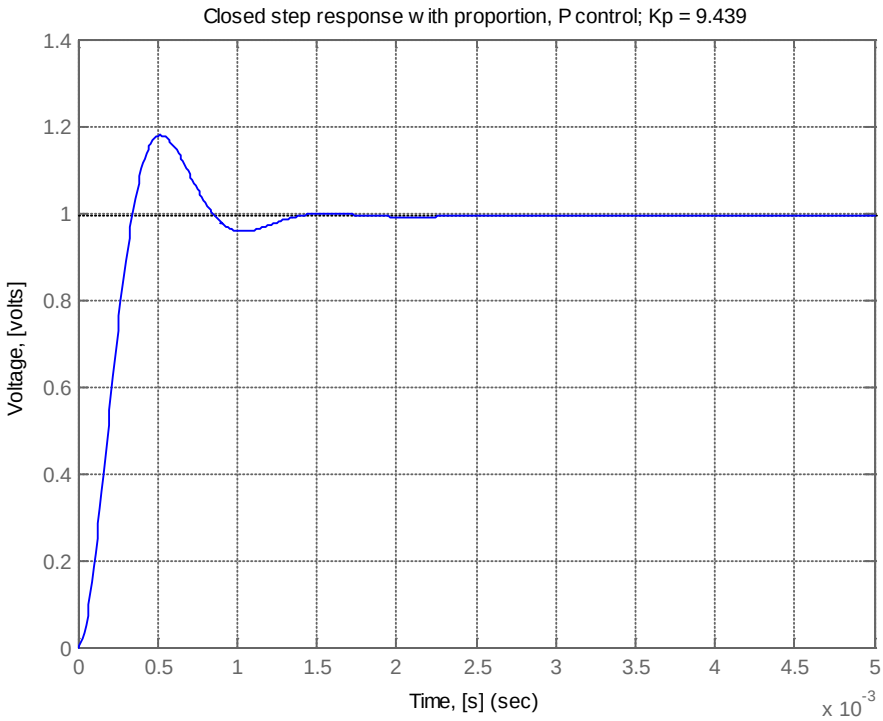

Figure 9.27 – P output for the Ziegler-Nichols tuning method

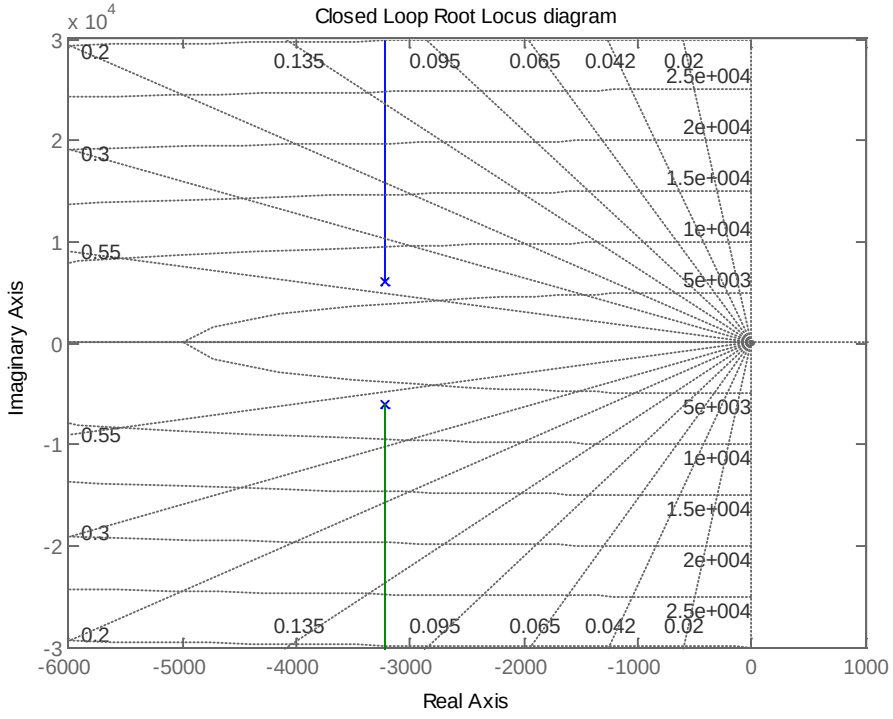

Figure 9.28 – P output for the Ziegler-Nichols tuning method root locus output

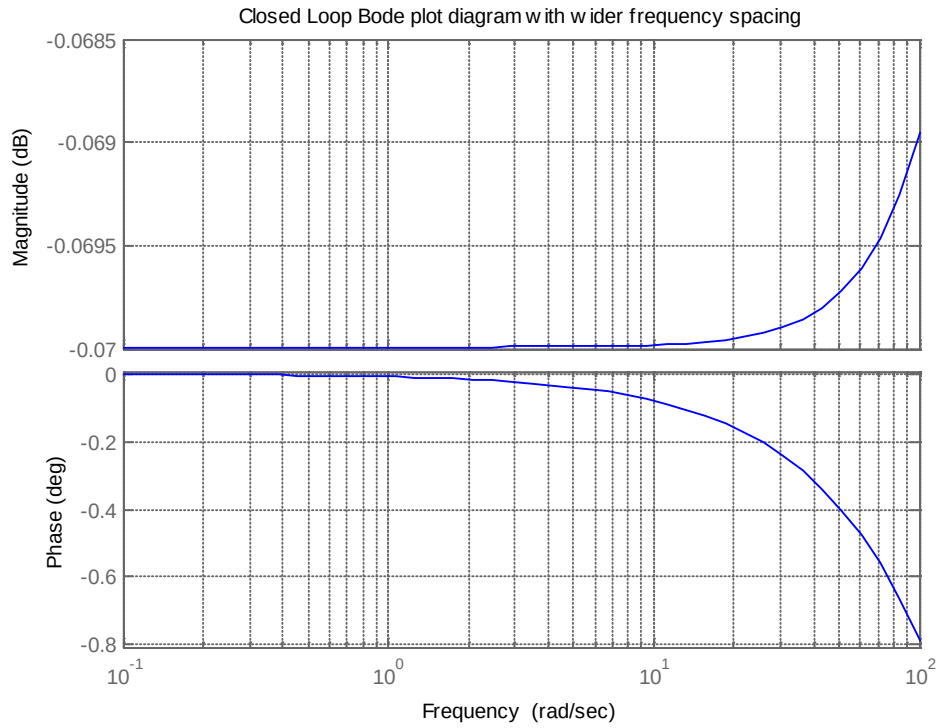

Figure 9.29 – P output for the Ziegler-Nichols tuning method Bode plot output

tclosedloopPI_zn.m

```
% Start of code
clear
close all
% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants
num = 1/Ke;
den = [tm*te tm 1];

%Ziegler-Nichol tuning parameter Kp and Ki
Kp = 8.495;
Ki = 620.07;

% For the PI equation
numc = [Kp Ki];
denc = [1 0];
% convule "num with numc" and "den with demc"
numa = conv(num, numc);
dena = conv(den, denc);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac, denac] = cloop(numa, dena);

% Plotting the new step-response
t = 0:0.00001:0.005
step(numac, denac, t);      % across 0.01 seconds timing
title('Closed step response with proportion, P control; Kp = 8.495
and Ki = 620.07');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
grid on;

% New G1 for overall closed loop trasnfer function
G1 = tf(numac, denac);
% plots the Root-locus
figure;
rlocus(G1);
title('Closed Loop Root Locus diagram');
grid on;
% plots the Nyquist diagram
figure;
nyquist(G1);
title('Closed Loop Nyquist diagram');
grid on;
% plots the Bode Plot
figure;
bode(G1,{0.1 , 100})
title('Closed Loop Bode plot diagram with wider frequency
spacing');
grid on;
%end of code
```

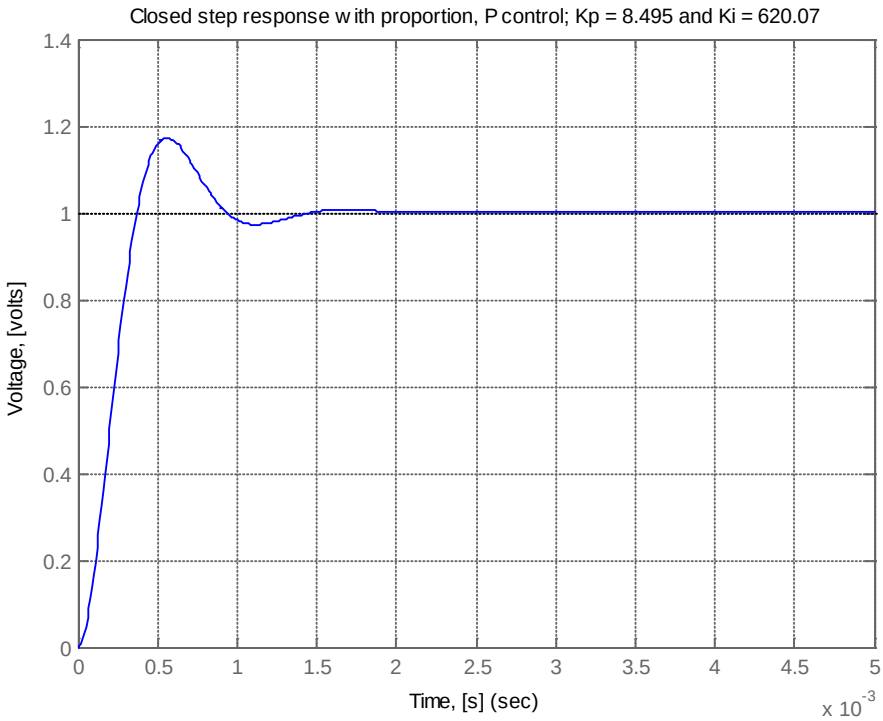

Figure 9.30 – PI output for the Ziegler-Nichols tuning method

tclosedloopPID_zn.m

```
% Start of code
% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants
num = 1/Ke;
den = [tm*te tm 1];

%Ziegler-Nichols parameter computed
Kp = 11.327;    %Proportional gain
Ki = 1381.34;   %Integral gain
Kd = 0.0232;    %Derivative gain

% For the PID equation
numc = [Kd Kp Ki ];
denc = [1 0];

% convule "num with numc" and "den with demc"
numa = conv(num, numc);
dena = conv(den, denc);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac, denac] = cloop(numa, dena);

% Plotting the new step-response
t = 0:0.00001:0.3;
step(numac, denac, t);      % across 0.01 seconds timing
title('Closed loop step response for ZN - Kp, Ki and Kd');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
%grid on;

% New G1 for overall closed loop trasnfer function
G1 = tf(numac, denac);

% plots the Root-locus
figure;
rlocus(G1);
title('Closed Loop Root Locus diagram');
grid on;
% plots the Nyquist diagram
figure;
nyquist(G1);
title('Closed Loop Nyquist diagram');
grid on;
% plots the Bode Plot
figure;
bode(G1,{0.1 , 100})
title('Closed Loop Bode plot diagram with wider frequency
spacing');
grid on;
%% End of code
```

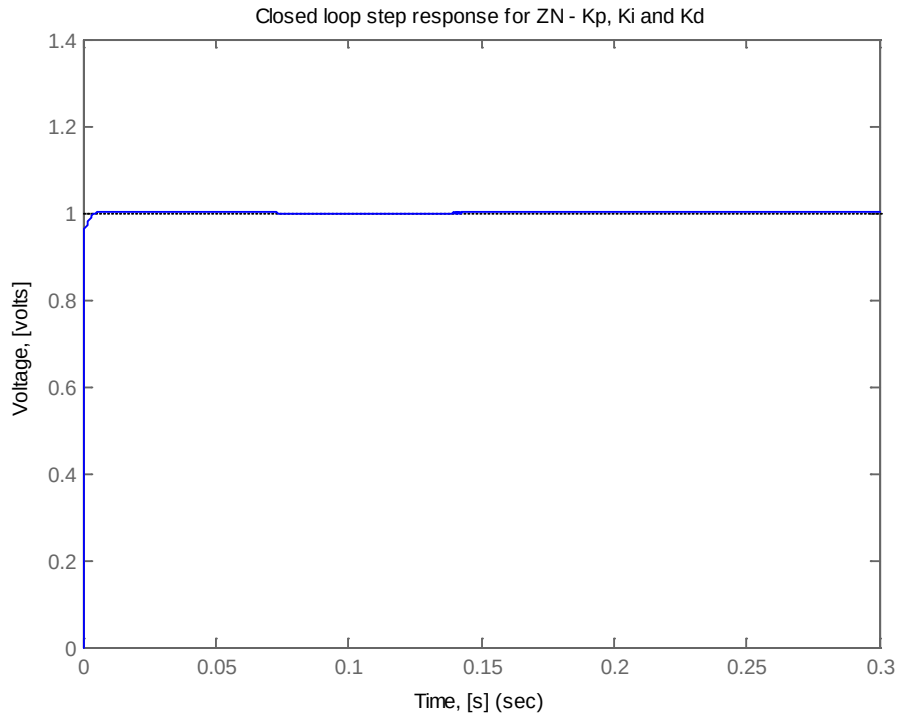

Figure 9.31 – Auto-scaled PID output for the Ziegler-Nichols tuning method

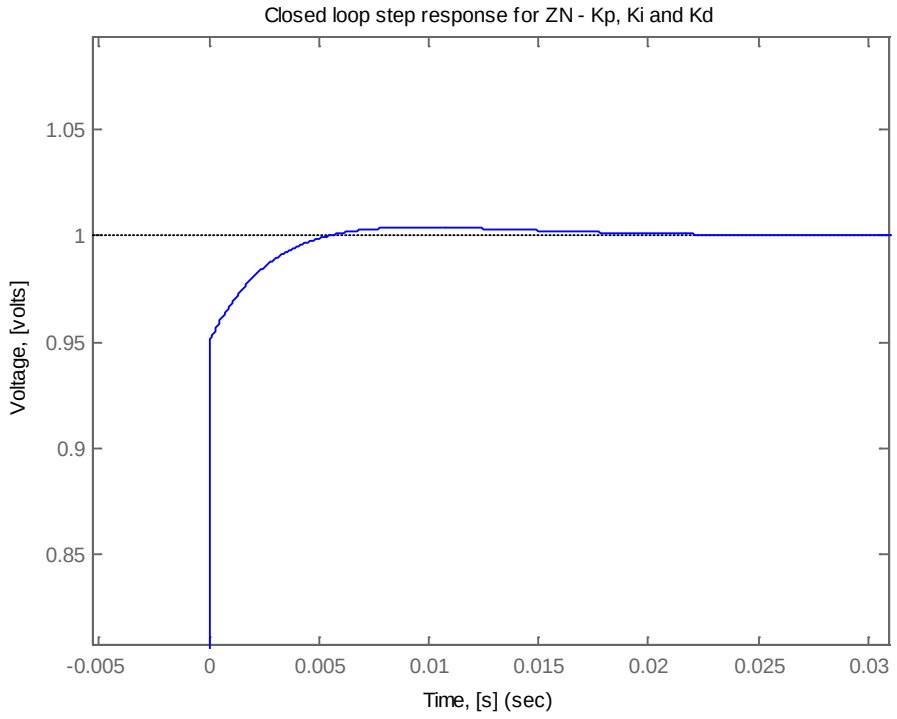

Figure 9.32 – Auto-scaled PID output for the Ziegler-Nichols tuning method (zoomed overshoot point)

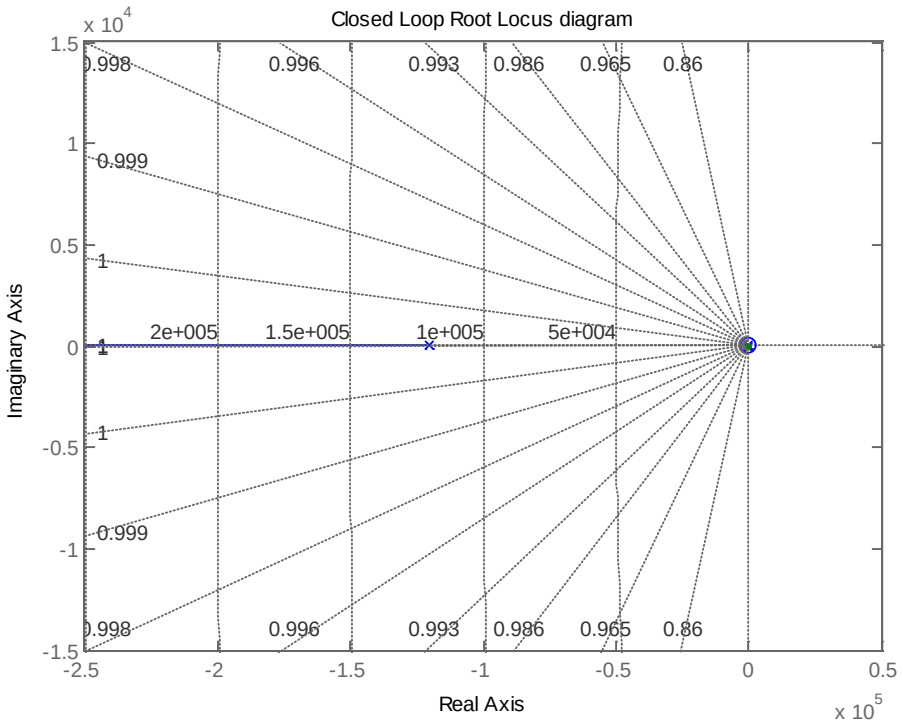

Figure 9.33 – PID Ziegler-Nichols tuning method Root locus diagram

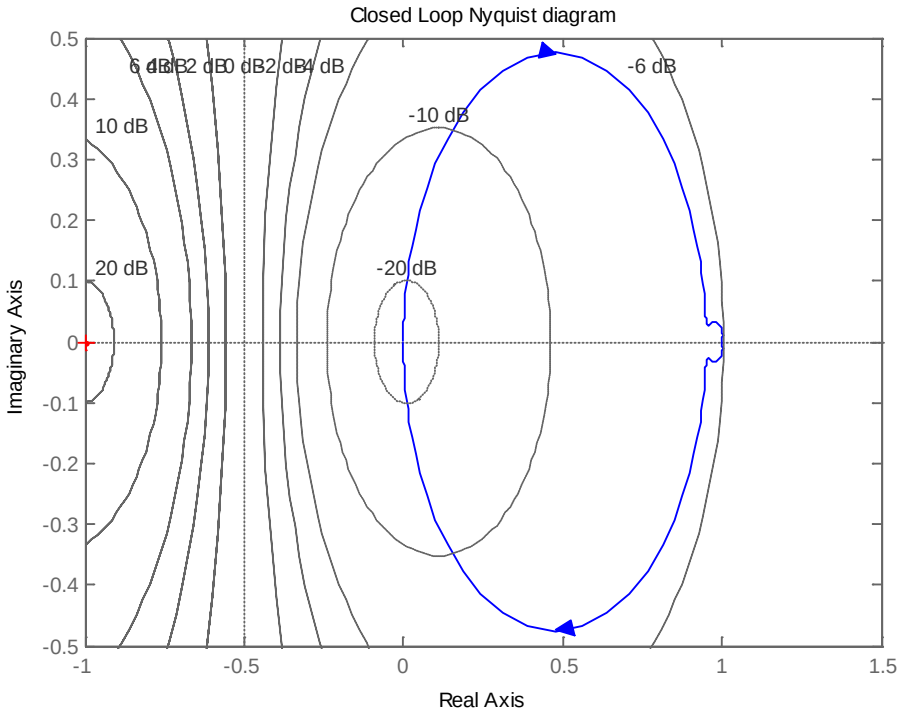

Figure 9.34 – PID Ziegler-Nichols tuning method Nyquist diagram

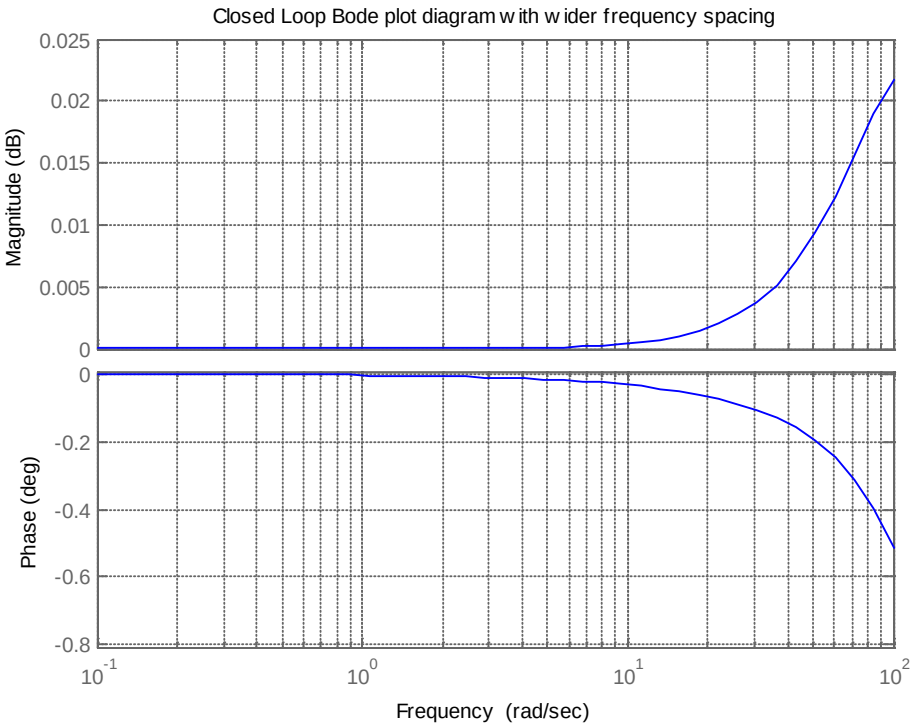

Figure 9.35 – PID Ziegler-Nichols tuning method Bode plot diagram

For a combined comparison of the Ziegler-Nichols tuning methods for the P, PI and PID, a separate m-file, `UpdatedPPIPID_znj.m` was created to execute the combination and this was done over different time spans (0.01, 0.03, 0.1 and 0.3). The various outputs figures are shown in figures 9.23, 9.24, 9.25 and 9.26.

UpdatedPPIPID_znj.m

```
%
% Start of code
%
clear
close all

% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants
num = 1/Ke;
den = [tm*te tm 1];


%----P starts
% assumed Kp = 10
Kp1 = 9.439;
numa1 = Kp1 * num;
dena1 = den;

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac1, denac1] = cloop(numa1, dena1);
%----P ends

%----PI Starts
%Ziegler-Nichols parameter computed
%Ziegler-Nichol tuning parameter Kp and Ki
Kp2 = 8.495;
Ki2 = 620.07;

% For the PI equation
numc2 = [Kp2 Ki2];
denc2 = [1 0];

% convule "num with numc" and "den with demc"
numa2 = conv(num, numc2);
dena2 = conv(den, denc2);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac2, denac2] = cloop(numa2, dena2);
%----PI ends

%----PID Starts
%Ziegler-Nichols parameter computed
Kp3 = 11.327;    %Proportional gain
Ki3 = 1381.34;   %Integral gain
Kd3 = 0.0232;    %Derivative gain
```

UpdatedPPIPID_znj.m (contd.)

```
% For the PID equation
numc3 = [Kd3 Kp3 Ki3 ];
denc3 = [1 0];

% convule "num with numc" and "den with demc"
numa3 = conv(num, numc3);
dena3 = conv(den, denc3);

% For the closed-loop transfer function, the following is obtained
% numac and denac used for the overall closed tranfer function in
this case
[numac3, denac3] = cloop(numa3, dena3);
%----PID ends

% Plotting the new step-response
t = 0:0.00001:0.3;

% New G1 for overall closed loop transfer function
G1 = tf(numac1, denac1);

G2 = tf(numac2, denac2);

G3 = tf(numac3, denac3);

% Plots the Step Response diagram
figure;
hold on
step(G1, t);
hold on
step(G2, t);
hold on
step(G3, t);
legend('P', 'PI', 'PID');
title('Closed Loop PID ZN step response generated for P, PI and
PID combinations');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
% End of code
```

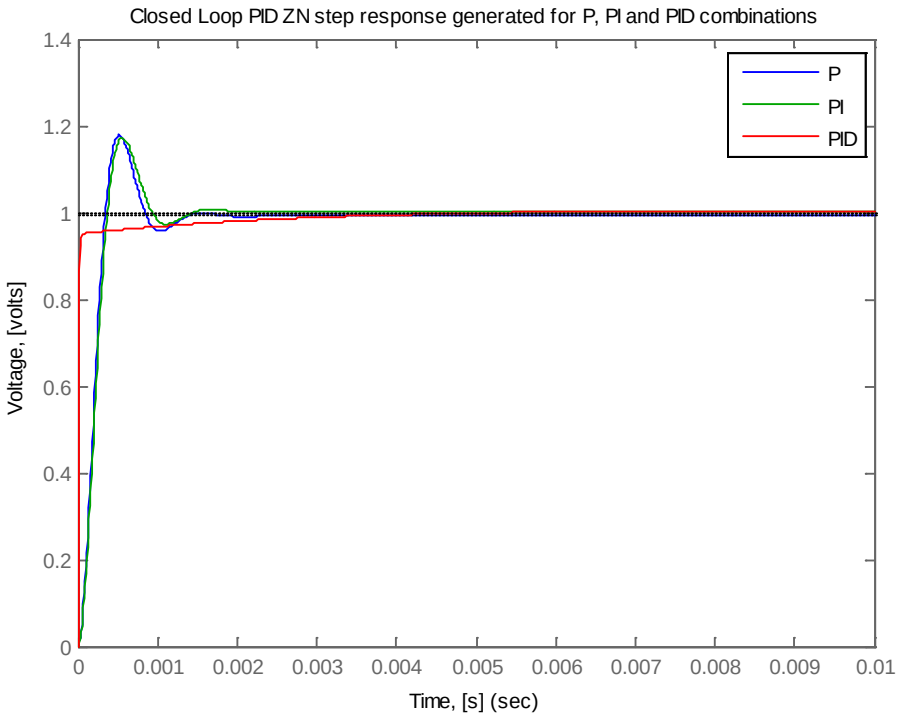

Figure 9.36 – Closed loop PID response for P, PI and PID with t-max=0.01s

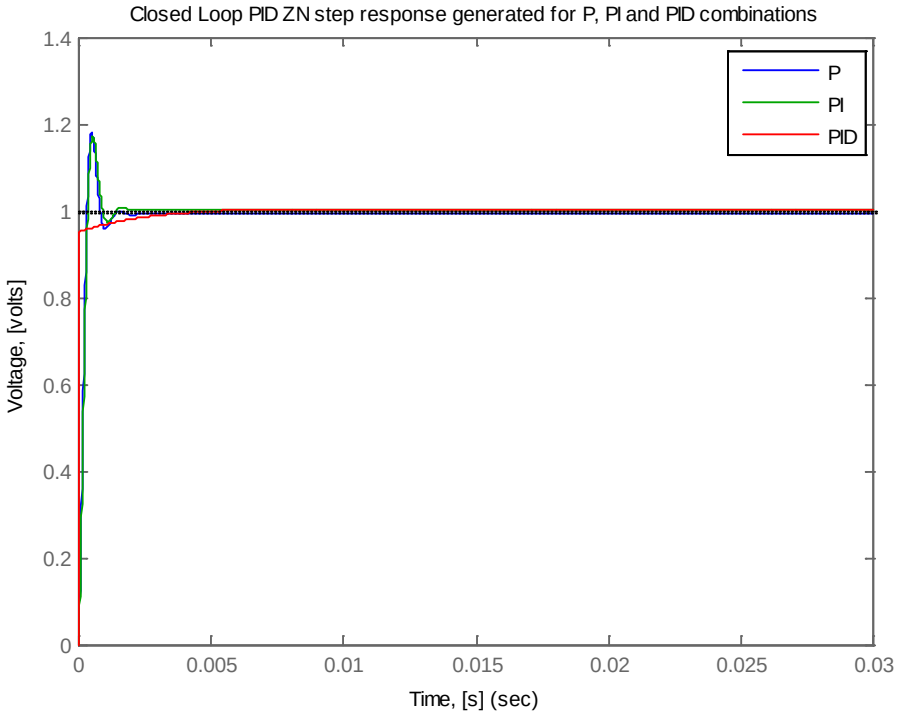

Figure 9.37 – Closed loop PID response for P, PI and PID with t-max=0.03

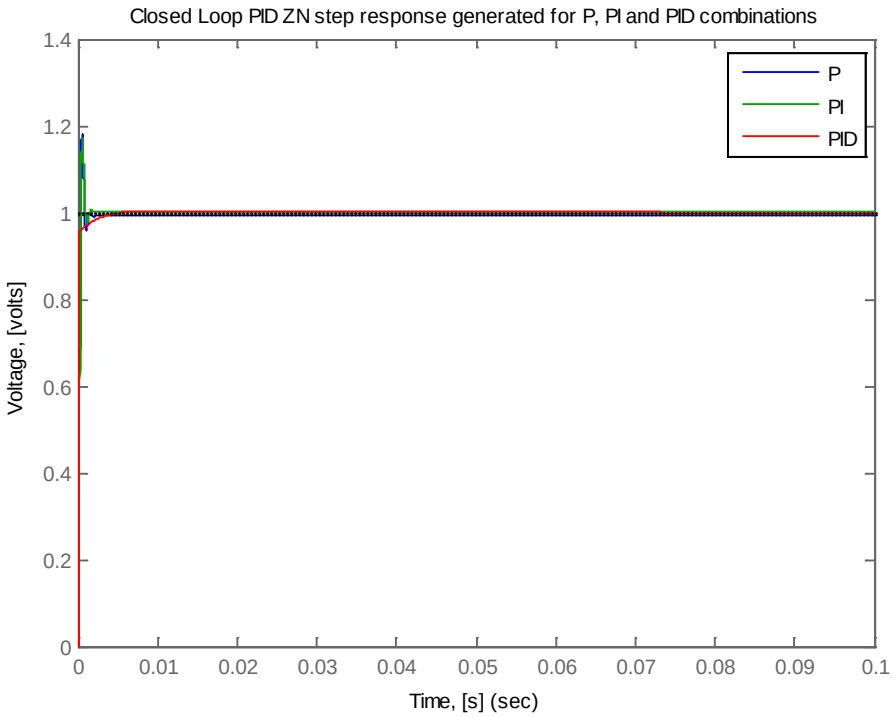

Figure 9.38 – Closed loop PID response for P, PI and PID with t-max=0.1

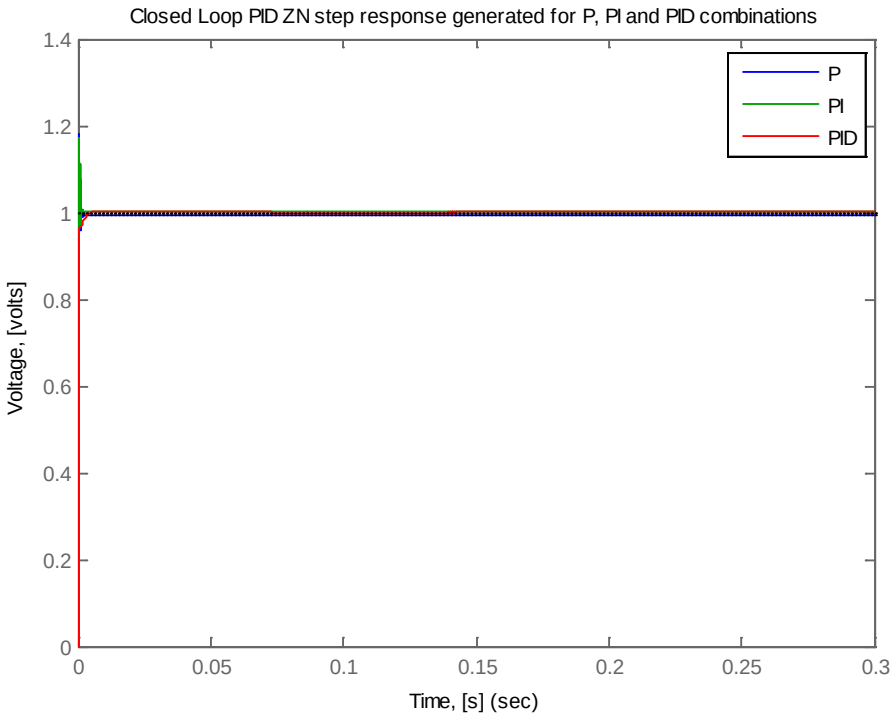

Figure 9.39 – Closed loop PID response for P, PI and PID with t-max=0.3

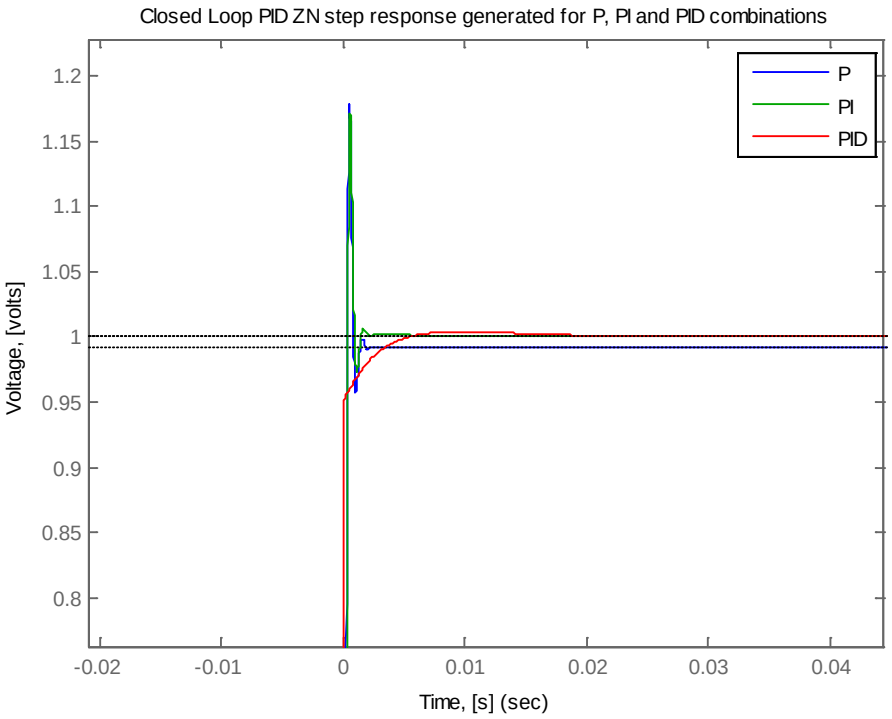

Figure 9.40 – Closed loop PID response for P, PI and PID (1st Zoom)

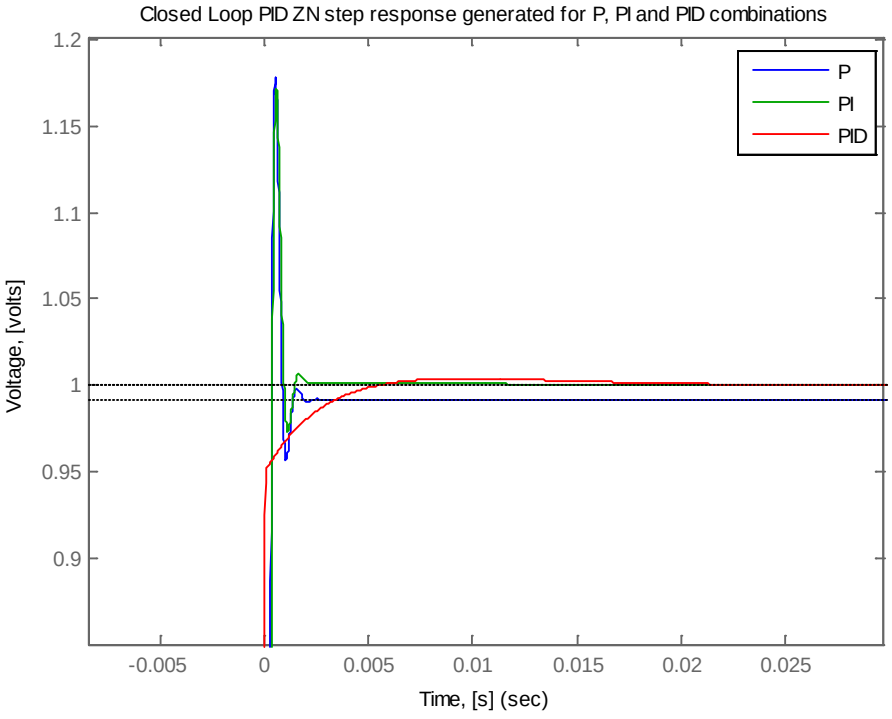

Figure 9.41 – Closed loop PID response for P, PI and PID (2nd Zoom)

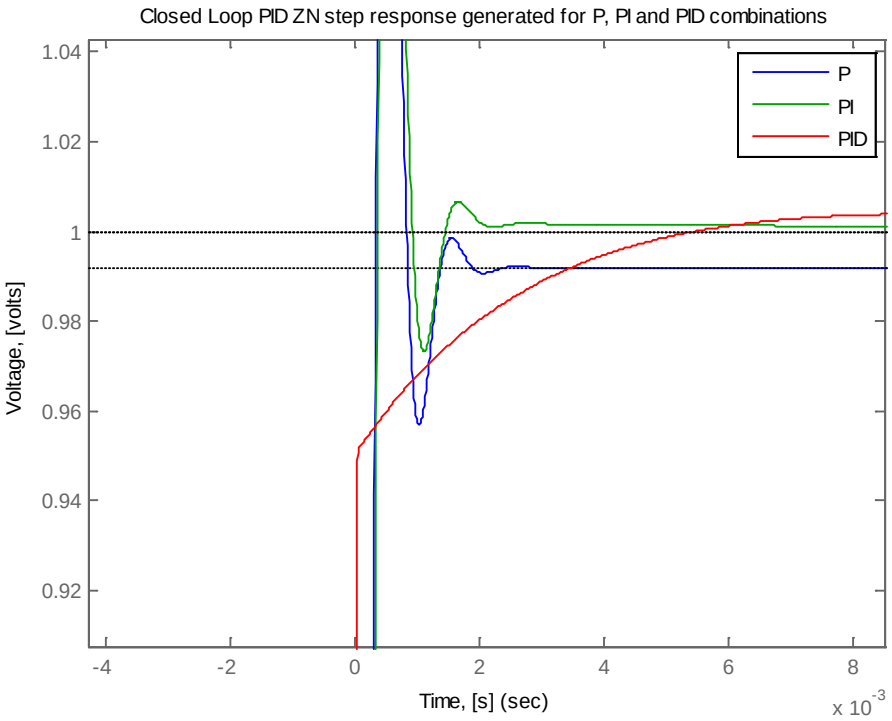

Figure 9.42 – Closed loop PID response for P, PI and PID (3rd Zoom)

9.4 Comparison effects of Trial and Error with Ziegler-Nichols tuning methods

This is made by creating m-file, `UpdatedPID_TErrznj.m` for only the PID parameters effects. The generated figure is as shown below in figure--:

UpdatedPPIPID_TErrznj.m

```
%
% Start of code
%
clear
close all

% includes constant parameters
constants
% includes evaluated constants
evaluatedconstants
num = 1/Ke;
den = [tm*te tm 1];

%Trial and Error PID parameters part
TErrorPID
%Ziegler-Nichols PID parameters part
ZNPIDcomp

% Plotting the new step-response
t = 0:0.00001:0.03;

% New G for overall closed loop transfer function
GZN = tf(numacZN, denacZN);

GTErr = tf(numacTErr, denacTErr);

% Plots the Step Response diagram
figure;
hold on
step(GZN, t);
hold on
step(GTErr, t);

legend('Trial and Error PID', 'Ziegler-Nichols PID');
title('Closed Loop PID for Trial and Error/Ziegler-Nichols step
response output for PID');
xlabel('Time, [s]')
ylabel('Voltage, [volts]')
% End of code
```

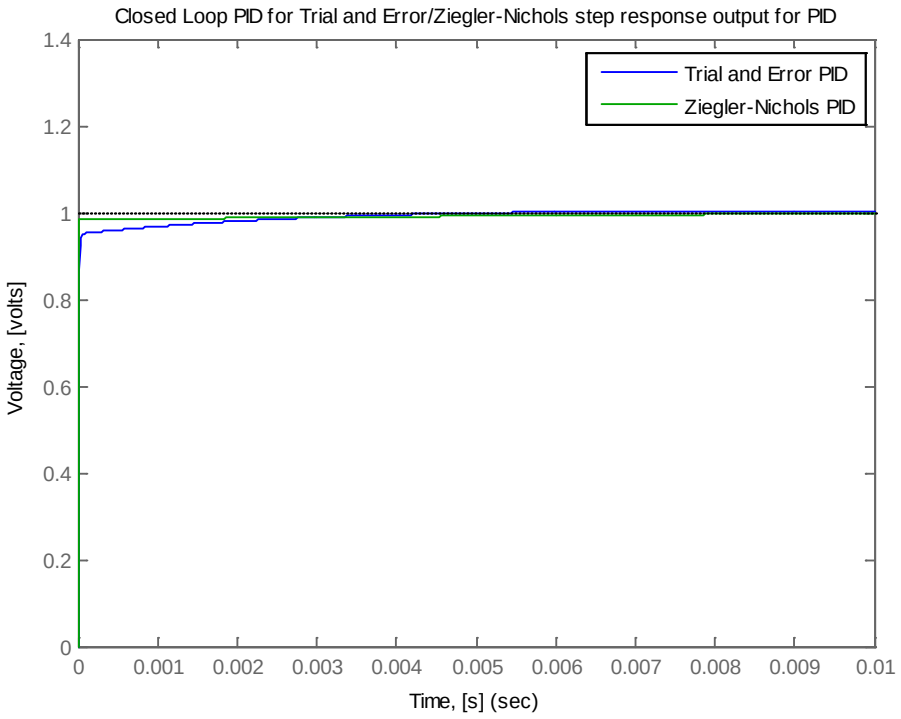

Figure 9.43 – Closed loop response for Trial and Error/Ziegler-Nichols tuning methods

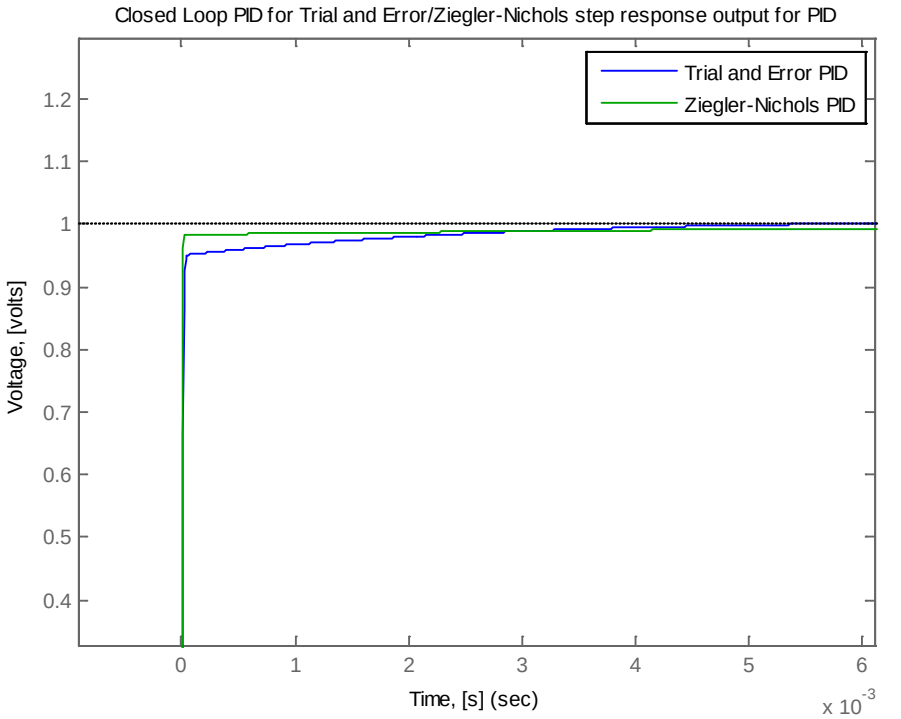

Figure 9.44 – Closed loop response for Trial and Error/Ziegler-Nichols tuning methods (1st zoomed)

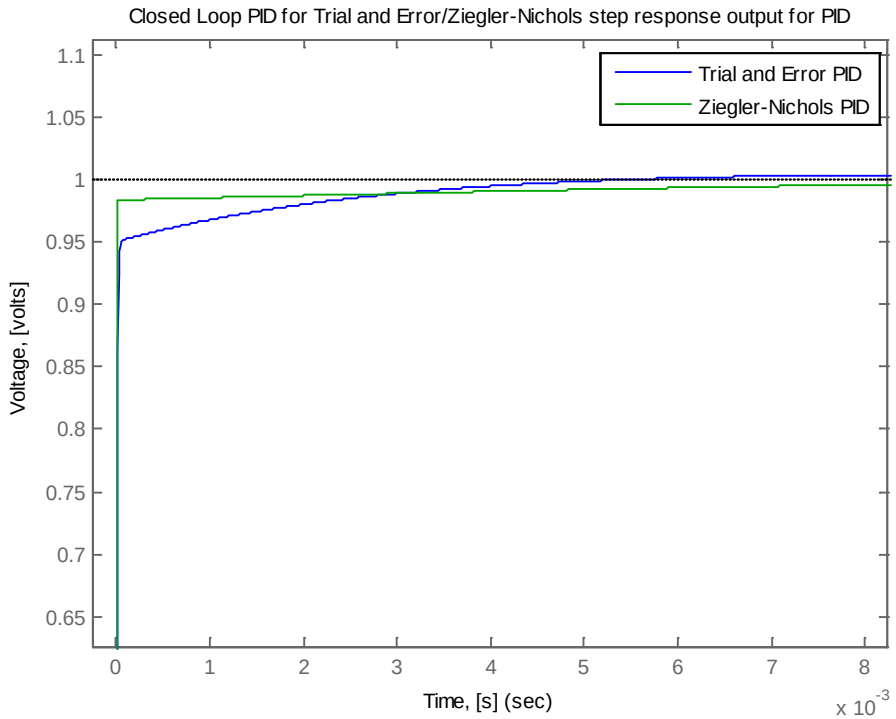

Figure 9.45 – Closed loop response for Trial and Error/Ziegler-Nichols tuning methods (2nd zoomed, right side)

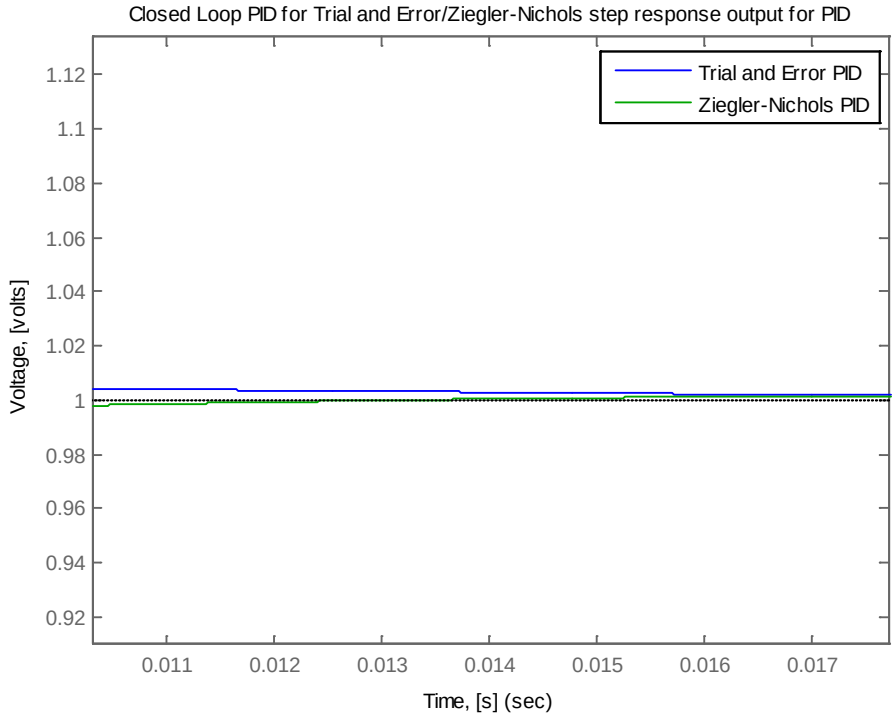

Figure 9.46 – Closed loop response for Trial and Error/Ziegler-Nichols tuning methods (3rd zoomed, left side, with t-max=0.03)

10 FOOTBALL PITCH LAYOUT MODEL

10.1 Dimensions of the Pitch

The football pitch model serves as the background for the main plot of the trajectory path of the robot wheel. The basic dimensions (in millimetres) used are given below and are scaled down to metres in the model plot.

	Part Label	Dimension (mm)
1.	Length	6050
2.	Width	4050
3.	Centre circle (radius)	500

Table 10.1 – Dimensions of the Football pitch layout model

The figure 10.1 below shows the main target design based on the *Laws of the F180 League 2009* [12]. The needed design for the thesis is the main pitch shown in white lines.

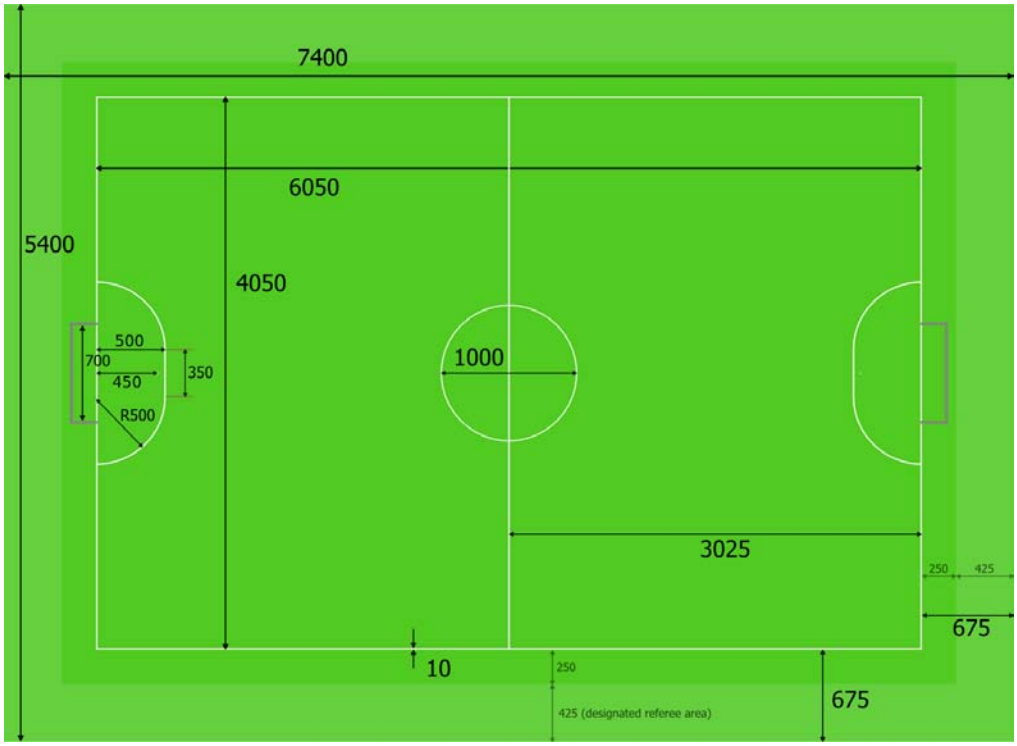

Figure 10.1 – Dimension of the standard pitch required [12]

10.2 Football pitch MATLAB design implementation

The figure 10.2 shows the full part label of the football pitch layout. But the main target design is shown in figure 10.1. This follows with the m-files used to generate the whole pitch layout `robotpatternUpdated.m`, `newFieldSpec.m`, `robotBlockpart.m`, `semiCircleBottomLeft.m`, `semiCircleBottomRight.m`, `semiCircleTopLeft.m`, `semiCircleTopRight.m` and `testcir2.m` (for centre circle plot) used to generate the actual design and shown in figure 10.2. The output football pitch generated is given under figure 10.3 below.

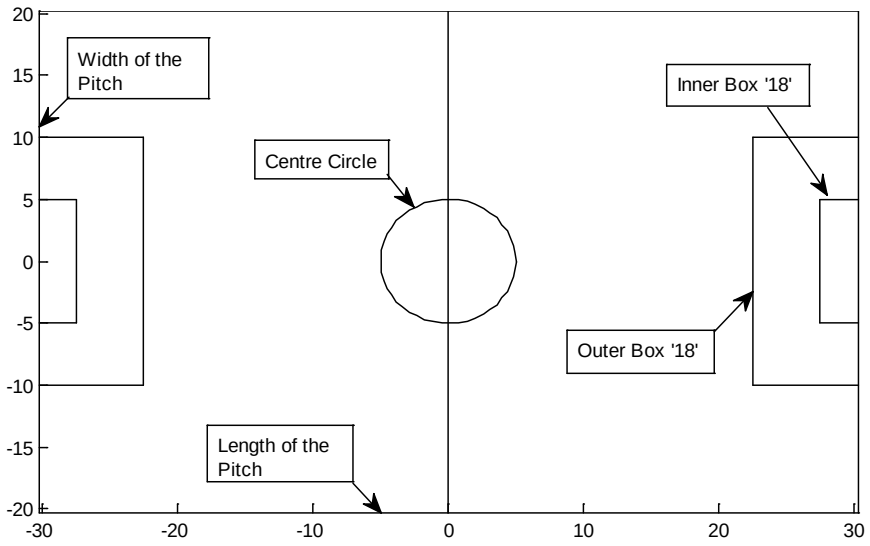

Figure 10.2 – Part label of the Football pitch layout model

testcir2.m

```
%draw circle code
%resolution of plot
t = linspace(0,2*pi,100000);

%assumed centre of the circle (cirX, cirY): sets at origin (0, 0)
cirX=30.25;
cirY=20.25;

%radius of the centre circle, 500mm=5m
r=5;

%circle dual equations
x = r*cos(t)+cirX;
y = r*sin(t)+cirY;

plot(x,y, 'Color','black');
%end of code
```

semiCircleBottomLeft.m

```
%draw circle code
%resolution of plot
t2 = linspace(2*pi, 3*pi/2,100000);
%assumed centre of the circle (cirX, cirY): sets at origin (0, 0)
cirX2=0;
cirY2=18.50;

%radius of the centre circle, 500mm=5m
r=5;

%circle dual equations
x2 = r*cos(t2)+cirX2;
y2 = r*sin(t2)+cirY2;

plot(x2, y2, 'Color', 'black')

%end of code
```

semiCircleBottomRight.m

```
%draw circle code
%resolution of plot
t3 = linspace(pi, 3*pi/2,100000);
%assumed centre of the circle (cirX, cirY): sets at origin (0, 0)
cirX3=60.50;
cirY3=18.50;

%radius of the centre circle, 500mm=5m
r=5;

%circle dual equations
x3 = r*cos(t3)+cirX3;
y3 = r*sin(t3)+cirY3;

plot(x3, y3, 'Color', 'black')

%end of code
```

semiCircleTopLeft.m

```
%draw circle code
%resolution of plot
t1 = linspace(0, pi/2,100000);
%assumed centre of the circle (cirX, cirY): sets at origin (0, 0)
cirX1=0;
cirY1=22.00;

%radius of the centre circle, 500mm=5m
r=5;

%circle dual equations
x1 = r*cos(t1)+cirX1;
y1 = r*sin(t1)+cirY1;

plot(x1, y1, 'Color', 'black')

%end of code
```

semiCircleTopRight.m

```
%draw circle code
%resolution of plot
t4 = linspace(pi/2, pi, 100000);
%assumed centre of the circle (cirX, cirY): sets at origin (0, 0)
cirX4=60.50;
cirY4=22.00;
%radius of the centre circle, 500mm=5m
r=5;
%circle dual equations
x4 = r*cos(t4)+cirX4;
y4 = r*sin(t4)+cirY4;
plot(x4, y4, 'Color', 'black')
%end of code
```

robotpatternUpdated.m

```
%refreshes figures for new ones
clear
close all

%activities needed on the robot field layout

%test plot sample
%-----------------------------------------------
%-----------------------------------------------
len=100;
robot=zeros(1,len);

%start position
X(1)=-40;
Y(1)=-40;

%move to cordinates
x_goal=0;
y_goal=0;

%
%just something to plot
%this will be for the actual robot movement path
%
for n=1:len
    robot(n)=sin(n)+10;
end
%

%plot robot movement
hold on
plot(robot, 'Color', 'red')

%plot the robot pitch layout
newFieldSpec
```

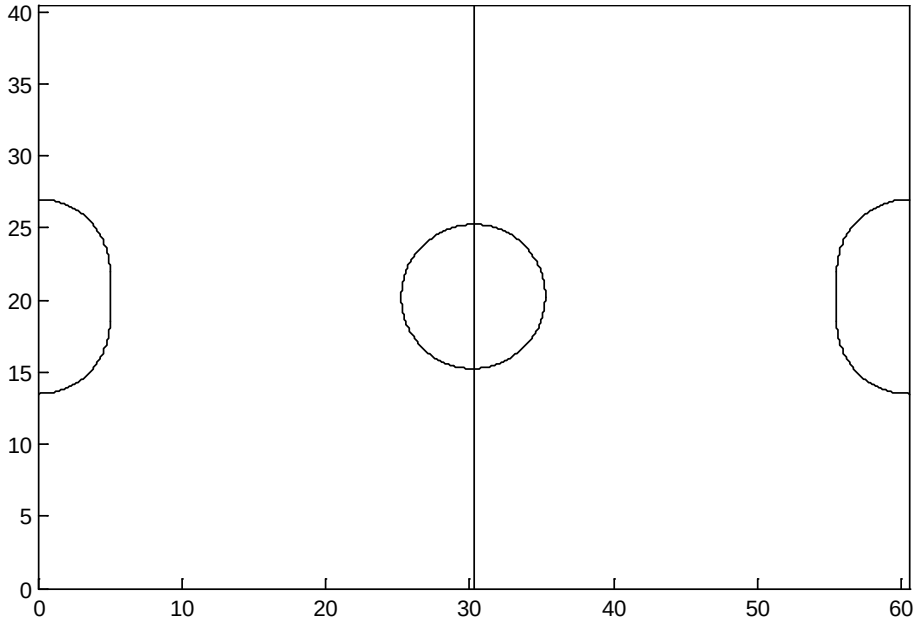

Figure 10.3 – Generated football pitch model using **`robotpatternUpdated.m`**

11 ROBOT 4-WHEEL MOTOR MODEL TRAJECTORY PLANNING

This part involves the cascaded arrangement of all the four wheels with connection to the corresponding system blocks affecting the overall performance of the robot path movement. The block arrangement used is as shown in figure 11.1 below:

After the necessary planning was done, the path simulation would be done on the football model developed with the MATLAB.

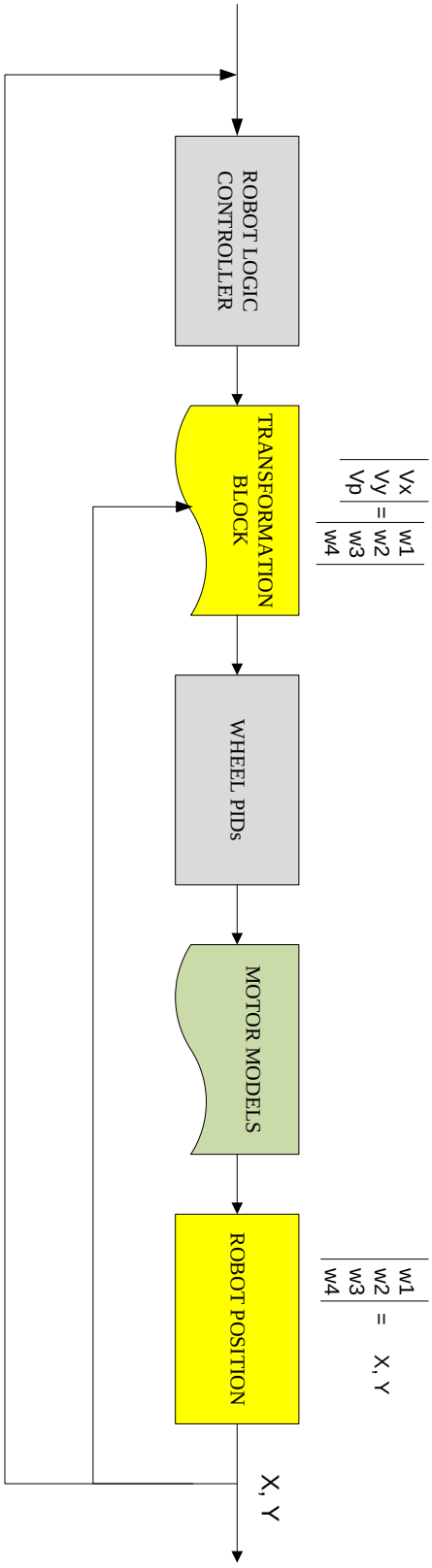

Figure 11.1 – Full robot implementation block

The robot wheels have the following wheel arrangement as shown in figure below figure 11.3 below: the wheels motors are in an asymmetrical arrangement; this is a prototype drawing from the figure 11.2. With radian evaluation, the angles p1, p2, p3 and p4 are related to the angle of the wheel axis – 53^{o}, 53^{o}, 45^{o}, and 45^{o}. The whole evaluations as regards this were done in the codes – `robotBlockpart.m, robotControllLogic.m, veloToWheel.m, wheelPIDs.m,` and `speedToXY.m` based on figure 11.1 implementation plan.

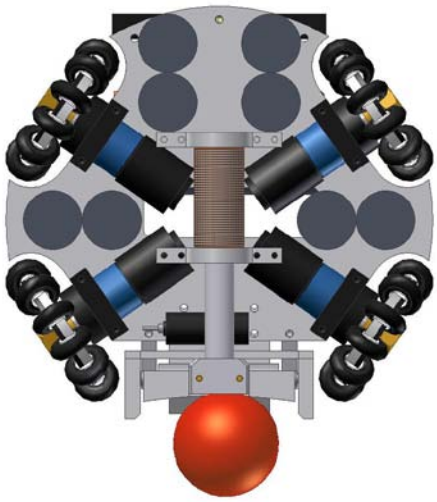

Figure 11.2 – An extract from "Omnidirectional control" [13]

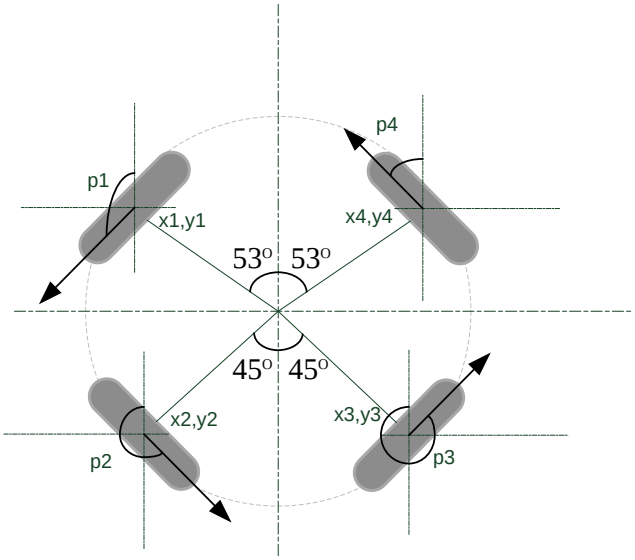

Figure 11.3 – Asymmetrical robot wheel arrangement based on the *Omnidirectional robot control*

robotTrajectoryPlan.m

```
%plots robot trajectory path
clear
close all
len=100;
robot=zeros(1,len);

X=zeros(1,100);Y=zeros(1,100);

%start position
X(1)=25;Y(1)=25;
a1(1)=0;a2(1)=0;a3(1)=0;a4(1)=0;
a1(2)=0;a2(2)=0;a3(2)=0;a4(2)=0;
a1(3)=0;a2(3)=0;a3(3)=0;a4(3)=0;

%move to cordinates
x_goal=0;y_goal=0;

n=1;k=3;
robotGain=0.00001;

%inlcudes motor constants
constants
% includes evaluated constants
evaluatedconstants

num = 1/Ke;
den = [tm*te tm 1];

%Ziegler-Nichols parameter computed
Kp = 11.327;    %Proportional gain
Ki = 1381.34;   %Integral gain
Kd = 0.0232;    %Derivative gain
% For the PID equation
numc = [Kd Kp Ki ];
denc = [1 0];

% convule "num with numc" and "den with demc"
numa = conv(num, numc);
dena = conv(den, denc);

sys = tf(numa,dena,1/1000);

integrationSums=[0, 0, 0, 0];

% robot block parts
robotBlockpart

%plot robot movement
hold on
plot(X,Y);

%plot the robot pitch layout
newFieldSpec
```

robotBlockpart.m

```
while n==1
    %done?
    %robot control logic - BLOCK 1
    [Vx(k),Vy(k),Vp(k)]=robotControlLogic(X(k-1),Y(k-
1),x_goal,y_goal,robotGain);

    %Done
    %transformation block  - BLOCK 2
    %transformation matrix from velocity vector to wheelspeeds
    [w1(k),w2(k),w3(k),w4(k)] = veloToWheel(Vx(k),Vy(k),Vp(k),X(k-
1),Y(k-1));

    %Done
    %Wheel PIDs - BLOCK 3
    %the idividual PID controllers for the wheels including
wheelmotor model
    temp=integrationSums;
    oldArray=[a1(k-1),a2(k-1),a3(k-1),a4(k-1)];
    oldoldArray=[a1(k-2),a2(k-2),a3(k-2),a4(k-2)];

[a1(k),a2(k),a3(k),a4(k),integrationSums]=wheelPIDs(w1(k),w2(k),w3
(k),w4(k),temp,oldArray,oldoldArray);

    %motor model

    %Done
    %robot position - BLOCK 4
    %converts actual wheel motor speed to robot X Y position
    [X(k),Y(k)]=speedToXY(a1(k),a2(k),a3(k),a4(k),X(k-1),Y(k-1));

    %check if close enough to the goal coordinates
    if abs(X(k)-x_goal)<0.1%% && abs(Y(k)-y_goal)<0.1
        n=0;
    end
    if abs(Y(k)-y_goal)<0.1%% && abs(Y(k)-y_goal)<0.1
        n=0;
    end

    if abs(X(k))>41
        n=0;
    end
    if abs(Y(k))>41
        n=0;
    end
    if k>99
        n=0;
    end
    %increment loop index
    k=k+1;
end
```

This will take the main part of the planning and trajectory simulation

robotControlLogic.m

```
function [Vx,Vy,Vp]=robotControlLogic(X,Y,x_goal,y_goal,k)

    Mag_x=x_goal-X;
    Mag_y=y_goal-Y;

    M=sqrt(Mag_x^2+Mag_y^2);

    Vx=k*Mag_x/M;
    Vy=k*Mag_y/M;
    Vp=0;

end
```

wheelPIDs.m

```
function
[a1,a2,a3,a4,intSums]=wheelPIDs(w1,w2,w3,w4,intSumsIn,y_old,y_oldo
ld)
    %actual PIDs here
    %Ziegler-Nichols parameter computed
    Kp = 11.327;    %Proportional gain
    Ki = 1381.34;   %Integral gain
    Kd = 0.0232;    %Derivative gain

    inputs=[w1, w2, w3, w4];

    %certainty problem
    for n=1:4
        PIDin=inputs(n);
        sumIn=intSumsIn(n);

        [y,sumOut]=myPID(PIDin,y_old(n),y_oldold(n),sumIn);

        intSums(n)=sumOut;
        outputs(n)=y;
    end

    a1=outputs(1);
    a2=outputs(2);
    a3=outputs(3);
    a4=outputs(4);
end
```

speedToXY.m

```
function [X,Y]=speedToXY(a1,a2,a3,a4,Xold,Yold)
%Calculate X Y position based on actual wheelspeeds since last
sample
% |W1|    |Vx|
% |W2| -> |Vy|
% |W3|    |Vp|
% |W4|

%Angle of each wheel in Rad, these angles does not change in this
simulation
p1=2.49582083; %143 deg
p2=3.92699082; %225 deg
p3=5.49778714; %315 deg
p4=0.645771823; %37 deg

%Co-ordinates of each wheel in Meter
[x1,x2,x3,x4,y1,y2,y3,y4]=wheelsXYfromXY(Xold,Yold,p1,p2,p3,p4);

  if 1

  %actual wheel speeds...
  W=[a1, a2, a3, a4];
  %transformation matrix
  A=[cos(p1),sin(p1),(-y1*cos(p1)+x1*sin(p1)),1;
       cos(p2),sin(p2),(-y2*cos(p2)+x2*sin(p2)),1;
       cos(p3),sin(p3),(-y3*cos(p3)+x3*sin(p3)),1;
       cos(p4),sin(p4),(-y4*cos(p4)+x4*sin(p4)),1];
  inversA=inv(A);
  else

  %actual wheel speeds…
  W=[a1, a2, a3];
  A=[cos(p1),sin(p1),(-y1*cos(p1)+x1*sin(p1));
       cos(p2),sin(p2),(-y2*cos(p2)+x2*sin(p2));
       cos(p3),sin(p3),(-y3*cos(p3)+x3*sin(p3));
       cos(p4),sin(p4),(-y4*cos(p4)+x4*sin(p4))];
  inversA=inv(A);
  end
  %use the inverse of A here since matrix division is not allowed

  B=W*inversA;

%??
X=Xold+B(1);
Y=Yold+B(2);
%rotation=B3(3); this is not needed if rotation is omitted
end
```

veloToWheel.m

```
function [w1,w2,w3,w4] = veloToWheel(Vx,Vy,Vp,X,Y)
%Calculate the individual wheelspeed based on the three component
%vector velocity of the robot
%|Vx|    |W1|
%|Vy| -> |W2|
%|Vp|    |W3|
%        |W4|
%Desired speed vectors

%Angle of each wheel in Rad
p1=2.49582083; %143 deg
p2=3.92699082; %225 deg
p3=5.49778714; %315 deg
p4=0.645771823; %37 deg

%Co-ordinates of each wheel in Meter
[x1,x2,x3,x4,y1,y2,y3,y4]=wheelsXYfromXY(X,Y,p1,p2,p3,p4);

if 0
    %Wheel 1
    x1=0.0677;y1=0.0511;

    %Wheel 2
    x2=-0.0599;y2=0.0596;

    %Wheel 3
    x3=-0.0599;y3=-0.0596;

    %Wheel 4
    x4=0.0677;y4=-0.0511;
end

%Transformation Matrix
A=[cos(p1),sin(p1),(-y1*cos(p1)+x1*sin(p1));
   cos(p2),sin(p2),(-y2*cos(p2)+x2*sin(p2));
   cos(p3),sin(p3),(-y3*cos(p3)+x3*sin(p3));
   cos(p4),sin(p4),(-y4*cos(p4)+x4*sin(p4));];

%3 component vector matrix
B=[Vx;Vy;Vp];
W=(A*B); %Matrix solution giving result for velocity of each wheel
w1=W(1);
w2=W(2);
w3=W(3);
w4=W(4);
end
```

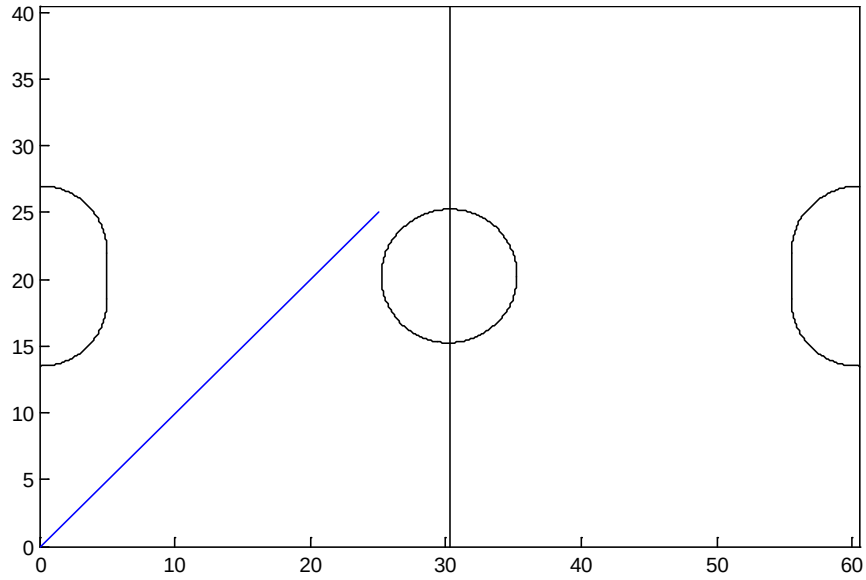

Figure 11.4 – The output of the robot path plotting

The blue line in figure 11.4 shows the planned path of the robot trajectory.

12 CONCLUSION, CHALLENGES AND RECOMMENDATION

12.1 Conclusion

In this work, the PID controller was used as a vital technical tool used in system modelling and control. It started with the analysis and reasons why an absolute précised control is important in drives particularly the BLDC motors and then the mathematical modelling. Also, the use of the MATLAB®/SIMULINK® to develop the robot football pitch model and the trajectory planning were additional parts to this work.

12.2 Challenges

Some of challenges faced include: the intensive study of MATLAB®/SIMULINK® and the various modelling techniques. More also, the knowledge of mathematical methods was needed to enhanced my modelling ability in this thesis as it required more mathematical skills. In additional, some of areas of control systems engineering had to be studied to have a blend of understanding in the areas of system stability. And one major challenging part was the aspect of model the path of the robot from one point to another. This part required some advanced mathematical skills which could not be implemented. A straight path was gotten in the trajectory simulation.

12.3 Recommendations – Possible improvement

This work could be improved by incorporating the hardware testing and possible laboratory testing. Also, to have a more précised PID parameters, new methods of PID tuning (the use of genetic algorithms) could be employed for optimal values. In addition to the use of PID controller, another instance of Single-Input-Single-Output (SISO) could be used under the MATLAB versatile toolbox.

More also, the real testing and program implementation of the BLDC motor could be harnessed by using the MATLAB®/SIMULINK® utilities and being able to

incorporate C-programming with the microcontroller. And more technical resources should be available to the student for proper execution of the work.

REFERENCES

[1] Siemens Training Education Program, STEP 2000 Series, *"Basics of DC drives and related products"*.

[2] Crouzet motor manuals, *"Some principles of DC motors"*.

[3] Microchip Technology Incorporated 2003, Padmaraja Yedamale, *"Brushless DC motor fundamentals"*.

[4] P. C. Sen. *Principles of Electric Machines and Power Electronics.* John Wiley & Sons, 1997.

[5] Carnegie-Mellon University and University of Michigan Online resources – *Control Tutorials for MATLAB: PID Tutorial*. Accessed 25th February 2009.

[6] Texas Instruments Incorporated. *DSP Solutions for BLDC Motors, 1997.*

[7] Åstrom, K and Hägglund, T (1994), *PID controllers: theory, design and tuning.* 2nd edition.

[8] Maxon EC motor, May 2008 edition, EC 45 flat ∅45 mm, brushless, 30 Watt Maxon flat motor.

[9] MATLAB/SIMULINK Documentations (Help file)

[10] Brian R Copeland , *The Design of PID Controllers using Ziegler Nichols Tuning,* 2008

[11] George W. Younkin, Electric Servo-motor equations and time constants.

[12] The *Laws of the F180 League 2009* – Small sized *robocup* league

[13] Raul Rojas, *Omnidirectional control*, Updated 18 May, 2005.

APPENDIX