

**Ministry of Higher Education
& Scientific Research
Ninevah University**



**College of Electronic Engineering
Department of
Systems and Control Engineering**

Solar energy Introduction

Submitted by:

Injad Amer Yaseen

Ahmed Farhan Mutlaq

Raad Abdalla Hassan

Supervised by:

Mr.Salam Asmer

Mr. Mohammed Nasrat

A project submitted as partial fulfillment for the requirements of the
degree of B.Sc. in Systems and Control Engineering

May - 2018

contents

CHAPTER 1 Introduction	5
1.1. Background.....	5
1.2. Objectives	5
CHAPTER 2 Literature view	6
CHAPTER 3 methodology.....	8
3.1. Principle of Buck-Boost converter	8
3.1.1 Mode 1.....	8
3.1.2 Mode2.....	10
3.2. Building model of transfers function for Buck-Boost converter.....	11
3.3. Implement the design in Simulink.....	12
3.4. Controller.....	13
3.4.1 PID controller	13
3.5. Tuning PID parameters using system identification.....	18
CHAPTER 4 Result and discussion	21
4.1. Open loop result	21
4.2. Closed loop result	23
CHAPTER 5 conclusion and future work	25
5.1. conclusion.....	25
5.2. future work	25

List of tables

Table 3-1 list of components	12
Table 3-2 PID parameters.....	21
Table 4-1 open loop components	22
Table 4-2 open loop response.....	23
Table 4-3 close loop response	24

List of figures

Figure 3-1: Mode 1 when switch ON.....	9
Figure 3-2: Mode 2 when switch OFF	10
Figure 3-3 open loop diagram	12
Figure 3-4 Block Diagram of a Process under Control System	13
Figure 3-5PID as the Control System:.....	14
Figure 3-6 The P-Control	15
Figure 3-7 PID Control Block diagram	16
Figure 3-8 Response to Unit step of a D controller.....	17
Figure 3-9 Response to Unit step of an I control.....	18
Figure 3-10 Response to Unit step of a P controller	18
Figure 3-11 run simulation	19
Figure 3-12 plant can't be linearized	19
Figure 3-13 new plant identification	20
Figure 3-14 desired response setup	21
Figure 5-1	25

Abstract

Solar panels are widely used in the world and give output voltage proportional to the intensity of the sunlight, so the output voltage is varying with respect to the day time (sun position in the sky) and not useful if not regulated to a constant value.

A buck-boost converter is a component found in solar panels which is used to regulate the voltage output produced by these solar panels. This converter can be adjusted to produce voltage with a larger or smaller value than the initial voltage. This experiment aims to create a voltage output at a constant value of 12 V, with a range of voltage input of 10 to 50 V. The voltage value produced by the converter is controlled by the pulse width produced by the PWM (Pulse Width Modulation) generator. The pulse width produced by the PWM generator is controlled by a PID controller, the controller is designed by MATLAB Simulink.

CHAPTER 1 *Introduction*

1.1. *Background*

Solar panels are widely used as an environmentally friendly electricity supply. It has the potential to cover all human energy needs [13], Photovoltaic (PV) panels are widely used all over the world which are either distributed on roof tops or installed in large power plants. It is very popular since it is simple to install and relatively cost effective. Unfortunately, a problem is often found, where the voltage output produced is not constant, mostly it needs to be stored in storing cells (Batteries) so we have to regulate the voltage to a specified value and that process is done by DC-DC converter as a voltage regulator.

Voltage regulator methodology is a constant dc voltage despite changes in line voltage, load and temperature. Voltage regulator can be classified into linear regulators and switching-mode regulators. Some drawbacks of linear regulators are poor efficiency, which also leads to excess heat dissipation and it is impossible to generate voltages higher than the supply voltage. Switching mode regulators can be separated into the following categories: Pulse-Width Modulated (PWM) dc-dc regulators, resonant dc-dc converters, and Switched-capacitor voltage regulators. The PWM dc-dc regulators can be divided into three important topologies: buck converter, boost converter, and buck-boost converter. The buck-boost converter is chosen for analysis in this project.

A buck-boost converter is a component found in solar panels which is used to regulate the voltage output produced by these solar panels. This converter can be adjusted to produce voltage with a larger or smaller value than the initial voltage and the polarity of voltage output will be the opposite of the voltage input's polarity. This experiment aims to create a voltage output at a constant value with a range of voltage input. The voltage value produced by the converter is controlled by the pulse width produced by the PWM (Pulse Width Modulation) generator. The PWM dc-dc buck-boost converter reduces and increases dc voltage from one level to another.

1.2. *Objectives*

The objectives of this project are as follows:

1. Modeling and simulation the dc-dc buck-boost converter for open-loop.
2. Design a control circuit for the buck-boost converter.
3. Modeling and simulation the dc-dc buck-boost converter for closed-loop.

CHAPTER 2 *Literature view*

Currently, there are plenty of technological applications that utilize natural, environmental-friendly sources of energy. However, a disadvantage often found in natural energy sources is that the intensity produced is uncertain. This occurrence is also found in solar panels, wherein the light intensity that enters is not always equal. Light intensity may be affected by various factors such as ones on gloomy or sunny weathers. This irregularity on light intensity leads to deviation of voltage output produced by the solar panel. Thus, with the use of buck-boost converters, the amount of output voltage may be set to higher or lower than the input voltage, enabling us to maintain the desired output voltage.

In order to design the control system, it is necessary to have an exact model of buck-boost converter. Several academic studies investigating Dc converters have been carried out from the early twentieth century. Dinniyah et al. designed a buck-boost converter for solar panels [1]. Xuelian and Qiang [2] also put forward the transfer function model of buck-boost converter by the state-space average method. The open loop transfer function model of uncompensated system was deduced according to the mathematic model of the buck boost converter. Also, the controller was designed according to frequency domain. In this paper, the authors concluded that this control system has the advantages of small overshoot and short settling time. It can also improve control system's real time property and anti-interference ability.

To obtain high performance control of a dc-dc converter, a good model of the converter is needed. System modeling is probably the most important phase in any form of system control design work. Analytic models were derived which will be utilized latter for control. In 2005, Johansson and Bengt [3] derived a transfer function for the buck, boost, and buck-boost converters operated in continuous conduction mode. A similar work was conducted by Priyadarshini et al. [4] to present open loop behavior of buck boost dc-dc converter operated under Continuous conduction mode (CCM). Then a PID controller was designed such that any input variations produce a constant output voltage. In the same vein, Yang and Shun in 2011 [5] introduced a module along with the controller action for the buck converter. In this paper modeling, modeling and control of a simple DC-DC Ericsson BMR 450 series buck converter is presented. For the given converter, analyzing the disturbance properties of a second order buck converter controller by a polynomial controller. The project is performed in

MATLAB and Simulink. The controller properties are evaluated for measurement noise and for parameter changes.

The simulation of the mathematical model is usually implemented to verify the behavior of the derived model. MATLAB Simulink software package can be advantageously used to simulate power converters. In their paper, Ali Khajezadeh et al [6] attempt to simulate all basic non-isolated power converters. The design of power electronic converter circuit with the use of closed loop scheme can easily be done with the help of state equations and MATLAB Simulink. Similarly, Priyadarshini's approach [4] was achieved with the aid of MATLAB Simulink. Likewise, Badr and Munaf in 2014 [7] utilized MATLAB Simulink environment to validate the response of the controlled model. The obtained results of the simulation show that the accuracy and the precision of the employed model to meet the desired voltage of the solenoid coil.

The best way to maximize the production of solar energy systems is implementing an efficient control system as well as the effectiveness of the method of the control in finding maximum power point. The voltage value produced by the converter is usually controlled by the pulse width produced by the PWM (Pulse Width Modulation) generator. The pulse width produced by the PWM generator is controlled by a microcontroller. The microcontroller is programmed to control the pulse width and frequency produced. PID controller has been widely applied in DC converters. Dinniyah et al. designed a digital PID controller scheme that was implemented in Arduino microcontroller and simulated in Proteus in order to keep a constant output voltage [1]. In the same field, Abdessamad and others presented a Comparative study of analog and digital Controller on Buck-Boost converter [8]. This paper reveals comparative performance between Analog and digital controller on DC/DC buck-boost converter. The design of power electronic converter circuit with the use of closed loop scheme needs modeling and then simulating the converter using the modeled equations. Another important aspect of the subject is to find the maximum power point. In his paper, Mahapatro [9] in 2013, presented the utilization of a buck-boost converter for control of photovoltaic power using Maximum Power Point Tracking (MPPT) control mechanism. The main aim of the project was to obtain the maximum possible amount of energy.

CHAPTER 3 *methodology*

3.1. Principle of Buck-Boost converter

In order to design the control system, it is necessary to have an exact model of buck-boost converter. The open loop transfer function model of uncompensated system is deduced according to the mathematic model of the buck-boost converter.

The circuit of Buck-Boost converter operation can be divided into two distinctive modes[10] :

- discontinuous conduction mode (DCM).
- continuous conduction mode (CCM).

In our system we focused on (CCM). The CCM is divided into two modes during one period of switching:

- Mode1
- Mode2

The output voltage is determined by Eq.2. According to Eq.2, it allows the output voltage to be lower or higher than the input voltage, which is determined by the duty cycle. When D is less than 0.5, output voltages will less than input voltage. Otherwise, output voltages will higher than input voltage

$$V_o = \frac{D}{1-D} \times V_i \quad (2)$$

Where D is duty cycle (switch on time in one period).

3.1.1 Mode 1

During mode1 1, the diode V_d is reversed biased and transistor Q is turn on. The input current flows into inductor L and transistor Q. The inductor current will

raise and the Inductor L will store energy. At the same time, capacitance C supplies the load R . Its equivalent circuit is shown as Fig.1. As shown in Fig.1, i_L and v_c are

the inductor current and the capacitor voltage respectively. R_{on} is on-resistance of transistor Q. r_L is

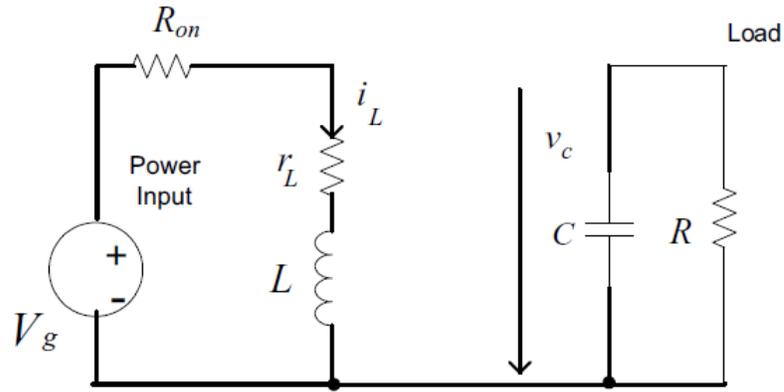


Figure 3-1: Mode 1 when switch ON

on-resistance of the Inductor L .

we can derive, i_L and v_c by applying Kirchhoff's Voltage Law (KVL) as following:

$$\frac{di}{dt} = -i \frac{(R_{on} + r_L)}{L} + \frac{V_i}{L}$$

$$\frac{dv_c}{dt} = -\frac{v_c}{RC}$$

Because R_{on} and r_L are very small, they can be ignored[10].

When written in state-space form, these equations become as Eq.5.

$$\mathbf{x} = \mathbf{A1}\dot{\mathbf{x}} + \mathbf{B1}V_i$$

$$\mathbf{y} = \mathbf{C1}\mathbf{x}$$

$$\mathbf{A1} = \begin{bmatrix} 0 & 0 \\ 0 & \frac{-1}{RC} \end{bmatrix}, \quad \mathbf{B1} = \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix}, \quad \mathbf{C1} = [0 \ 1]$$

$$\mathbf{x} = \begin{bmatrix} i \\ v_c \end{bmatrix} \quad \mathbf{y} = V_o = v_c$$

3.1.2 Mode2

During mode 2, transistor Q turns off and the current, which was flowing through inductor L, will flow through L, C, V_D and the load R. At the same time, the

inductor current will fall until the transistor Q is switched off. Its equivalent circuit is shown as Fig.2

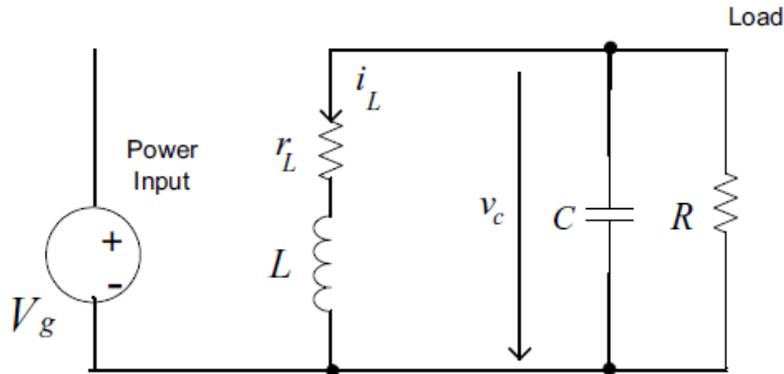


Figure 3-2: Mode 2 when switch OFF

Applying KVL to the circuit, we get the following:

$$\frac{di_L}{dt} = \frac{v_c}{L}$$

$$\frac{dv_c}{dt} = \frac{-i}{C} - \frac{v_c}{RC}$$

Now taking state-space model:

$$\dot{x} = A_2 x + B_2 V_i$$

$$y = C_2 x$$

$$A_2 = \begin{bmatrix} 0 & \frac{1}{L} \\ \frac{-1}{C} & \frac{-1}{RC} \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C_2 = [0 \ 1]$$

$$x = \begin{bmatrix} i \\ v_c \end{bmatrix} \quad y = V_o = v_c$$

3.2. Building model of transfers function for Buck-Boost converter

from both modes (mode1 and mode 2) average matrices are taken as following:

The averaged matrix A is as Eq.

$$\mathbf{A} = \mathbf{D} * \mathbf{A1} + (\mathbf{1} - \mathbf{D})\mathbf{A2}$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \frac{1-D}{L} \\ -\frac{1-D}{C} & \frac{-1}{RC} \end{bmatrix}$$

In a similar manner, the averaged matrices B and C are evaluated, with the following results:

$$\mathbf{B} = \mathbf{DB1} + (\mathbf{1} - \mathbf{D})\mathbf{B2} = \mathbf{DB1}$$

$$\mathbf{B} = \begin{bmatrix} \frac{D}{L} \\ \mathbf{0} \end{bmatrix}$$

$$\mathbf{C} = \mathbf{DC1} + (\mathbf{1} - \mathbf{D})\mathbf{C2} = \mathbf{C1} = \mathbf{C2}$$

$$\mathbf{C} = [\mathbf{0} \ \mathbf{1}] \quad (8)$$

Taking Laplace transform to find t.f of output to input:

$$G(s) = \mathbf{C}[\mathbf{SI} - \mathbf{A}]^{-1}\mathbf{B} + \mathbf{D}$$

Takin term $[\mathbf{SI} - \mathbf{A}]^{-1}$ to simplify:

$$= \left[\begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} - \begin{bmatrix} \mathbf{0} & \frac{1-D}{L} \\ -\frac{1-D}{C} & \frac{-1}{RC} \end{bmatrix} \right]^{-1} = \left[\begin{bmatrix} \mathbf{S} & \frac{D-1}{L} \\ \frac{1-D}{C} & \mathbf{S} + \frac{1}{RC} \end{bmatrix} \right]^{-1} = \frac{adj}{det} = \frac{\begin{bmatrix} \mathbf{S} + \frac{1}{RC} & \frac{1-D}{L} \\ \frac{D-1}{C} & \mathbf{S} \end{bmatrix}}{\mathbf{S}^2 + \frac{\mathbf{S}}{RC} + \frac{D^2 - 2D + 1}{LC}}$$

$$\mathbf{C}[\mathbf{SI} - \mathbf{A}]^{-1} = [\mathbf{0} \ \mathbf{1}] * \frac{\begin{bmatrix} \mathbf{S} + \frac{1}{RC} & \frac{1-D}{L} \\ \frac{D-1}{C} & \mathbf{S} \end{bmatrix}}{\mathbf{S}^2 + \frac{\mathbf{S}}{RC} + \frac{D^2 - 2D + 1}{LC}} = \frac{\begin{bmatrix} \frac{D-1}{C} & \mathbf{S} \end{bmatrix}}{\mathbf{S}^2 + \frac{\mathbf{S}}{RC} + \frac{D^2 - 2D + 1}{LC}}$$

$$C[SI - A]^{-1}B = \frac{\begin{bmatrix} \frac{D-1}{C} & S \end{bmatrix}}{S^2 + \frac{S}{RC} + \frac{D^2 - 2D + 1}{LC}} \begin{bmatrix} D \\ L \\ 0 \end{bmatrix} = \frac{\frac{D^2 - D}{LC}}{S^2 + \frac{S}{RC} + \frac{D^2 - 2D + 1}{LC}}$$

Simplifying: $\frac{D^2 - D}{LCS^2 + \frac{LS}{R} + D^2 - 2D + 1} \xrightarrow{\text{yields}} G_{vi} = \frac{D(D-1)}{LCS^2 + \frac{L}{R}S + (1-D)^2}$

3.3. Implement the design in Simulink

We have used MATLAB Simulink to implement the design and to see the the response in both open loop and closed loop

In simulation we used values of components shown in table[11]:

component	value
Inductor L	2200 uH
Capacitor C	8200 uF
Resistor R	5.1 Ω

Table 3-1 list of components

The model diagram is shown in fig.3 below as open loop design:

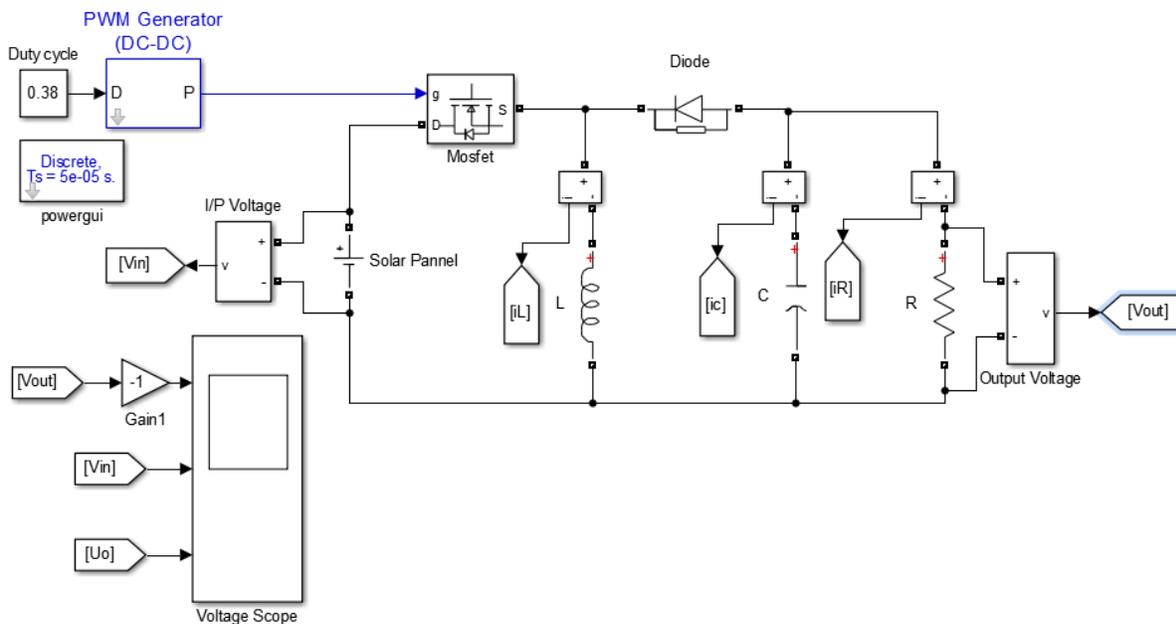


Figure 3-3 open loop diagram

3.4. Controller

What is a control system? Why do we need it?

- Open Loop and Closed loop Systems
- Examples of Control Systems
- What is a setpoint?
- What is Process Variable
- Control Output

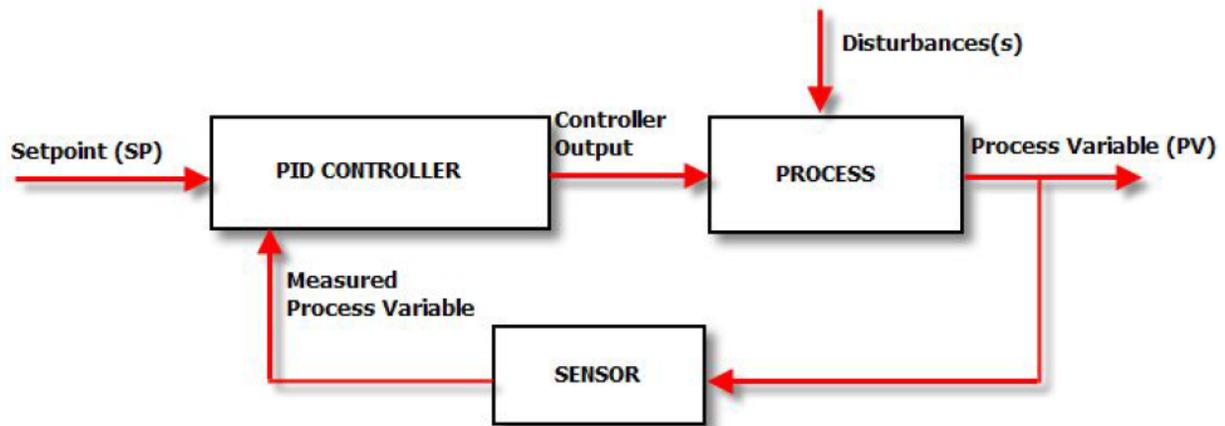


Figure 3-4 Block Diagram of a Process under Control System

3.4.1 PID controller

A proportional–integral–derivative controller (PID controller) is a control loop feedback mechanism. As the name suggests, PID algorithm consists of three basic coefficients: proportional, integral and derivative which are varied to get optimal response.

PID as the Control System:

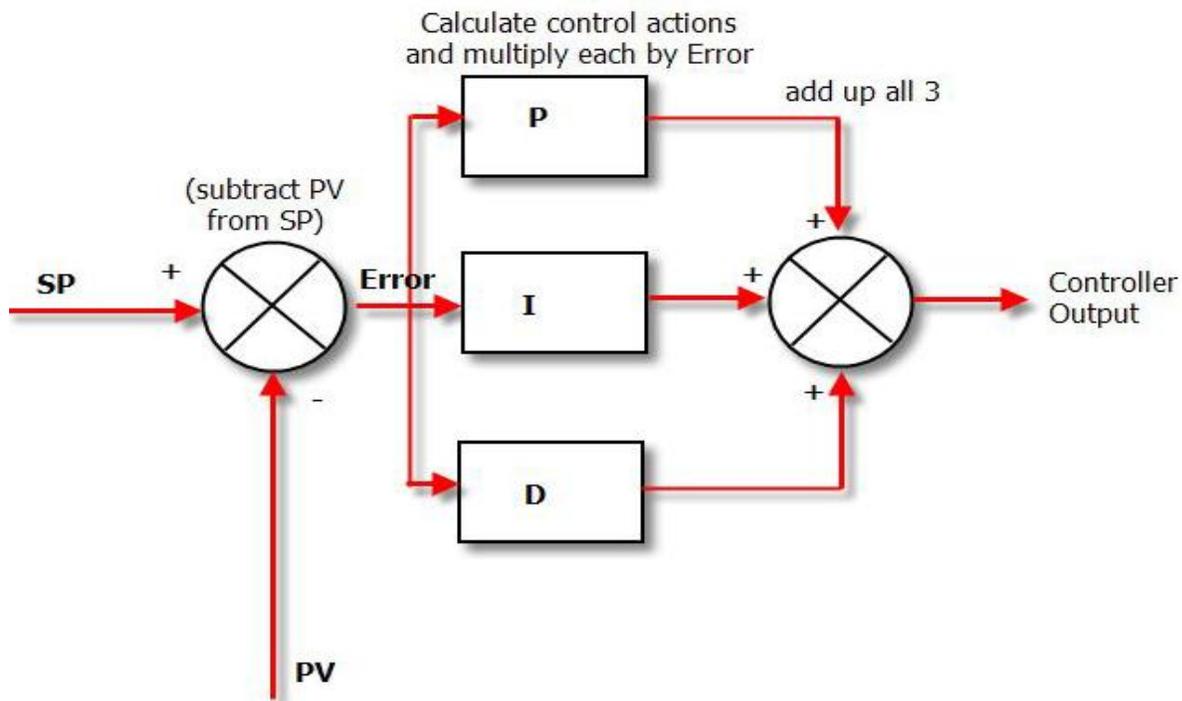


Figure 3-5PID as the Control System:

How does PID work?

The entire idea of this algorithm revolves around manipulating the error. The error as is evident is the difference between the Process Variable and the Setpoint.

$$\text{ERROR} = \text{PV} - \text{SP}$$

These 3 modes are used in different combinations:

- P – Sometimes used
- PI - Most often used
- PID – Sometimes used
- PD – Very rare, useful for controlling servomotors

The P-Control

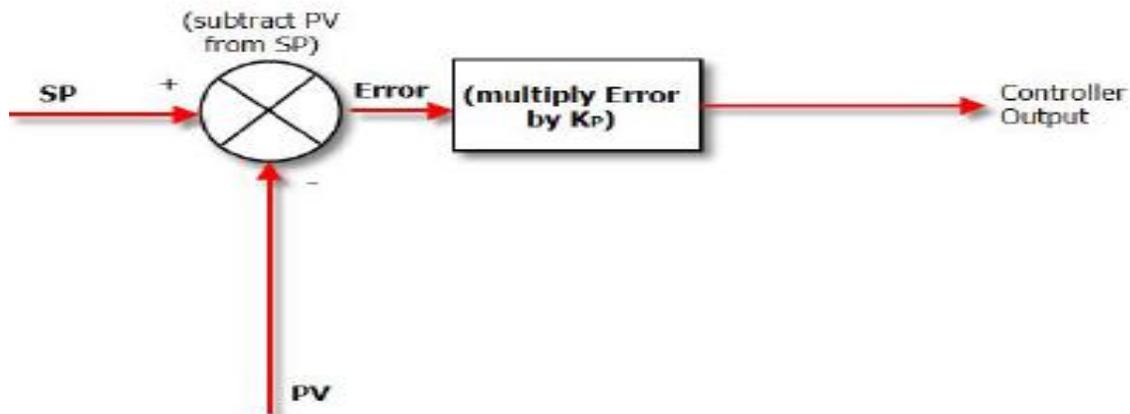


Figure 3-6 The P-Control

In Proportional Only mode, the controller simply multiplies the Error by the Proportional Gain (K_p) to get the controller output. The Proportional Gain is the setting that we tune to get our desired performance from a “P only” controller.

The proportionality constant used for P-Control is K_p .

Drawbacks of P-Control

- Too high a value of K_p will lead to the oscillation of PV.
- Also, the P-controller tends to generate an offset value.
- Proportional controllers also increase the maximum overshoot of the system

The PI-control

$$\text{Controller Output(COI)} = K_I \int e \, d\tau$$

$$\text{CO} = \text{COP} + \text{COI}$$

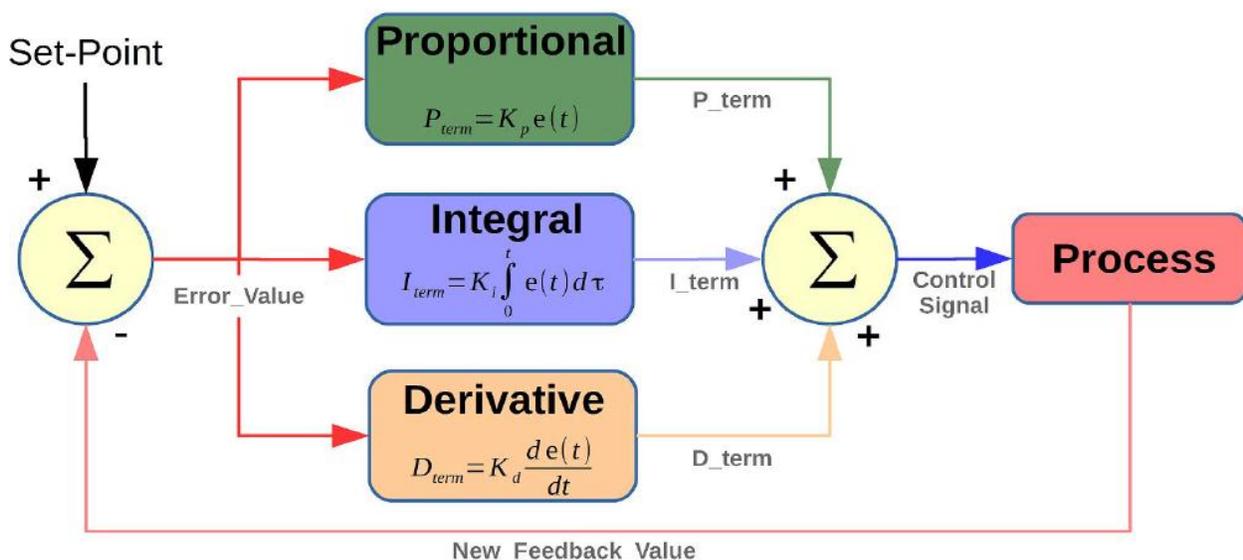
$$= K_P e + \left(\int e \, d\tau \right) / \tau_I$$

$$= K_P \left(e + \left(\int e \, d\tau \right) / \tau_I N \right)$$

As $e(t)$ grows or shrinks, the amount added to CO grows or shrinks immediately and proportionately. The past history and current trajectory of the controller error have no influence on the proportional term computation. Integral Action Eliminates Offset.

PID Control - Best of Everything

The proportional corrects instances of error, the integral corrects accumulation of error, and the derivative corrects present error versus error the last time it was checked. The effect of the derivative is to counteract the overshoot caused by P and I. When the error is large, the P and the I will push the controller output. This controller response makes error change quickly, which in turn causes the derivative to more aggressively counteract the P and the I.



Tuning a PID Controller

Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to optimum values for a target response.

- Bump Test and Modelling (Manual Control)
- Tuning
- Simulation

Too High K_P will lead to oscillation in values and will tend to generate an offset K_I will counteract the offset. Higher Value of K_I implies that the Setpoint will reach the PV too fast If this action is very fast, the process variable is prone to be unsteady. K_D keeps this under control.

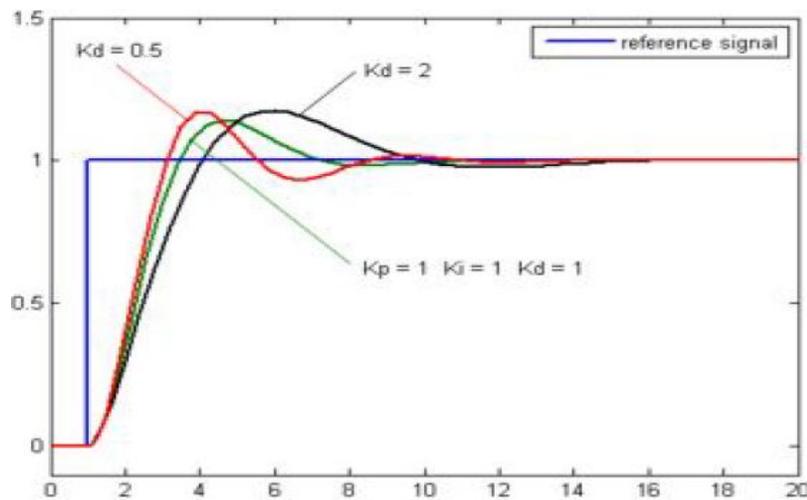


Figure 3-8 Response to Unit step of a D controller

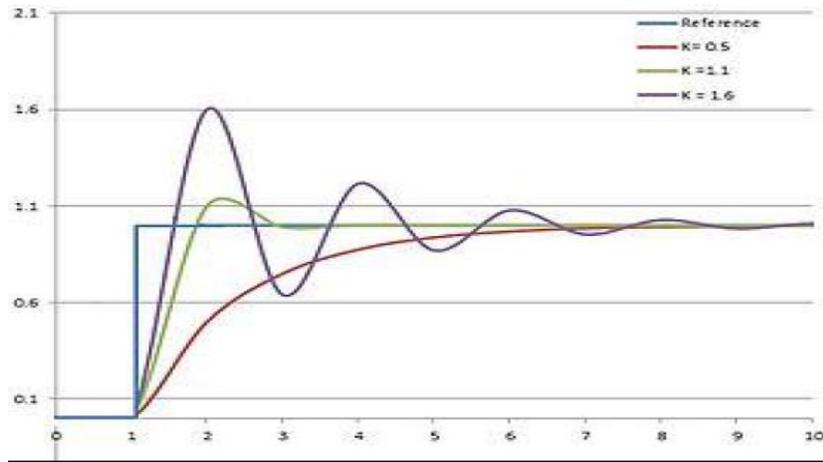


Figure 3-9 Response to Unit step of an I control

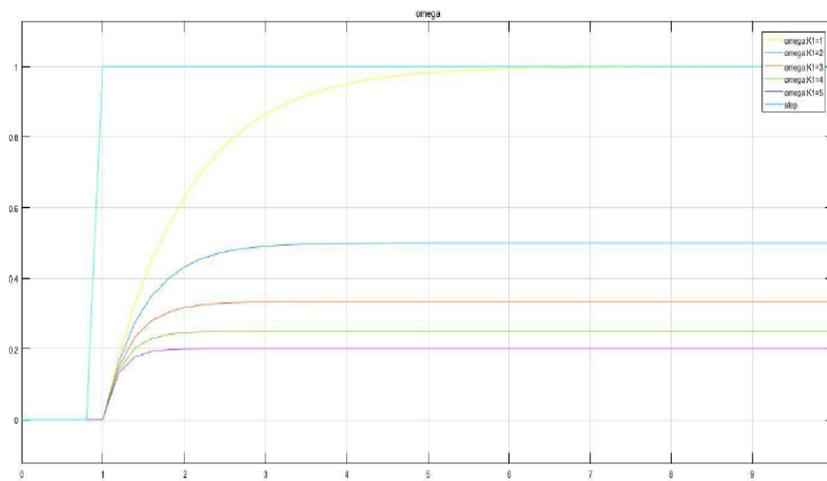


Figure 3-10 Response to Unit step of a P controller

3.5. Tuning PID parameters using system identification

The system required PID controller, there are many ways to tune parameters (K_p , K_i , K_d) because the exist of digital switch (MOSFET), the system is nonlinear, so it is required to be linear, linearization is done automatically in Simulink by get the i/o data from simulation data and identify new plant to apply the parameters on which, this operation called system identification process.

The following steps show how tuning done;

- 1- Run the plant once to store data required (parameters are chosen default) and the input voltage chosen near the desired operation point ($v_i = 10v$) as shown in the following figure:

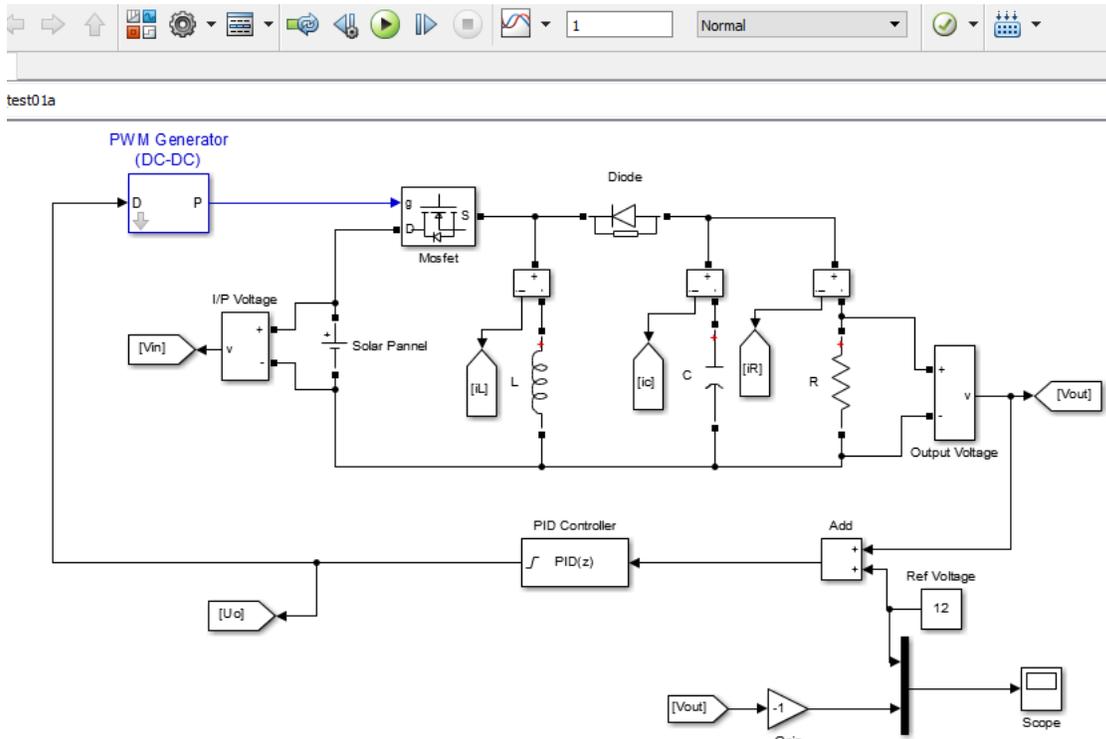


Figure 3-11 run simulation

- Open PID block and then choose (Tune...), a message shows that the plant can't be linearized as shown in fig.12 below:

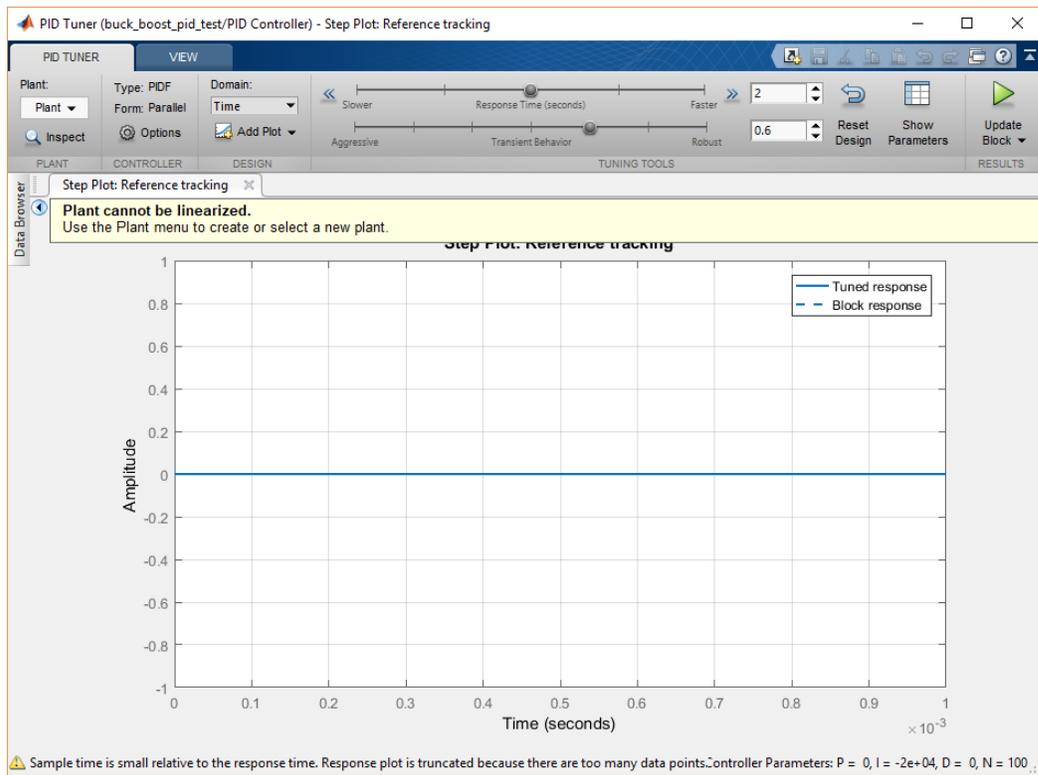


Figure 3-12 plant can't be linearized

That problem is solved in the next step:

- 3- Choose plant > I/O data from (Simulate data) to get I/O data required for estimation of the response (the response is assumed as a new linear system) , fig.12 below shows how to choose:

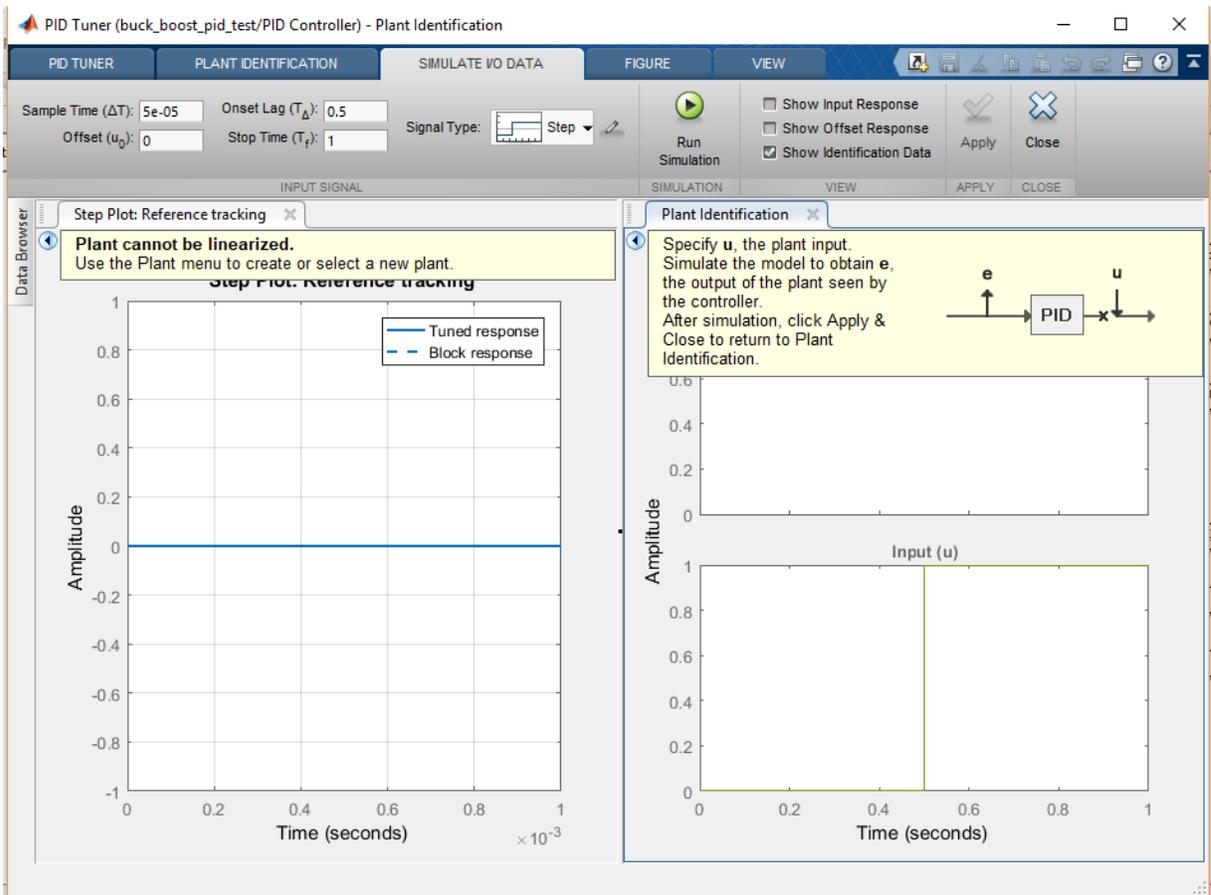


Figure 3-13 new plant identification

- 4- after the new plant has been ready to set the desired response by change of (offset u_0 , signal Type: Step, amplitude A) and change in required response amplitude ,response time, transient behavior as shown in fig.14 below :

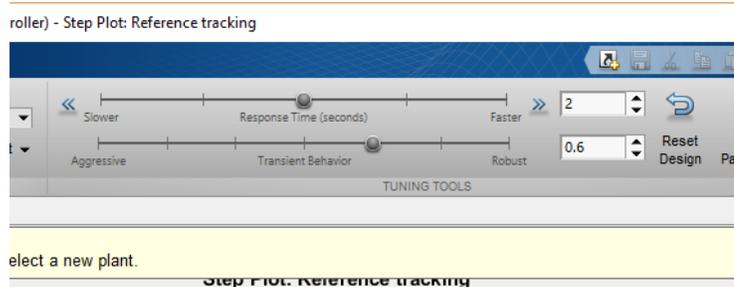


Figure 3-14 desired response setup

5- after choosing the satisfied response, apply the parameters to the plant, there will be new values of (Kp, Ki, Kd) , the optimum values we got are shown in table.2:

PID parameter	value
Kp	6.67537838450162e-05
Ki	2.67015135380065
Kd	0.0001

Table 3-2 PID parameters

CHAPTER 4 *Result and discussion*

In experimental work using Simulink the response was captured for many input voltages in both open loop and closed loop and easy to see the difference between open loop and closed loop response.

4.1. *Open loop result*

In open loop response, we have desired V_o and actual V_{in} and can calculate duty cycle by equation (4.1):

$$D = \frac{\frac{V_o}{V_{in}}}{1 + \frac{V_o}{V_{in}}} \dots\dots\dots (4.1)$$

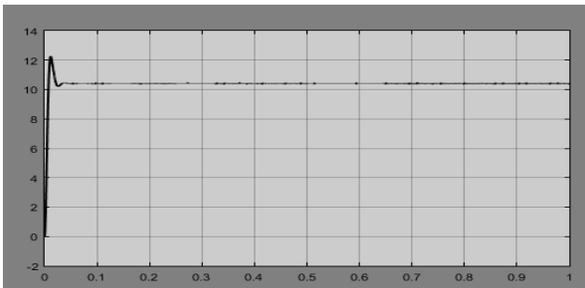
Many values of V_{in} were applied and the required duty cycle was calculated and changed manually as the following table.3:

V_{in}	D	actual V_o	V_{out}
8	0.6	10.5	12
10	0.5455	10.5	12
12	0.5	10.7	12
14	0.4615	10.8	12
22	0.3529	10.9	12
28	0.3	11	12
32	0.2727	10.95	12
36	0.25	11.1	12
40	0.2308	11.1	12
45	0.2105	11.1	12
50	0.1935	11.1	12

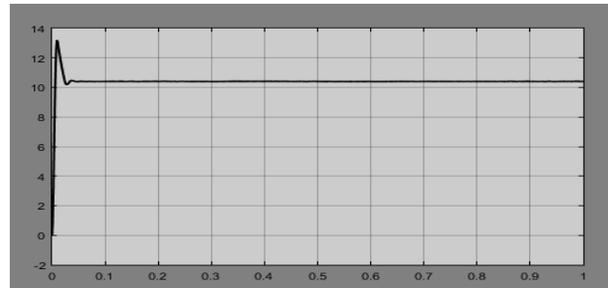
Table 4-1 open loop components

Each response in above table is captured in graph, the following graph shows how the open loop response and show the unsatisfied overshoot and steady state error:

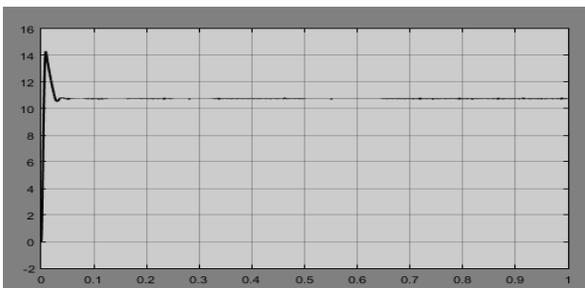
Table 4.2 show the charts of open loop response



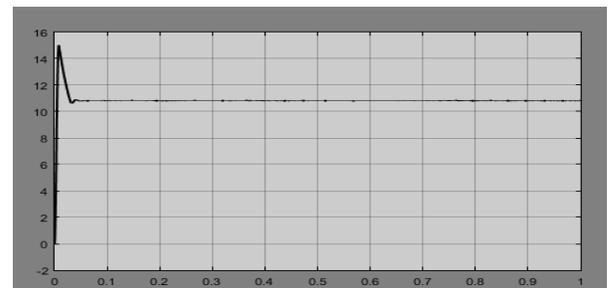
Fig(a), $V_{in}=8, D=0.6$



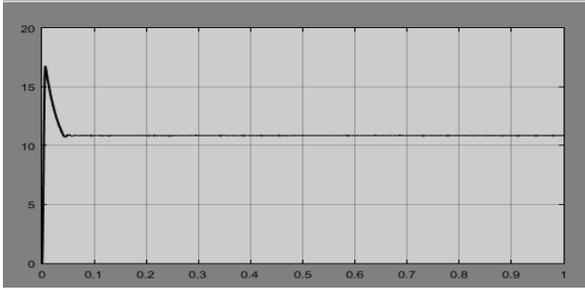
Fig(b), $V_{in}=10, D=0.54$



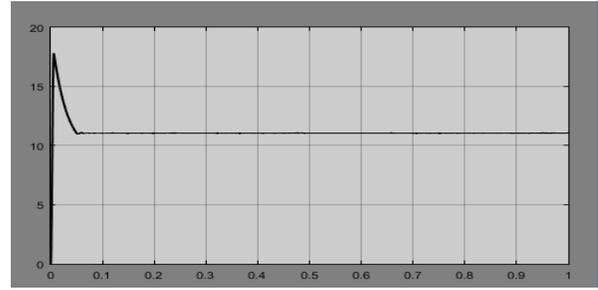
Fig(c), $V_{in}=12, D=0.5$



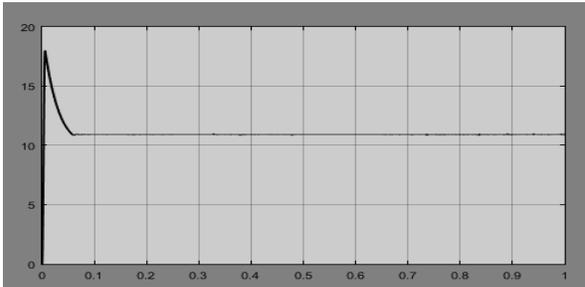
Fig(d), $V_{in}=14, D=0.46$



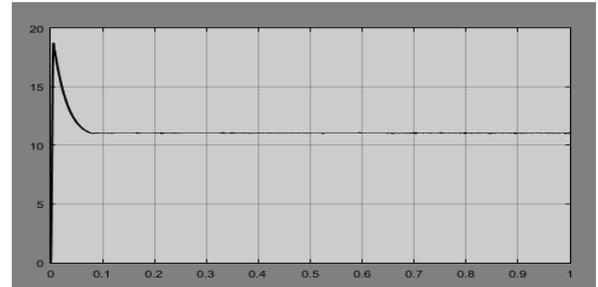
Fig(f), $V_{in}=22, D=0.35$



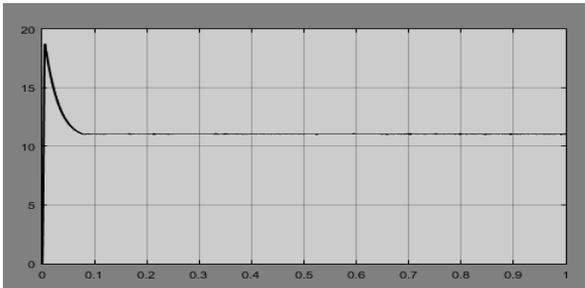
Fig(g), $V_{in}=28, D=0.3$



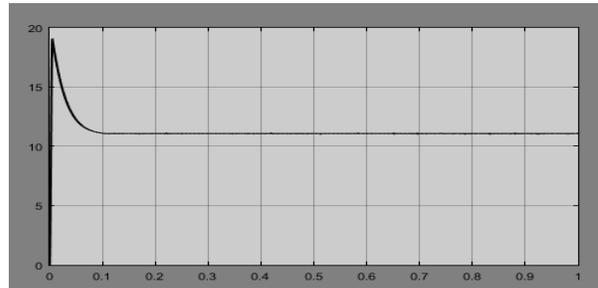
Fig(h), $V_{in}=32, D=0.27$



Fig(i), $V_{in}=36, D=0.5$



Fig(j), $V_{in}=40, D=0.23$



Fig(k), $V_{in}=45, D=0.21$

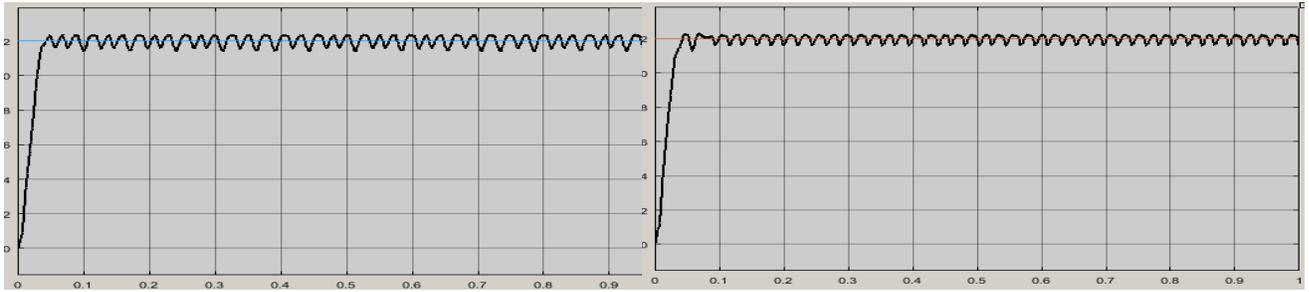
Table 4-2 open loop response

As seen in the open loop response, the response was not good enough to apply in physical device, so that the controller was designed (to reduce the overshoot and eliminate steady state error)

4.2. Closed loop result

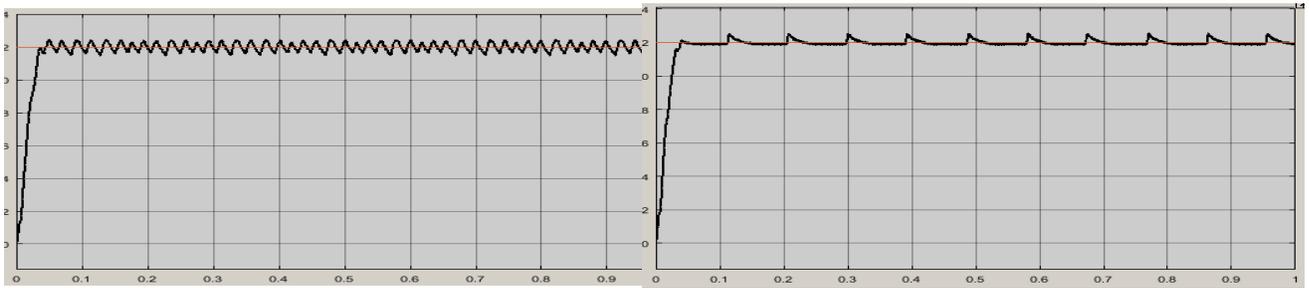
After the design of PID controller, many input voltages were applied in the experimental work, when change the input value there is no need to change duty cycle manually because it is regulated automatically by PID controller.

As shown in table 4.3, there is no steady state error but accepted ripple when $V_i \leq 12v$



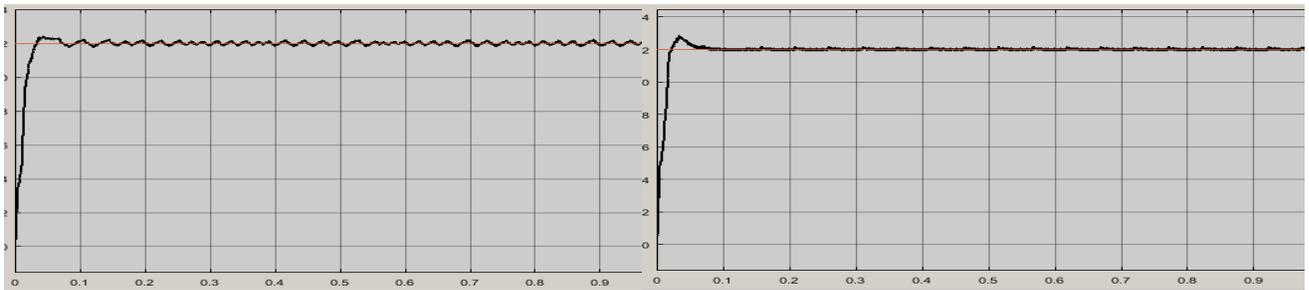
Fig(a) $V_i=8$

Fig(b) $V_i=10$



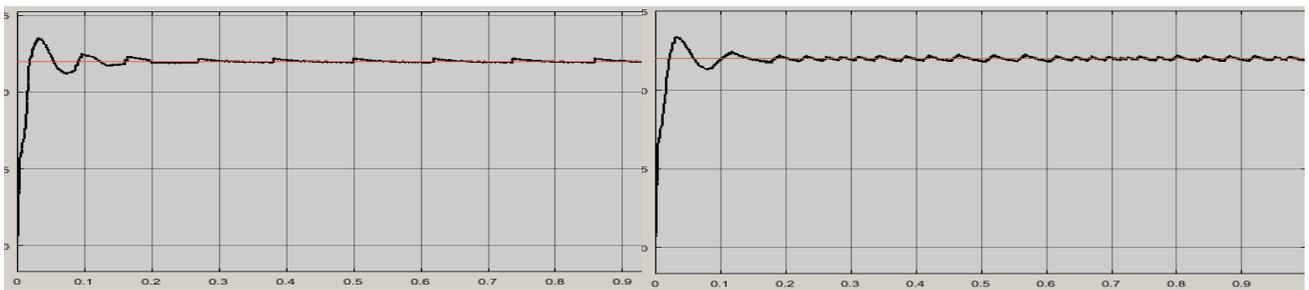
Fig(c) $V_i=12$

Fig(d) $V_i=14$



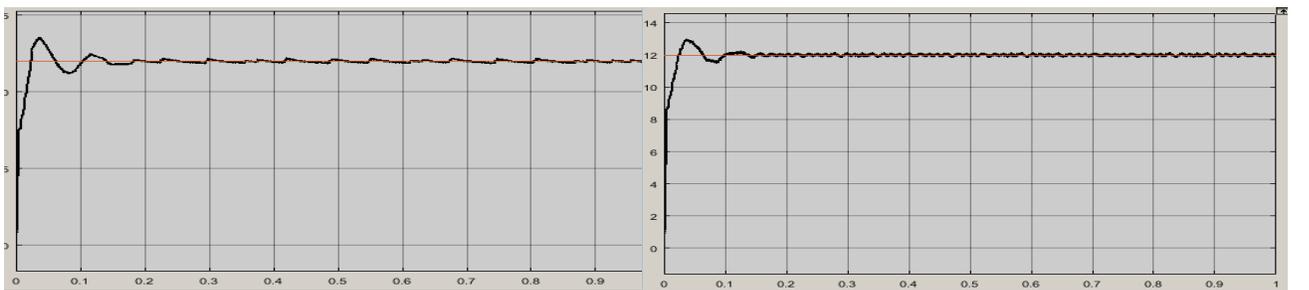
Fig(e) $V_i=22$

Fig(f) $V_i=28$



Fig(g) $V_i=32$

Fig(h) $V_i=36$



Fig(i) $V_i=40$

Fig(j) $V_i=45$

Table 4-3 close loop response

CHAPTER 5 *conclusion and future work*

5.1. *conclusion*

as the graphs show, we conclude the importance of the control systems in different devices and process, in open loop simulation there is error about 20% of the desired voltage and that value is not accepted and my cause fault to the system, and the overshoot was about 50% when high voltage applied (30-40), that over shoot may cause damage to the physical components of the system.

5.2. *future work*

many things are considered as future work, like the practical circuit we didn't finish it because a problem in the design of gate driver of the MOSFET, because V_{gs} need to be applied between gate and source, the source isn't connected to the ground of the circuit, so it needed a virtual ground, can be implemented by using two sync MOSFETs with ic IR2108 and it is not available in market.

One of future work consideration is implementation of (FUZZY PID CONTROLLER) as a program executed in an (sync. Buck converter v1.8 made in Iraq) fig (5.1) it is buck converter and can be configured as buck-boost converter, int is embedded system based on STM32 microcontroller.

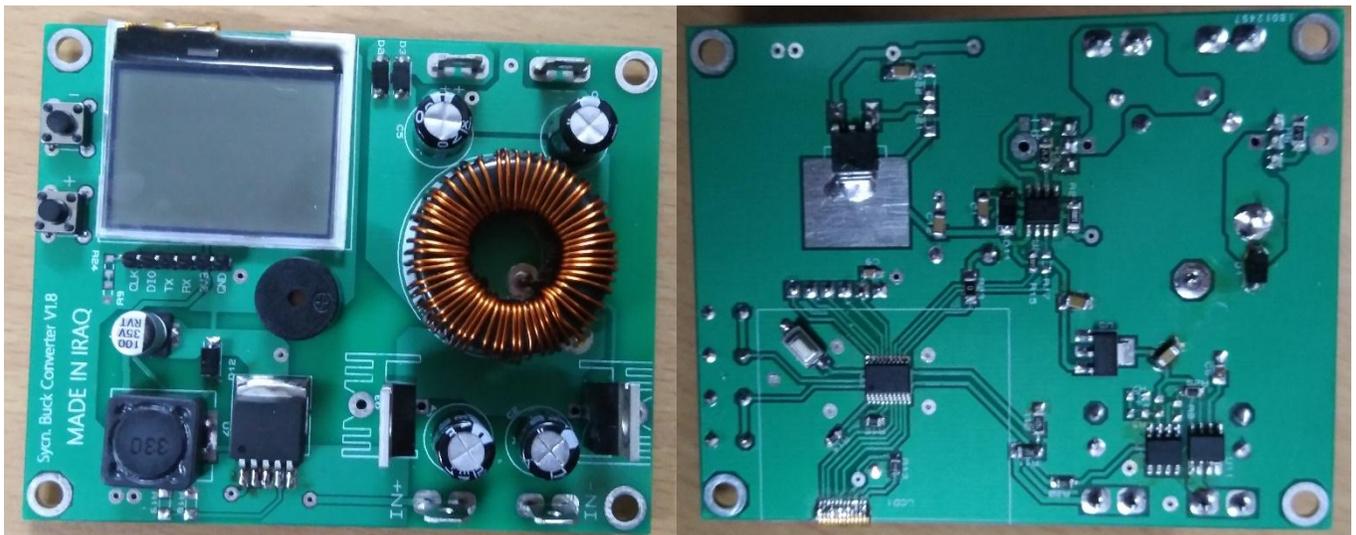


Figure 5-1

List of References

- [1] F. S. Dinniyah, W. Wahab, and M. Alif, "Simulation of Buck-Boost Converter for Solar Panels using PID Controller," *Energy Procedia*, vol. 115, pp. 102-113, 2017.
- [2] X. Zhou and Q. He, "Modeling and Simulation of Buck-Boost Converter with Voltage Feedback Control," in *MATEC Web of Conferences*, 2015, vol. 31: EDP Sciences.
- [3] B. Johansson, "DC-DC converters-dynamic model design and experimental verification," 2005.
- [4] D. Priyadarshini and S. Rai, "Design, Modelling and Simulation of a PID Controller for Buck Boostand Cuk Converter," *International Journal of Science and Research (IJSR) ISSN (Online)*, pp. 2319-7064.
- [5] S. Yang, "Modelling and control of a Buck converter," ed, 2011.
- [6] A. Khajezadeh, A. Ahmadipour, and M. S. Motlagh, "DC-DC CONVERTERS VIA MATLAB/SIMULINK."
- [7] M. F. Badr, "Modelling and Simulation of Closed Loop Controlled DC-DC Converter Fed Solenoid Coil," 2014.
- [8] B. Abdessamad, K. Salah-ddine, and C. E. Mohamed, "A Comparative study of Analog and digital Controller On DC/DC Buck-Boost Converter Four Switch for Mobile Device Applications," *arXiv preprint arXiv:1306.5180*, 2013.
- [9] S. K. Mahapatro, "Maximum power point tracking (MPPT) Of solar cell using buck-boost converter," *International Journal of Engineering & Technology*, vol. 2, pp. 1810-1821, 2013.
- [10] N. A. Ahmed, *Electric Power Systems Research*. 73,4 (2013)
- [11] International Conference – Alternative and Renewable Energy Quest, AREQ 2017, 1-3 February 2017, Spain "Simulation of Buck-Boost Converter for Solar Panels using PID"

Ministry of Higher Education
& Scientific Research
University of Ninevah



College of Electronic Engineering
Department of Systems &
Control Engineering

Turbine Speed Control Using Genetic Algorithm Based On PID Controller

Submitted By

Tiba Saad Mahmood

Sadi Ismail Sadi

Supervised By

Mr. Abdullah Ibrahim Abdullah

Mr. Mohammed Abduljalil

***A project submitted as partial fulfillment for the requirements of the
degree of Bachelor of Science in Systems and Control Engineering***

May - 2018

***Turbine Speed Control Using Genetic Algorithm
Based On PID Controller***

Submitted By

Tiba Saad Mahmood

Sadi Ismail Sadi

Supervised By

Mr. Abdullah Ibrahim Abdulla

Mr. Mohammed Abduljalil

***A Project Submitted As Partial Fulfillment For The
Requirements Of The Degree Of B.Sc. In Systems and Control
Engineering***

May – 2018

Acknowledgement

Alhamdulillah. Firstly, I would express my highest appreciation to Allah S.W.T the Almighty and Merciful for giving me strength and because of His Will; I manage to complete my Final Year Project.

With this opportunity, I would like to acknowledge and extend my heartfelt gratitude to my supervisor, **Mr. Abdullah** and **Mr. Mohammed** for your guidance, support and valuable advice throughout the progress of this project. It would have been difficult to complete this project without enthusiastic support, insight and advice given by them.

My appreciation also goes to **my family** who has been so supportive all these years. Thanks for their encouragement, love and emotional supports that they had given to me.

I have gained a lot of help and support from friends and staffs in the Faculty of Electronic Engineering. Once again, I want to take this opportunity to say thank you to them for their advice and idea that help me completed my project.

Thank you very much. Your sincere help will be remembered for life.

ABSTRACT

It is known that PID controller is employed in every facet of industrial automation. The application of PID controller span from small industry to high technology industry. For those who are in heavy industries such as refineries and ship-buildings, working with PID controller is like a routine work. Hence how do we optimize the PID controller? Do we still tune the PID as what we use to for example using the classical technique that have been taught to us like Ziegler-Nichols method? Or do we make use of the power of computing world by tuning the PID in a stochastic manner?

In this dissertation, it is proposed that the controller be tuned using the Genetic Algorithm technique. Genetic Algorithms (GAs) are a stochastic global search method that emulates the process of natural evolution. Genetic Algorithms have been shown to be capable of locating high performance areas in complex domains without experiencing the difficulties associated with high dimensionality.

Using genetic algorithms to perform the tuning of the controller will result in the optimum controller being evaluated for the system every time.

For this study, the model selected is of turbine speed control system. The reason for this is that this model is often encountered in refineries in a form of steam turbine that uses hydraulic governor to control the speed of the turbine. The PID controller of the model will be designed using the classical method and the results analyzed. The same model will be redesigned using the GA method. The results of both designs will be compared, analyzed and conclusion will be drawn out of the simulation made.

Contents

Chapter	Title	Page
	List of Figure	V
	List of Tables	VI
	List of Symbols	VI
1	Introduction	1
1.1	Objectives	1
1.2	Background	1
2	Genetic Algorithm	4
2.1	Introduction	4
2.2	Characteristics of Genetic Algorithm	4
2.3	Reproduction	5
2.4	Crossover	7
2.5	Mutation	9
2.6	Genetic Algorithm Process	10
2.7	Elitism	11
2.8	Objective Function for a Genetic Algorithm	12
3	PID Controller	13

3.1	Introduction	13
3.2	PID Controller	14
3.3	Continuous PID	14
4	Designing Of PID Controller Using Ziegler-Nichols	16
4.1	Introduction	16
4.2	Designing PID Parameters	16
4.3	Analysis of the Classically Designed Controller	24
5	Designing Of PID Controller Using Genetic Algorithm	30
5.1	Introduction	30
5.2	Initializing the Population of the Genetic Algorithm	31
5.3	Setting The GA Parameters	32
5.4	Performing The Genetic Algorithm	34
5.5	The Objective Function Of The Genetic Algorithm	34
5.6	Results Of The Implemented Genetic Algorithm PID Controller	37
6	Conclusions And Further Works	41
6.1	Conclusions	41
6.2	Further Works	42
	References	43
	Appendix/Matlab Script and Function	44

LIST OF FIGURES

Figure Number	Title	page
Figure 1	Typical Turbine Speed Control.	1
Figure 2.1	Depiction of roulette wheel selection.	6
Figure 2.2	Illustration of a Single Point Crossover	7
Figure 2.3	Illustration of a Multi-Point Crossover	8
Figure 2.4	Illustration of a Uniform Crossover	9
Figure 2.5	Illustration of Mutation Operation	10
Figure 2.6	Graphical Illustration the Genetic Algorithm Outline	10
Figure 3.1	Schematic of The PID Controller – Non-Interacting Form	13
Figure 3.2	Block Diagram Of Continuous PID Controller	14
Figure 4.1	Illustration of sustained Oscillation with Period P_{cr}	16
Figure 4.2	Block Diagram Of Controller And Plant	17
Figure 4.3	Plot of root locus for $G(s)$	19
Figure 4.4	Bode and Nichols Plot of the open loop $G(s)$	20
Figure 4.5	Illustrated the Close Loop Transfer Function	22
Figure 4.6	Simplified System	22
Figure 4.7	Unit Step Response of The Designed System	24
Figure 4.8	Improved System Response	27

Figure 4.9	Optimized System Response	28
Figure.5.1	Chromosome Definition	35
Figure 5.2	GA-PID Controller Design For the turbine control system	36
Figure 5.3	PID Response With Population Size Of 20	37
Figure 5.4	Illustration Of Genetic Algorithm Converging Through Generations	38
Figure 5.5	PID Response With Population Size Of 40	39
Figure 5.6	Illustration Of Genetic Algorithm Converging Through Generations.	40

LIST OF Tables

Table Number	Title	Page
Table 1	Routh Array	18
Table 2	Recommended PID Value Setting	21
Table 3	Result Of Classical Z-N And Genetic PID Controller	40

LIST OF SYMBOLS

<p>GA - Genetic Algorithm</p> <p>RWS-Roulette Wheel selection.</p> <p>SUS-Stochastic Universal Sampling.</p> <p>ZN- Ziegler-Nichols</p> <p>K_{cr} -Critical gain.</p> <p>P_{cr} -Critical period.</p>

Chapter 1

Introduction

1.1 Objectives

The objectives of this project are as follows:

- i. To develop the PID controller to control the speed of turbine.
- ii. To design the controller using Ziegler- Nichols' and genetic algorithm (GA).
- iii. To analyze the performance of the proposed controllers.

1.1 Background

In refineries, in chemical plants and other industries the gas turbine is a well known tool to drive compressors. These compressors are normally of centrifugal type. They consume much power due to the fact that very large volume flows are handled. The combination gas turbine-compressor is highly reliable. Hence the turbine-compressor play significant role in the operation of the plants.[1]

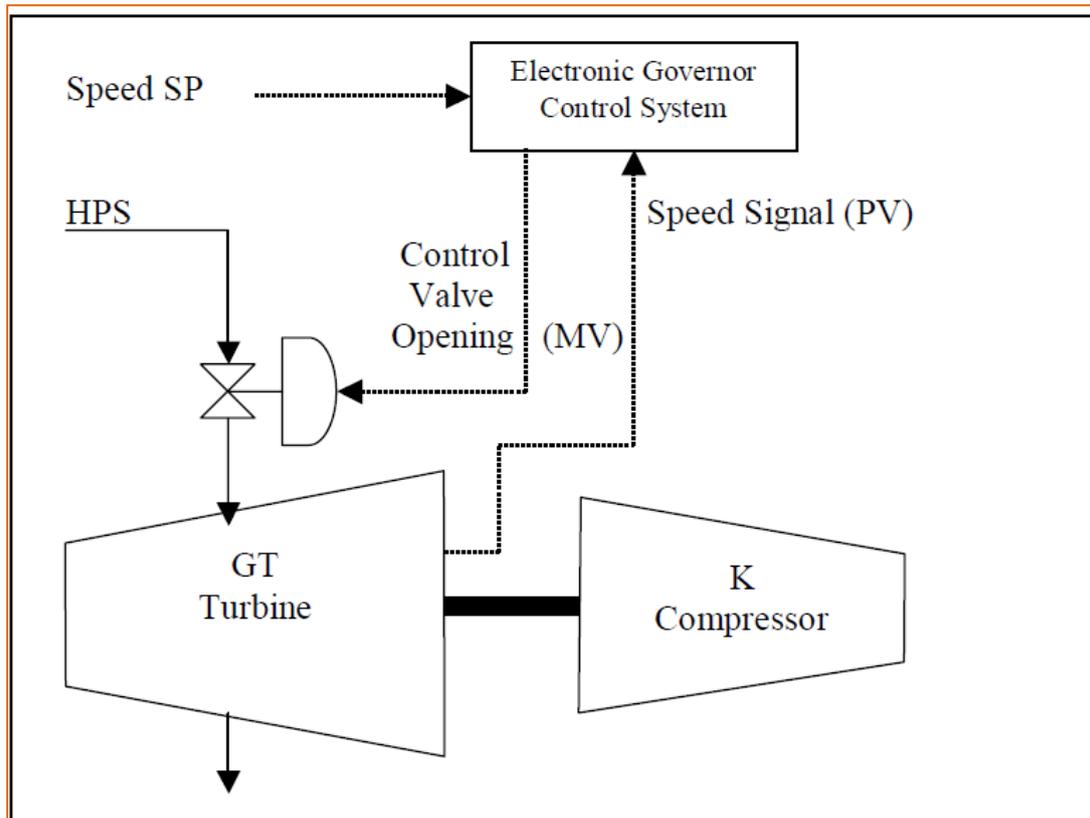


Figure 1. Typical Turbine Speed Control.

In the figure 1 set up, the high pressure steam (HPS) is usually used to drive the turbine. The turbine which is coupled to the compressor will then drive the compressor. The hydraulic governor which, acts as a control valve will be used to throttle the amount of steam that is going to the turbine section. The governor opening is being controlled by a PID which is in the electronic governor control panel.[1]

It is a known fact that the PID controller is employed in every facet of industrial automation. The application of PID controller span from small industry to high technology industry. For those who are in heavy industries such as refineries and ship buildings, working with PID controller is like a routine work. Hence how do we optimize the PID controller? Do we still tune the PID as what we use to for example using the classical technique that have been taught to us like Ziegler-Nichols method? Or do we make use of the power of computing world by tuning the PID in a stochastic manner?

In this project, it is proposed that the controller be tuned using the Genetic Algorithm technique. Using genetic algorithms to perform the tuning of the controller will result in the optimum controller being evaluated for the system every time.

For this study, the model selected is of turbine speed control system. The reason for this is that this model is often encountered in refineries in a form of steam turbine that uses hydraulic governor to control the speed of the turbine as illustrated in figure 1. The complexities of the electronic governor controller will not be taken into consideration in this dissertation. The electronic governor controller is a big subject by itself and it is beyond the scope of this study.

Nevertheless this study will focus on the model that makes up the steam turbine and the hydraulic governor to control the speed of the turbine. In the context of refineries, you can consider the steam turbine as the heart of the plant. This is due to the fact that in the refineries, there are lots of high capacities compressors running on steam turbine. Hence this makes the control and the tuning optimization of the steam turbine significant.[2]

In this project, it will be shown that the GA tuned PID will result in a better optimization of the process. Here is a brief description of how GA works. A GA is typically initialized with a random population consisting of between 20-100 individuals. This population or mating pool is usually represented by a real-valued number or a binary string called a chromosome. How well an individual performs a task is measured and

assessed by the objective function. The objective function assigns each individual a corresponding number called its fitness. The fitness of each chromosome is assessed and a survival of the fittest strategy is applied. There are three main stages of a genetic algorithm, these are known as reproduction, crossover and mutation. During the reproduction phase the fitness value of each chromosome is assessed. This value is used in the selection process to provide bias towards fitter individuals. Just like in natural evolution, a fit chromosome has a higher probability of being selected for reproduction. This continues until the selection criterion has been met. The probability of an individual being selected is thus related to its fitness, ensuring that fitter individuals are more likely to leave offspring. Multiple copies of the same string may be selected for reproduction and the fitter strings should begin to dominate. Once the selection process is complete, the crossover algorithm is initiated. The crossover operations swaps certain parts of the two selected strings in a bid to capture the good parts of old chromosomes and create better new ones. Genetic operators manipulate the characters of a chromosome directly, using the assumption that certain individual's gene codes, on average, produce fitter individuals. The crossover probability indicates how often crossover is performed . A probability of 0% means that the offspring. will be exact replicas of their parents. and a probability of 100% means that each generation will be composed of entirely new offspring . Using selection and crossover on their own will generate a large amount of different strings. However there are two main problems with this:[3]

1. Depending on the initial population chosen, there may not be enough diversity in the initial strings to ensure the GA searches the entire problem space.
2. The GA may converge on sub-optimum strings due to a bad choice of initial population.

These problems may be overcome by the introduction of a mutation operator into the GA. Mutation is the occasional random alteration of a value of a string position. It is considered a background operator in the genetic algorithm. The probability of mutation is normally low because a high mutation rate would destroy fit strings and degenerate the genetic algorithm into a random search. Mutation probability values of around 0.1% or 0.01% are common, these values represent the probability that a certain string will be selected for mutation for an example for a probability of 0.1%; one string in one thousand will be selected for mutation. Once a string is selected for mutation, a randomly chosen element of the string is changed or mutated.

Chapter 2

Genetic Algorithm

2.1 Introduction

Genetic Algorithms (GA.s) are a stochastic global search method that mimics the process of natural evolution. It is one of the methods used for optimization. John Holland formally introduced this method in the United States in the 1970 at the University of Michigan. The continuing performance improvements of computational systems has made them attractive for some types of optimization.

The genetic algorithm starts with no knowledge of the correct solution and depends entirely on responses from its environment and evolution operators such as reproduction, crossover and mutation to arrive at the best solution. By starting

at several independent points and searching in parallel, the algorithm avoids local

minima and converging to sub optimal solutions. [4]

2.2 Characteristics of Genetic Algorithm

Genetic Algorithms are search and optimization techniques inspired by two biological principles namely the process of natural selection and the mechanics of natural genetics. GAs manipulate not just one potential solution to a problem but a collection of potential solutions. This is known as population. The potential solution in the population is called chromosomes. These chromosomes are the encoded representations of all the parameters of the solution. Each chromosomes

is compared to other chromosomes in the population and awarded fitness rating

that indicates how successful this chromosomes to the latter. To encode better solutions, the GA will use genetic operators or evolution operators such as crossover and mutation for the creation of new chromosomes from the existing ones in the population .This is achieved by either merging the existing ones in the population or by modifying an existing chromosomes. The selection mechanism for parent chromosomes takes the fitness of the parent into account. This will ensure that the better solution will have a higher chance to procreate and donate their beneficial characteristic to their offspring.

A genetic algorithm is typically initialized with a random population consisting of

between 20-100 individuals. This population or also known as mating pool is usually represented by a real-valued number or a binary string called a chromosome. For illustrative purposes, the rest of this section represents each chromosome as a binary string. How well an individual performs a task is measured and assessed by the objective function. The objective function assigns

each individual a corresponding number called its fitness. The fitness of each chromosome is assessed and a survival of the fittest strategy is applied. In this

project, the magnitude of the error will be used to assess the fitness of each chromosome.

There are three main stages of a genetic algorithm, these are known as reproduction, crossover and mutation. This will be explained in details in the following section.[4]

2.3 Reproduction

During the reproduction phase the fitness value of each chromosome is assessed. This value is used in the selection process to provide bias towards fitter individuals. Just like in natural evolution, a fit chromosome has a higher probability of being selected for reproduction. An example of a common selection technique is the ‘*Roulette Wheel*’ selection method, Figure 2.1. Each individual in the population is allocated a section of a roulette wheel; the size of the section is proportional to the fitness of the individual. A pointer is spun and the individual to whom it points is selected. This continues until the selection criterion has been met. The probability of an individual being selected is thus related to its fitness, ensuring that fitter individuals are more likely to leave offspring. Multiple copies of the same string may be selected for reproduction and the fitter strings should begin to dominate. However, for the situation illustrated in Figure 2.1, it is not implausible for the weakest string (01001) to dominate the selection process.

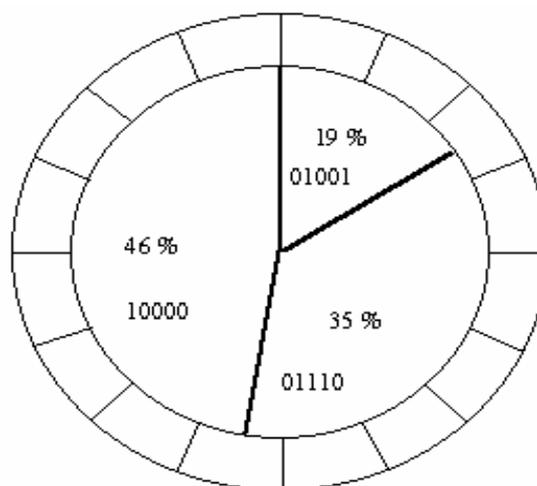


Figure 2.1 Depiction of roulette wheel selection.

There are a number of other selection methods available and it is up to the user to select the appropriate one for each process. All selection methods

are based on the same principal i.e. giving fitter chromosomes a larger probability of selection.[4]

Four common methods for selection are:

1. Roulette Wheel selection.(RWS)
2. Stochastic Universal sampling.(SUS)
3. Normalized geometric selection.
4. Tournament selection.

2.4 Crossover

Once the selection process is complete, the crossover algorithm is initiated. The crossover operations swaps certain parts of the two selected strings in a bid to capture the good parts of old chromosomes and create better new ones. Genetic operators manipulate the characters of a chromosome directly, using the assumption that certain individual's gene codes, on average, produce fitter individuals. The crossover probability indicates how often crossover is performed. A probability of 0% means that the 'offspring' will be exact replicas of their 'parents' and a probability of 100% means that each generation will be composed of entirely new offspring. The simplest crossover technique is the Single Point Crossover. There are two stages involved in single point crossover:[4]

1. Members of the newly reproduced strings in the mating pool are 'mated' (paired) at random.
2. Each pair of strings undergoes a crossover as follows: An integer k is randomly selected between one and the length of the string less one, $[1, L-1]$. Swapping all the characters between positions $k+1$ and L inclusively creates two new strings.

Example: If the strings 10000 and 01110 are selected for crossover and the value of k is randomly set to 3 then the newly created strings will be 10010 and 01100 as shown in Figure 2.2.

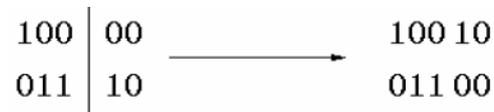


Figure 2.2 Illustration of a Single Point Crossover

More complex crossover techniques exist in the form of Multi-point and Uniform Crossover Algorithms. Multi-point crossover is an extension of the single point crossover algorithm and operates on the principle that the parts of a chromosome that contribute most to its fitness might not be adjacent. There are three main stages involved in a Multi-point crossover.

1. Members of the newly reproduced strings in the mating pool are ‘mated’ (paired) at random.
2. Multiple positions are selected randomly with no duplicates and sorted into ascending order.
3. The bits between successive crossover points are exchanged to produce new offspring.

Example: If the string *11111* and *00000* were selected for crossover and the multipoint crossover positions were selected to be 2 and 4 then the newly created strings will be *11001* and *00110* as shown in Figure 2.3.



Figure 2.3 Illustration of a Multi-Point Crossover

In uniform crossover, a random mask of ones and zeros of the same length as the parent strings is used in a procedure as follows.

1. Members of the newly reproduced strings in the mating pool are 'mated' (paired) at random.
2. A mask is placed over each string. If the mask bit is a one, the underlying bit is kept. If the mask bit is a zero then the corresponding bit from the other string is placed in this position.[4]

Example: If the string *10101* and *01010* were selected for crossover with the mask *10101* then newly created strings would be *11111* and *00000* as shown in Fig. 2.4.

$$\begin{array}{ccc}
 10101 & \xrightarrow{10101} & 11111 \\
 01010 & & 00000
 \end{array}$$

Figure 2.4 Illustration of a Uniform Crossover

Uniform crossover is the most disruptive of the crossover algorithms and has the capability to completely dismantle a fit string, rendering it useless in the next generation. Because of this Uniform Crossover will not be used in this project.[4]

2.5 Mutation

Using *selection* and *crossover* on their own will generate a large amount of different strings. However there are two main problems with this:

1. Depending on the initial population chosen, there may not be enough diversity in the initial strings to ensure the GA searches the entire problem space.
2. The GA may converge on sub-optimum strings due to a bad choice of initial population.

These problems may be overcome by the introduction of a mutation operator into the GA. Mutation is the occasional random alteration of a value of a string position. It is considered a background operator in the genetic algorithm. The probability of mutation is normally low because a high mutation rate would destroy fit strings and degenerate the genetic

algorithm into a random search. Mutation probability values of around 0.1% or 0.01% are common, these values represent the probability that a certain string will be selected for mutation i.e. for a probability of 0.1%; one string in one thousand will be selected for mutation. Once a string is selected for mutation, a randomly chosen element of the string is changed or ‘mutated’. For example, if the GA chooses bit position 4 for mutation in the binary string *10000*, the resulting string is *10010* as the fourth bit in the string is flipped as shown in Figure 2.5.[4]

$$10000 \xrightarrow{\text{Mutate}} 10010$$

Figure 2.5 Illustration of Mutation Operation

2.6 Genetic Algorithm Process

In this section the process of Genetic Algorithm will be summarized in a flowchart. The summary of the process will be described below.

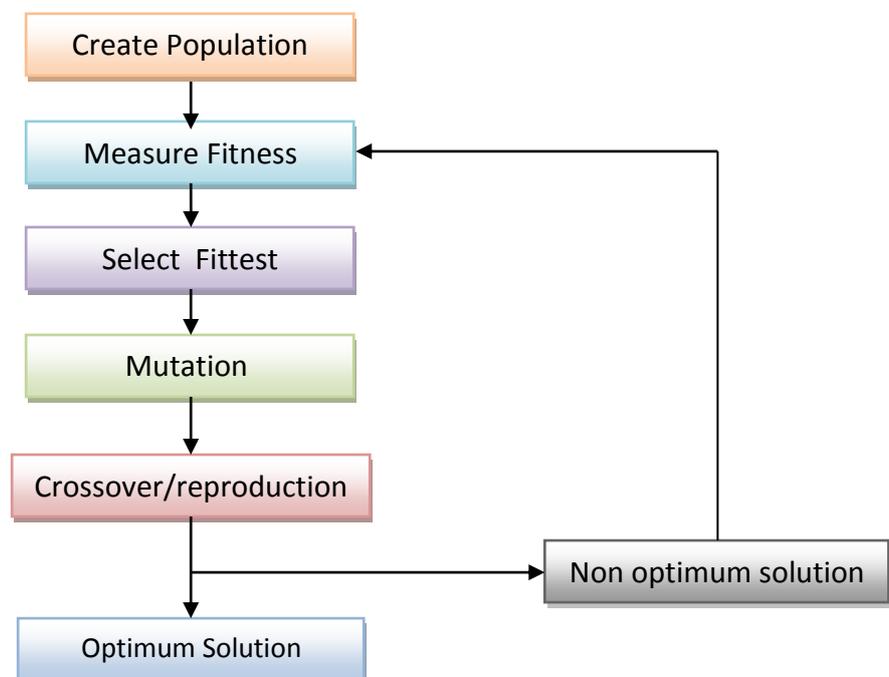


Figure 2.6 Graphical Illustration the Genetic Algorithm Outline

The steps involved in creating and implementing a genetic algorithm are as follows:

1. Generate an initial, random population of individuals for a fixed size.
2. Evaluate their fitness.
3. Select the fittest members of the population.
4. Reproduce using a probabilistic method (e.g., roulette wheel).
5. Implement crossover operation on the reproduced chromosomes (choosing probabilistically both the crossover site and the ‘mates’).
6. Execute mutation operation with low probability.
7. Repeat step 2 until a predefined convergence criterion is met.

The convergence criterion of a genetic algorithm is a user-specified condition e.g. the maximum number of generations or when the string fitness value exceeds certain threshold.

2.7 Elitism

With crossover and mutation taking place, there is a high risk that the optimum solution could be lost as there is no guarantee that these operators will preserve the fittest string. To counteract this, elitist models are often used. In an elitist model, the best individual from a population is saved before any of these operations take place. After the new population is formed and evaluated, it is examined to see if this best structure has been preserved. If not, the saved copy is reinserted back into the population. The GA then continues on as normal [4]

2.8 Objective Function for a Genetic Algorithm

Writing an objective function is the most difficult part of creating a genetic algorithm. In this project, the objective function is required to evaluate the best PID controller for the turbine system. An objective function could be created to find a PID controller that gives the smallest overshoot, fastest rise time or quickest settling time but in order to combine all of these objectives it was decided to design an objective function that will minimize the error of the controlled system. Each chromosome in the population is passed into the objective function one at a time. The chromosome is then evaluated and assigned a number to represent its fitness, the bigger its number the better its fitness. The genetic algorithm uses the chromosome's fitness value to create a new population consisting of the fittest members.[5]

Each chromosome consists of three separate strings constituting a P, I and D term, as defined by the 3-row 'bounds' declaration when creating the population. When the chromosome enters the evaluation function, it is split up into its three terms, and the P, I and D gains are used to create a PID controller according to Equation

$$G_{PID} = \frac{K_d s^2 + K_p s + K_i}{s} \quad - - - - - 1$$

The newly formed PID controller is placed in a unity feedback loop with the turbine transfer function. In order to reduce the compile time of the program the turbine transfer function is defined in another file and imported as a global variable. The controlled system is given a step input and the error is assessed using an appropriate error performance criterion MSE. The chromosome is assigned an overall fitness value according to the magnitude of the error, the smaller the error the larger the fitness value.

Chapter 3

PID Controller

3.1 Introduction:

PID controller consists of Proportional Action, Integral Action and Derivative Action. It is commonly refer to Ziegler-Nichols PID tuning parameters. It is by far the most common control algorithm [1]. In this chapter, the basic concept of the PID controls will be explained.

PID controllers algorithm are mostly in feedback loops. PID controllers can be implemented in many forms. It can implemented as a stand-alone controller or as part of Direct Digital Control (DDC) package or even Distributed Control System (DCS). The latter is a hierarchical distributed process control system which is widely used in process plants such as pharmaceutical or oil refining industries. It is interesting to note that more than half of the industrial controllers in use today utilize PID or modified PID control schemes. Below is a simple diagram illustrating the schematic of the PID controller. Such set up is known as non- interacting form or parallel form.[1]

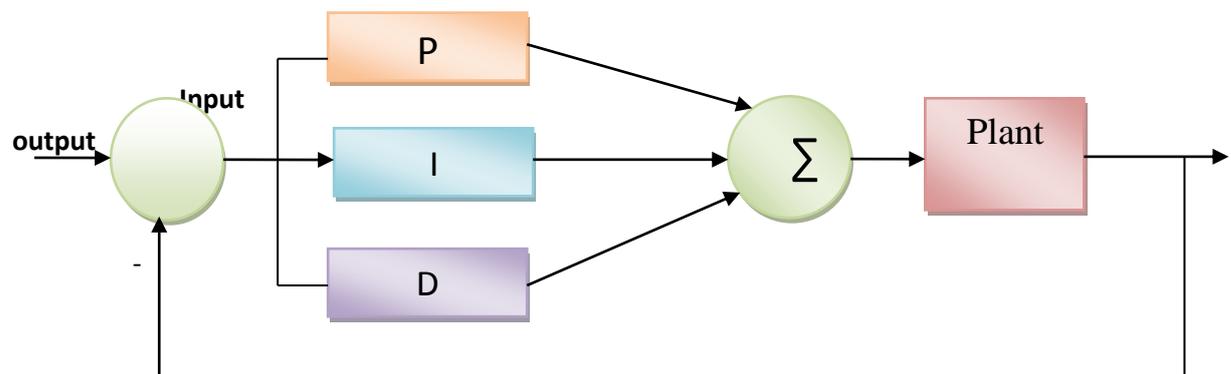


Figure 3.1 Schematic of The PID Controller –Non-Interacting Form

3.2 PID Controller

In proportional control,

$$P_{term} = K_p \times Error \quad \text{--- 2}$$

It uses proportion of the system error to control the system. In this action an offset is introduced in the system. In Integral form,

$$I_{term} = K_I * \int Error dt \quad \text{--- 3}$$

It is proportional to the amount of error in the system. In this action, the I-action will introduce a lag in the system. This will eliminate the offset that was introduced on by the P-action. In Derivative control,

$$D_{term} = K_D * \frac{d(error)}{dt} \quad \text{--- 4}$$

It is proportional to the rate of change of the error. In this action, the D-action will introduce a lead in the system. This will eliminate the lag in the system that was introduced by the I-action earlier on.[2]

3.3 Continuous PID

The three controllers when combined together can be represented by the following transfer function.

$$G_C(s) = K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \quad \text{--- 5}$$

This can be illustrated in the following block diagram

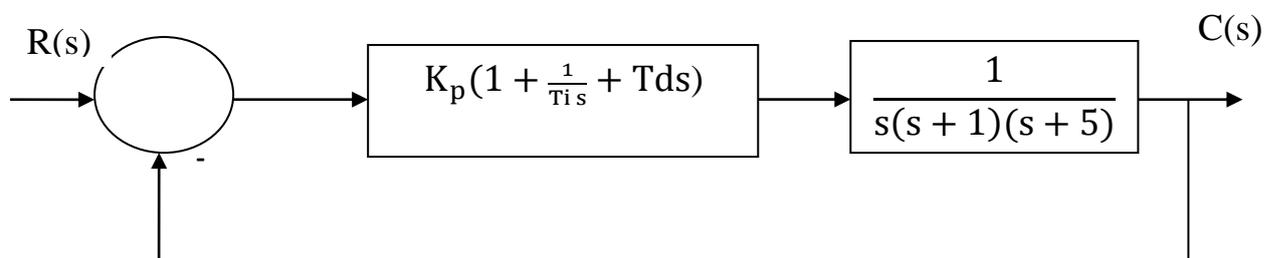


Figure 3.2 Block Diagram Of Continuous PID Controller.

What the PID controller does is basically is to act on the variable to be manipulated through a proper combination of the three actions that is the P control action, I control action and D control action. The P action

is the control action that is proportional to the actuating error signal, which is the difference between the input and the feedback signal. The I action is the control action which is proportional to the integral actuating error signal. Finally the D action is the control action which is proportional to the derivative of the actuating error signal.

With the integration of all the three actions, the continuous PID can be realized. This type of controller is widely used in industries all over the world. In fact a lot of research, studies and application has been discovered in the recent years.[2]

Chapter 4

Designing Of PID Controller Using Ziegler-Nichols

4.1 Introduction

For the system under study, Ziegler-Nichols tuning rule based on critical gain K_{cr} and critical period P_{cr} will be used. In this method, the integral time T_i will be set to infinity and the derivative time T_d to zero. This is used to get the initial PID setting of the system.

In this method, only the proportional control action will be used. The K_p will be increase to a critical value K_{cr} at which the system output will exhibit sustained oscillations. In this method, if the system output does not exhibit the sustained oscillations hence this method does not apply.

In this chapter, it will be shown that the inefficiency of designing PID controller using the classical method. [7].

4.2 Designing PID Parameters

From the response below, the system under study is indeed oscillatory and hence the Ziegler-Nichols tuning rule based on critical gain K_{cr} and critical period P_{cr} can be applied. [7]

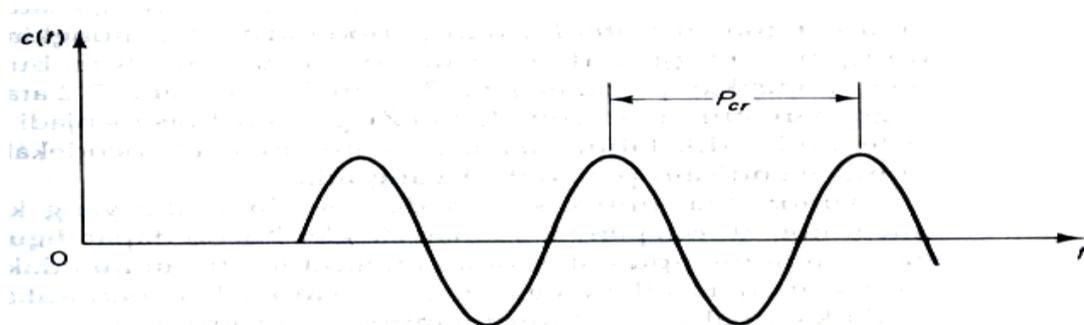


Figure 4.1 Illustration of sustained Oscillation with Period P_{cr} .

The transfer function of the PID controller is

$$G_C(s) = K_P(1 + \frac{1}{T_i S} + T_d S)$$

The objective is to achieve a unit-step response curve of the designed system that exhibits a maximum overshoot of 30 %. If the maximum overshoot is excessive says about greater than 40%, fine tuning should be done to reduce it to less than 30%.

The system under study above has a following block diagram

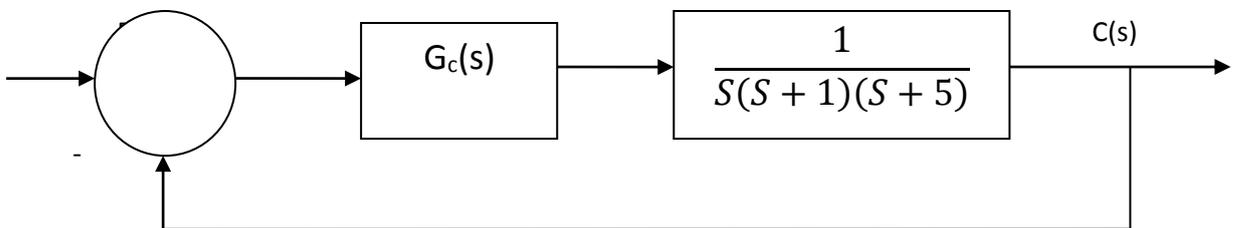


Figure 4.2 Block Diagram Of Controller And Plant

Since $T_i = \infty$ and $T_d = 0$, this can be reduced to the transfer function of

$$\frac{C(s)}{R(s)} = \frac{K_p}{S(S+1)(S+5) + K_p}$$

The value of K_p that makes the system marginally stable so that sustained oscillation occurs can be obtained by using the Routh's stability criterion.

Since the characteristic equation for the closed-loop system is

$$S^3 + 6S^2 + 5S + K_p = 0$$

From the Routh's Stability Criterion, the value of K_p that makes the system marginally stable can determined. [7]

Table 1 illustrates the Routh array.

Table 1. Routh Array

S^3	1	5
S^2	6	K_p
S^1	$(30-K_p)/6$	0
S^0	K_p	0

By observing the coefficient of the first column, the sustained oscillation will occur if $K_p=30$.

Hence the critical gain K_{cr} is

$$K_{cr}=30$$

Thus with K_p set equal to K_{cr} , the characteristic equation becomes

$$S^3+6S^2+5S+30=0$$

The frequency of the sustained oscillation can be determined by substituting the S terms with jw term. Hence the new equation becomes

$$(jw)^3 + 6(jw)^2 + 5(jw) + 30 = 0$$

This can be simplified to

$$6(5 - w^2) + jw(5 - w^2) = 0$$

From the above simplification, the sustained oscillation can be reduced to

$$w^2 = 5$$

or

$$w = \sqrt{5} = 2.23603 \text{ rad/sec}$$

A second method we can use the root-locus to tune the Ziegler-Nichols to evaluate the PID gains for the system. Using the 'rlocfind' command in Matlab, the crossover point and gain of the system were found to be $0.0000 + 2.23603i$ and 30 respectively as shown in Figure 4.3.

```

% root-locus to calculate crossover point (w )and gain (Kcr) using
MATLAB
clear
clc
n=[1];
d=[1 6 5 0];
rlocus(n,d)
rlocfind(n,d)

```

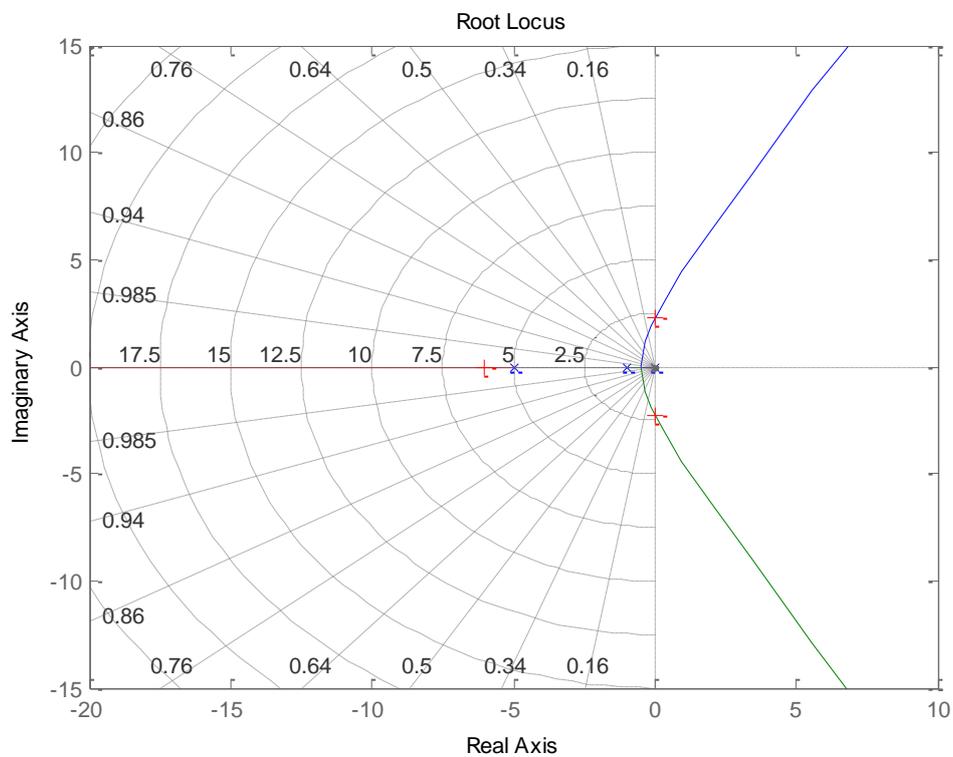


Figure 4.3 Plot of root locus for $G(s)$.

A third method we can use the bode plot or Nichols to tuned the Ziegler-Nichols PID controller by using the 'bode' command in Matlab to evaluate the gain margin as shown in figure 4.4.

```

% Bode plot to calculate gain ( $K_{cr}$ ) and ( $w$ ) from gain margin using
MATLAB
clear
clc
n=[1];

```

```
d=[1 6 5 0];
```

```
G=tf(n , d);
```

```
sisotool(G)
```

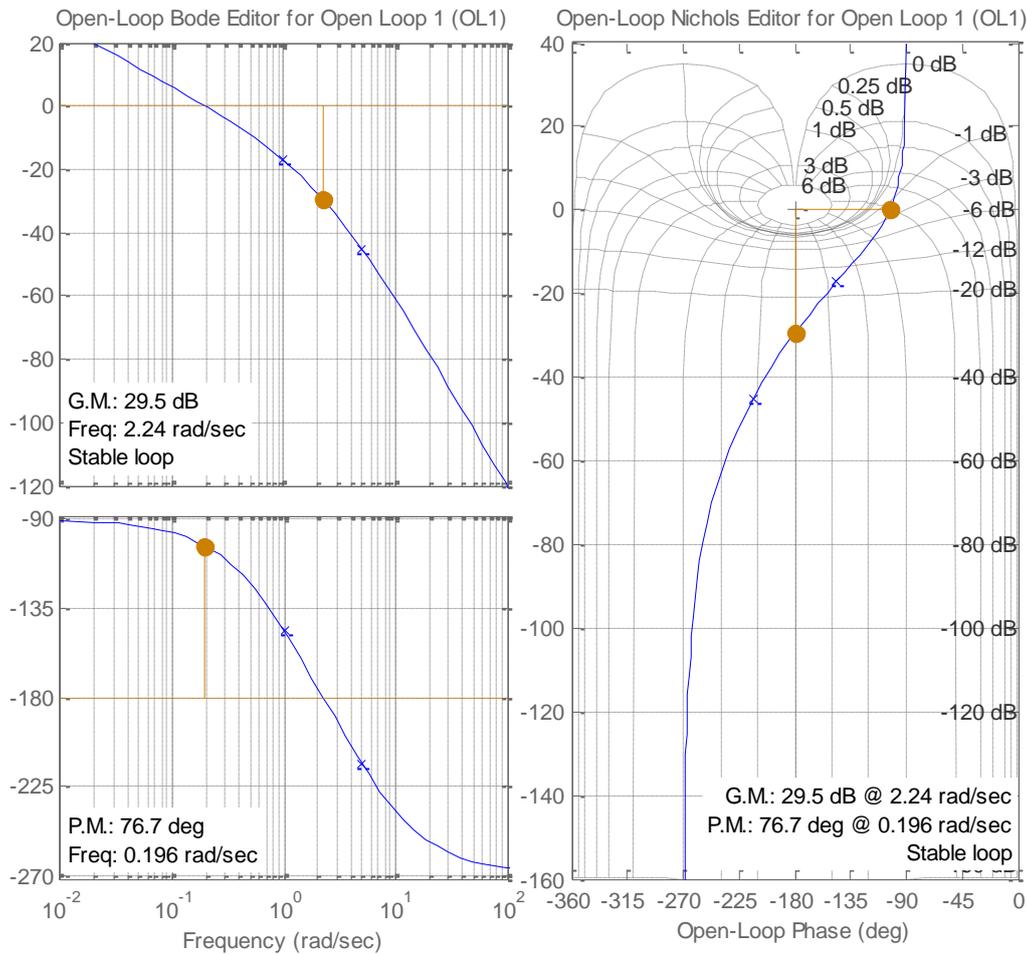


Figure 4.4 Bode and Nichols Plot of the open loop $G(s)$.

From gain margin we can calculate K_{cr} as follows

$$20 \log K_{cr} = \text{gain margin}$$

$$K_{cr} = 10^{29.5 \text{ dB}/20} = 29.8538$$

$$\omega = 2.24 \text{ rad/sec}$$

The period of the sustained oscillation can be calculated as

$$\begin{aligned} P_{cr} &= 2\pi/\sqrt{5} \\ &= 2.8099 \text{ sec} \end{aligned}$$

From Ziegler-Nichols frequency method of the second method, the able suggested tuning rule according to the formula shown. From these we are able to estimate the parameters of K_p , T_i and T_d . [2]

Table 2. Recommended PID Value Setting

Type of Controller	K_p	T_i	T_d
P	$0.5 K_{cr}$	∞	0
PI	$0.45 K_{cr}$	$(1/1.2) P_{cr}$	0
PID	$0.6 K_{cr}$	$0.5 P_{cr}$	$0.125 P_{cr}$

Hence the from the table, the values of the PID parameters K_p , T_i and T_d will be

$$K_p = 0.6 * K_{cr} = 0.6 * 30 = 18$$

$$T_i = 0.5 * 2.8099$$

$$= 1.405$$

$$\implies K_i = K_p / T_i = 30 / 1.405 = 21.35231$$

$$T_d = 0.125 * 2.8099$$

$$= 0.351$$

$$\implies K_d = K_p * T_d = 30 * 0.351 = 10.53$$

The transfer function of the PID controller with all the parameters is given as

$$\begin{aligned}
 G_s(s) &= K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \\
 &= 18 \left(1 + \frac{1}{1.405 s} + 0.35124 s \right) \\
 &= \frac{6.3223(s + 1.435)^2}{s}
 \end{aligned}$$

From the above transfer function, we can see that the PID controller has pole at the origin and double zero at $s = -1.4235$.

The block diagram of the control system with PID controller is shown in fig.4.5.

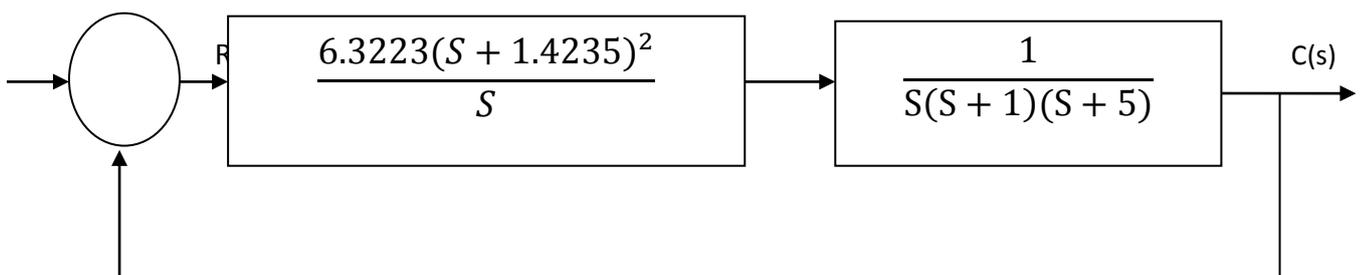


Figure 4.5 Illustrated the Close Loop Transfer Function.

Using MATLAB function, the following system can be easily calculated. The system in fig. 4.4 can be reduced to a single block by using the following MATLAB function. Below is the MATLAB codes that will calculate the two blocks in series.

```
% calculation of series system response using MATLAB
num1=[0 6.3223 17.999 12.8089];
den1= [0 0 1 0];
num2= [0 0 0 1];
den2= [1 6 5 0];
[num, den]=series (num1, den1, num2, den2);
Printsys (num, den)
```

This will give the following answer

```
num/den=
  6.3223 s^2 + 17.999 s + 12.8089
-----
          s^4 + 6 s^3 + 5 s^2
```

Hence the above diagram is reduced to simplified system as shown in

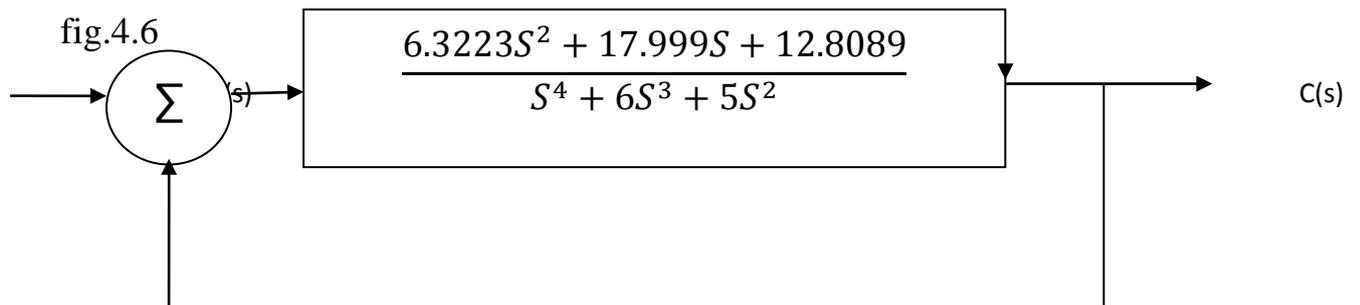


Figure 4.6 Simplified System

Using another MATLAB function, the overall function with its feedback can be calculated as follow

```
% calculation of feedback system response using MATLAB
num1= [0 0 6.3223 17.999 12.8089];
den1= [1 6 5 0 0];
num2= [0 0 0 0 1];
den2= [0 0 0 0 1];
[num, den]=feedback (num1, den1, num2, den2);
Printsys (num, den)
```

This will result to

$$\text{num/den} = \frac{6.3223 s^2 + 17.999 s + 12.8089}{s^4 + 6 s^3 + 11.3223 s^2 + 17.8089 s + 12.8089}$$

Therefore the overall close loop system response of

$$\frac{C(s)}{R(s)} = \frac{6.3223S^2 + 17.999S + 12.808}{S^4 + 6S^3 + 11.3223S^2 + 18S + 12.8089}$$

The unit step response of this system can be obtained with MATLAB and shown in figure 4.7.

```
%MATLAB script of the Designed PID controller system
num=[0 0 6.3223 18 12.8];
den=[1 6 11.3223 18 12.811];
step (num, den);
grid;
title ('Unit Step Response of the designed system')
```

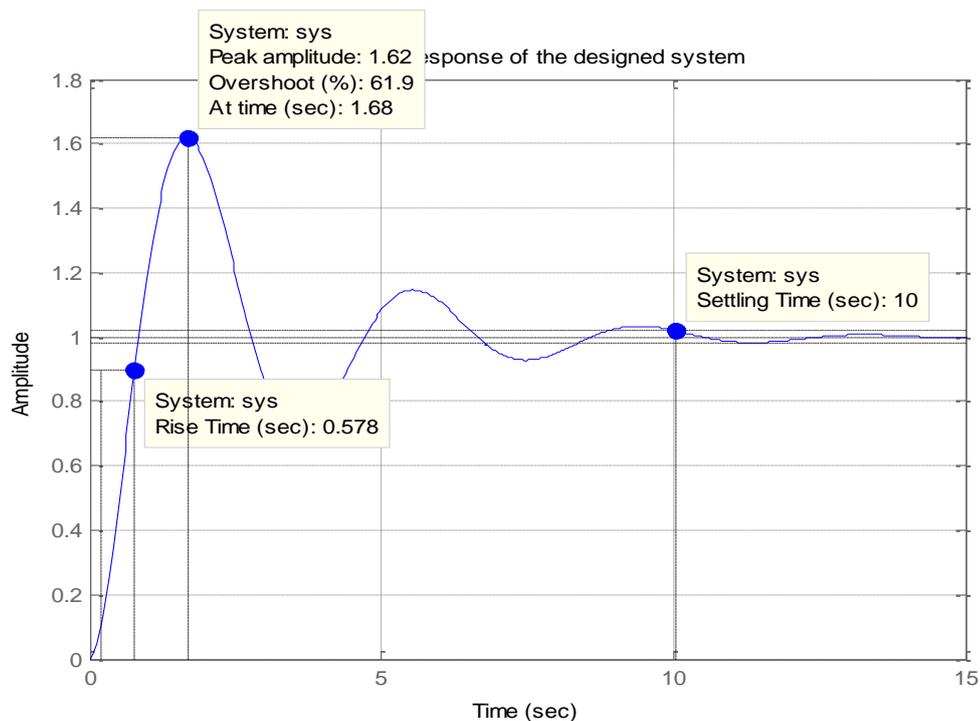


Figure 4.7 Unit Step Response of The Designed System.

4.3 Analysis of the Classically Designed Controller

From the above diagram, we can utilize the response of the system. The zero and pole of the system can be calculated using the MATLAB function "tf2zp". We can analyze them via the following parameters:

Delay time, t_d

Rise time, t_r

Peak time, t_p

Maximum Overshoot, M_p

Settling time, t_s

The delay time, t_d of the above system is the time taken to reach 50% of the final response time is about 0.5 sec.

The rise time, t_r is the time taken to reach 5 to 95% of the final value is about 0.578 sec.

The peak time, t_p is the time taken for the system to reach the first peak of overshoot is 1.68 sec.

The Maximum overshoot, M_p of the system is approximately 61.9 %. Finally, the Settling time, t_s is about 10 sec. from the analysis above, the system has not been tuned to its optimum. Here we can improve the system by looking into the system zero and pole.

The system zeros and poles can be calculated using MATLAB function mentioned below.

```
% calculation of zero and pole of the system response using MATLAB
```

```
num=[0 0 6.3223 17.999 12.8089];
```

```
den=[1 6 11.3223 17.009 12.8089];
```

```
[z, p, k]= tf2zp (num, den)
```

```
Results:
```

z=

-1.4387

-1.4282

p=

-4.0478

-0.3532+1.5542i

-0.3532- 1.5542i

-1.2457

k=

6.3223

The above result shows that the system is stable since all the poles are located on the left side of the s-plane.

To optimize the response further, the PID controller transfer function must be revisited.

The transfer function of the designed PID controller is

$$\begin{aligned}G_c(s) &= K_p \left(1 + \frac{1}{T_i S} + T_d S \right) \\ &= 18 \left(1 + \frac{1}{1.405 S} + 0.35124 S \right) \\ &= \frac{6.3223(S+1.4235)^2}{S}\end{aligned}$$

The PID controller has a double zero of -1.4235. By trial and error, let keeps the $K_p=18$ and change the location of the double zero from -1.4235 to -0.65.

The new PID controller will have the following parameters:

$$\begin{aligned}G_C(s) &= 18\left(1 + \frac{1}{8.077s^2 + 0.7692s}\right) \\ &= 13.846 \frac{(s + 0.65)^2}{s}\end{aligned}$$

$$= \frac{13.864s^2 + 17.998s + 5.65}{s^4 + 6s^3 + 5s^2}$$

The total response with a unity feedback can be calculated as follow

$$\frac{C(s)}{R(s)} = \frac{13.846s^2 + 17.998s + 5.85}{s^4 + 6s^3 + 16.846s^2 + 17.998s + 5.85}$$

The response of the above system can be illustrated in the fig.4.8.

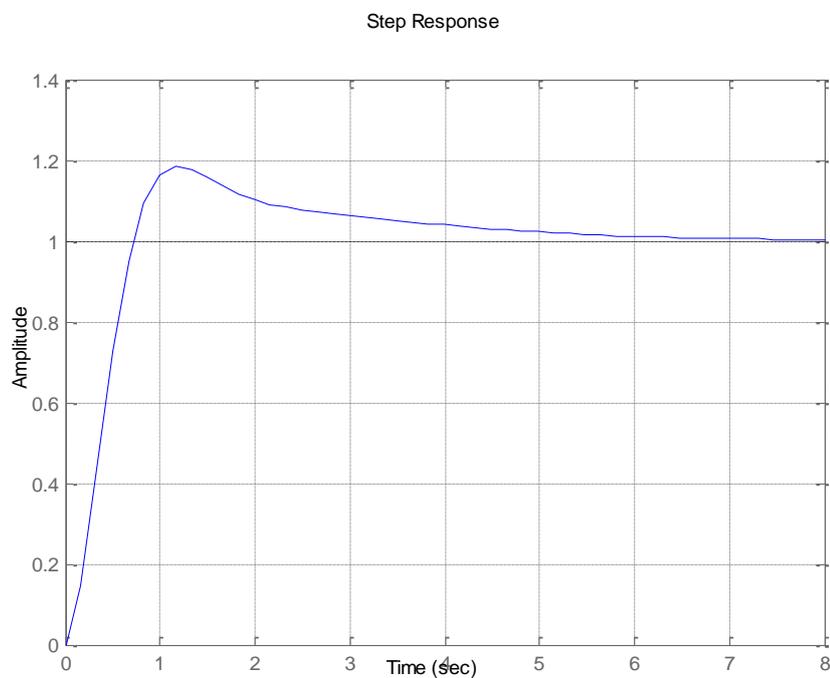


Figure 4.8 Improved System Response

The new system response has somehow improved. The Maximum Overshoot, M_p had reduced to approximately 18%. The Settling Time, t_s has improved from 14 sec to 6 sec. the Peak Time, t_p and Delay time, t_d has increased. The final amplitude has improved at the expense of the system time. The new PID parameters can be calculated as are $K_p=18$, $T_i=3.077$ and $T_d=0.7692$.

To improve the system further, lets increase the K_p value to 39.42. The location of double zero will be kept the same i.e. $s=0.65$. The new transfer function of the PID controller will be

$$\begin{aligned} G(s) &= 39.42\left(1 + \frac{1}{3.077s} + 0.769s\right) \\ &= 30.322 \frac{(s + 0.65)^2}{s} \end{aligned}$$

Using the MATLAB command, the above function together with the plant transfer function and the unity feedback can be determined.

The result is
$$G_c(s) = \frac{27.729s^2 + 36s + 11.7}{s^4 + 6s^3 + 32.729s^2 + 36s + 11.7}$$

The system response can be shown in fig.4.9

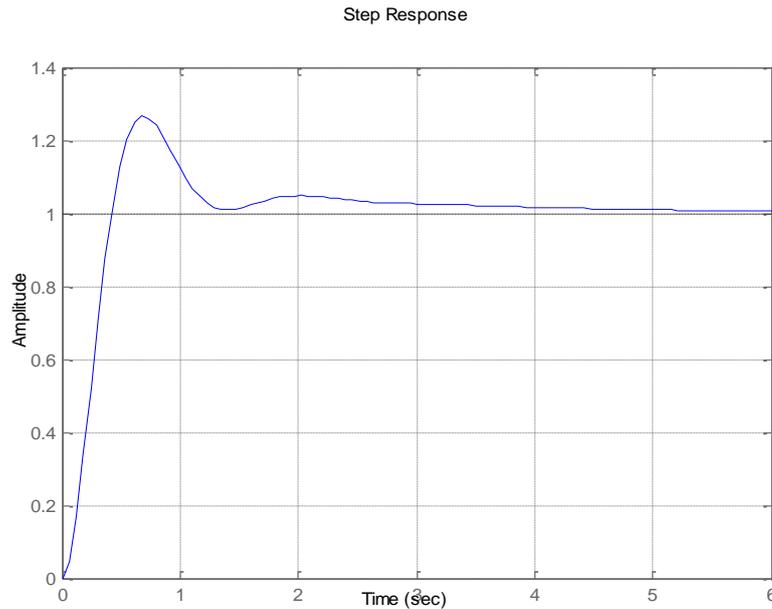


Figure 4.9 Optimized System Response

The above response shows that the system has improved. The response is faster than the one shown in figure 4.7. The Maximum Overshoot of 26.5 %. This is still acceptable since the Maximum Overshoot allowable is less than 30 %. The settling time t_s of 3.65 sec. The rise time t_r of 0.286 sec. The new PID controller can be calculated as $K_p=39.42$, $T_i=3.077$ and $T_d=0.7692$.

In the various plots above, the various responses and its design parameters can be observed. Hence we can clearly see that the final parameters are more superior than the earlier two responses. However the setback is the M_p , which is more than the M_p of the second response. Nevertheless the final response M_p is still within the 30 % Maximum Overshoot allowable. The settling time, t_s of the second and the third response fared much better than the first response. The t_s reached its steady-state in much faster than the original time taken by the original response.

It is interesting to observe that these values are approximately twice the values suggested by the second method of Ziegler-Nichols tuning rule.

Hence we can conclude that Ziegler-Nichols tuning has provided us a starting point for a fine tuning.

It is observed that for the case where the double zero is located at $s = -1.425$, increasing the value of K_p increases the speed of the response. However this does not improve the percentage maximum overshoot. In fact varying K_p has little impact on the percentage maximum overshoot. On the other hand, varying the double zero has significant effect on the maximum overshoot. The zero is shifted from -1.425 to -0.65 and we observed that the maximum overshoot reduces. Finally to achieve a better result, we have to have to double the K_p value coupled with the new zero value and hence the better percentage maximum overshoot can be achieved. The above can explained through the root-locus analysis.

Chapter 5

Designing Of PID Controller Using Genetic Algorithm

5.1 Introduction

Before we go into the above subject. It is good to discuss the differences between Genetics Algorithm against the traditional methods. This will help us understand

why GA is more efficient than the latter. Genetic algorithms are substantially different to the more traditional search and optimization techniques. The five main differences are:

1. Genetic algorithms search a population of points in parallel, not from a single point.
2. Genetic algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the direction of the search.
3. Genetic algorithms use probabilistic transition rules, not deterministic rules.
4. Genetic algorithms work on an encoding of a parameter set not the parameter set itself (except where real-valued individuals are used).
5. Genetic algorithms may provide a number of potential solutions to a given problem and the choice of the final is left up to the user.

5.2 Initializing the Population of the Genetic Algorithm

The Genetic Algorithm has to be initialized before the algorithm can proceed. The Initialization of the population size, variable bounds and the evaluation function are required. These are the initial input that are required in order for the Genetic Algorithm process to start.

The following code is based on the Genetic Algorithm Optimization Toolbox

(GAOT) .

```
%Initialising the genetic algorithm
populationSize=20;
variableBounds=[-100 100;-100 100;-100 100];
evalFN='PID_objfun_MSE';
%Change this to relevant object function
evalOps=[ ];
options=[1e-6 1];
initPop=initializega(populationSize,variableBounds,evalFN,
evalOps,options)
```

The following codes are used to initialize the GA. The codes will be explained in details.

- **Population Size** - The first stage of writing a Genetic Algorithm is to create a population. This command defines the population size.
- **Variable Bounds** - Since this project is using genetic algorithms to optimize the gains of a PID controller there are going to be three strings assigned to each member of the population, these members will be comprised of a P, I and a D string that will be evaluated throughout the course of the GA. The three terms are entered into the genetic algorithm via the declaration of a three row *variable bounds* matrix. The number of rows in the *variable bounds* matrix represents the number of terms in each member of the population.

- **EvalFN** - The evaluation function is the declaration of the file name containing the objective function.

- **Options** - Although the previous examples in this section were all binary encoded, this was just for illustrative purposes. Binary strings have two main drawbacks:

1. They take longer to evaluate due to the fact they have to be converted to/from binary.

2. Binary strings lose precision during conversion.

As a result of this and the fact that they use less memory, real (floating point) numbers will be used to encode the population. This is signified in the options command in section 5.2, where the '1e-6' term is the floating point precision and the '1' term indicates that real numbers are being used (0 indicates binary encoding is being used).

- **Initialisega** - This command combines all the previously described terms and creates an initial population of 20 real valued members between -100 and 100 with 6 decimal place precision.

5.3 Setting The GA Parameters

The following are codes for setting up the GA. The details of the code used will be explained below.

```
%Setting the parameters for the genetic algorithm
bounds=[-100 100;-100 100; -100 100];
evalFN='PID_objfun_MSE';%change this to relevant object function
evalOps=[ ];
startPop=initPop;
opts=[1e-6 1 0];
termFN='maxGenTerm';
termOps=100;
selectFN='normGeomSelect';
selectOps=0.08;
xOverFNs='arithXover';
xOverOps=4;
mutFNs='unifMutation';
mutOps=[8 100 2];
```

- **Bounds** - The variable bounds are for the genetic algorithm to search within a specified area. These bounds may be different from the ones used to initialize the population and they define the entire search space for the genetic algorithm.
- **startPop** - The starting population of the GA, 'startPop', is defined as the population described in the previous section, i.e. 'initPop', see 5.2.
- **opts** - The options for the Genetic Algorithm consist of the precision of the string values i.e. 1e-6, the declaration of real coded values, 1, and a request for the progress of the GA to be displayed, 1, or suppressed, 0.
- **TermFN** - This is the declaration of the termination function for the genetic algorithm. This is used to terminate the genetic algorithm once certain criteria has been met. In this project, every GA will be terminated when it reaches a certain number of generations using the 'maxGenTerm' function. This termination method allows for more control over the compile time that is the amount of time it takes for the genetic algorithm to reach its termination criterion of the genetic algorithm when compared with other termination criteria e.g. convergence termination criterion.
- **TermOps** - This command defines the options, if any, for the termination

function. In this example the termination options are set to 100, which means

that the GA will reproduce one hundred generations before terminating. This

number may be altered to best suit the convergence criteria of the genetic algorithm i.e. if the GA converges quickly then the termination options should

be reduced.

- **SelectFN** - Normalised geometric selection ('normGeomSelect') is the primary

selection process to be used in this project. The GAOT toolbox provides two

other selection functions, Tournament selection and Roulette wheel selection.

Tournament selection has a longer compilation time than the rest and as the

overall run time of the genetic algorithm is an issue, tournament selection will

not be used. The roulette wheel option is inappropriate due to the reasons mentioned in section 2.3.

- **SelectOps** - When using the ‘normGeomSelect’ option, the only parameter that

has to be declared is the probability of selecting the fittest chromosome of each generation, in this example this probability is set to 0.08.

- **XOverFN** - Arithmetic crossover was chosen as the crossover procedure. Single point crossover is too simplistic to work effectively on a chromosome

with three alleles, a more uniform crossover procedure throughout the chromosome is required. Heuristic crossover was discarded because it performs the crossover procedure a number of times and then picks the best one. This increases the compilation time of the program and is undesirable.

The Arithmetic crossover procedure is specifically used for floating point numbers and is the ideal crossover option for use in this project.

- **XOverOptions** - This is where the number of crossover points is specified. In the example shown in section 5.3, the number of crossovers points is set to four.
- **mutFNs** - The ‘multiNonUnifMutation’, or multi non-uniformly distributed

mutation operator, was chosen as the mutation operator as it is considered to

function well with multiple variables.

- **MutOps** - The mutation operator takes in three options when using the ‘multiNonUnifMutation’ function. The first is the total number of mutations,

normally set with a probability of around 0.1%. The second parameter is the maximum number of generations and the third parameter is the shape of the distribution. This last parameter is set to a value of two, three or four where the number reflects the variance of the distribution.

5.4 Performing The Genetic Algorithm

The genetic algorithm is compiled using the command shown below . The function ‘**ga.m**’ will evaluate and iterate the genetic algorithm until it fulfills the criteria described by it termination function.

```
%Performing the genetic algorithm
[x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,opts,...
termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFNs,mutOps);
```

Once the genetic algorithm is completed, the above function will return four variables:

x = The best population found during the GA.
 endPop = The GA.s final population.
 bestPop = The GA.s best solution tracked over generations.
 traceInfo = The best value and average value for each generation.

5.5 The Objective Function Of The Genetic Algorithm

This is the most challenging part of creating a genetic algorithm is writing the objective function.. In this project, the objective function is required to evaluate the best PID controller for the system. An objective function could be created to find a PID controller that gives the smallest overshoot, fastest rise time or quickest settling time. However in order to combine all of these objectives it was decided to design an objective function that will minimize the error of the controlled system instead. Each chromosome in the population is passed into the objective function one at a time. The chromosome is then evaluated and assigned a number to represent its fitness, the bigger its number the better its fitness. The genetic algorithm uses the chromosome's fitness value to create a new population consisting of the fittest members. Below are the codes for the Objective Function.

```
function [x_pop, fx_val]=PID_objfun_MSE(x_pop,options)
global sys_controlled
global time
global sysrl
% Splitting the chromosomes into 3 separate strings.
Kp=x_pop(2);
Ki=x_pop(3);
Kd=x_pop(1);
% creating the PID controller from current values
pid_den=[1 0];
pid_num=[Kd Kp Ki];
pid_sys=tf(pid_num,pid_den); % overall PID controller
```

Each chromosome consists of three separate strings constituting a P, I and D term, as defined by the 3-row shown in figure 5.1 . When the chromosome enters the evaluation function, it is split up into its three Terms. The P, I and D gains are used to create a PID controller according to the equation below.

$$G_{PID} = \frac{K_d s^2 + K_p s + K_i}{s}$$

K_d	K_p	K_i
-------	-------	-------

Fig.5.1 . Chromosome Definition

The newly formed PID controller is placed in a unity feedback loop with the system transfer function shown in figure 5.2. This will result in a reduce of the compilation time of the program. The system transfer function is defined in another file and imported as a global variable. The controlled system is then given a step input and the error is assessed using an error performance criterion such as Mean Square Error or in short MSE. The MSE is an accepted measure of control and of quality but its practical use as a measure of quality is somehow limited [6]. The chromosome is assigned an overall fitness value according to the magnitude of the error, the smaller the error the larger the fitness value. Below is the codes and the formula used to implement the MSE Objective Function.

- **Mean Square Error (MSE)**

$$I_{MSE} = \frac{1}{n} \sum_{i=1}^n (e(t))^2$$

```

%Calculating the MSE
for i=1: size(t)
error(i) = 1-y(i);
end
error_sq = error * error' ;
MSE=error_sq/max(size(error));

```

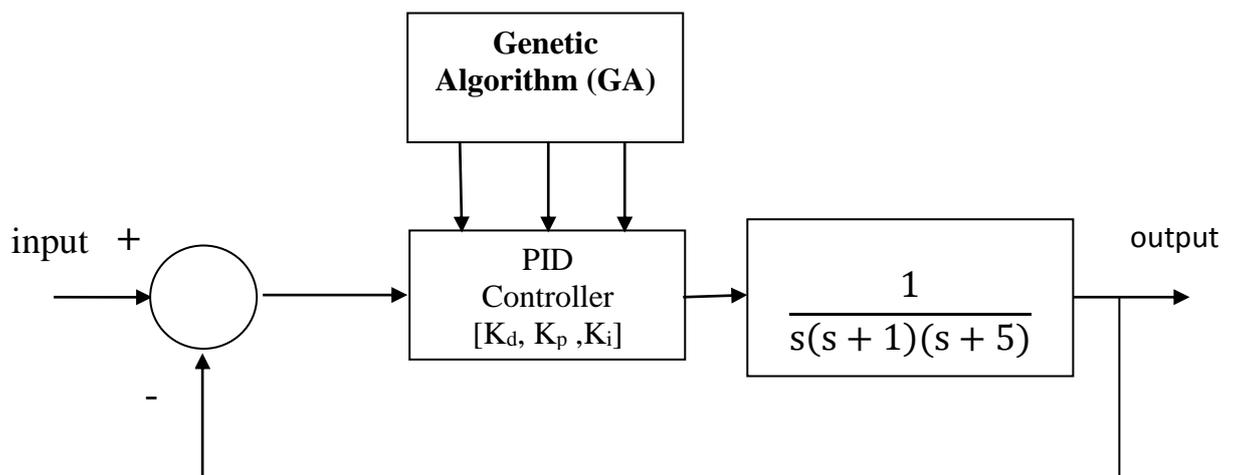


Figure 5.2. GA-PID Controller Design For the turbine control system.

Additional code was added to ensure that the genetic algorithm converges to a controller that produces a stable system. The code, shown in Figure below , assesses the poles of the controlled system and if they are found to be unstable that is on the right half of the s-plane, the error is assigned

an extremely large value to make sure that the chromosome is not reselected.

```
%Ensuring controlled system is stable
poles=pole(sys_controlled);
if poles(1)>0
MSE=10e5;
elseif poles(2)>0
MSE=10e5;
elseif poles(3)>0
MSE=10e5;
elseif poles(4)>0
MSE=10e5;
end
fx_val=MSE;
```

5.6 Results Of The Implemented Genetic Algorithm PID Controller

In the following section, the results of the implemented GA- PID Controller will be analyzed. The GA- PID controller was initialized with population size of 20&40. The response of the GA- PID will then be analyzed for the smallest overshoot, fastest rise time and the fastest settling time. The best response will then be selected. From the following responses, the GA- PID will be compared to the Ziegler-Nichols method. The superiority of GA against the Z-N method will be shown. The plot of the GA designed PID with the population size of 20 is shown in figure 5.3.

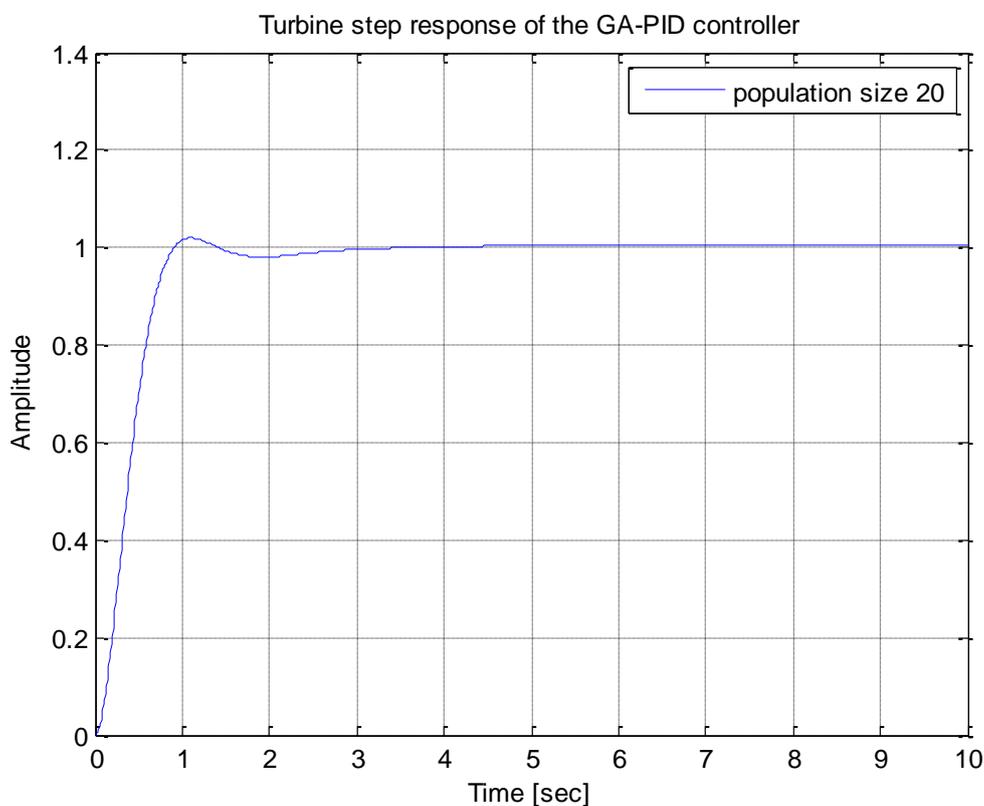


Figure 5.3 PID Response With Population Size Of 20.

The details of the system response will be analyzed. The response of system looks reasonable stable. The overshoot of the response is 1.98 %. The settling time of the response is 0.837 seconds and finally the response of the rise time is 0.56 seconds.

The closed loop poles are

$$P_1 = -2.630651198980039 + 2.984859824251561i$$

$$P_2 = -2.630651198980039 - 2.984859824251561i$$

$$P_3 = -0.721973109485876$$

$$P_4 = -0.016724492554048$$

The convergence curve for each gain is called as particle for K_p , K_i and K_d is plotted to give an idea how the GA converged to its final value has been illustrated in Figures 5.4 is the outcome of convergence curve for output response.

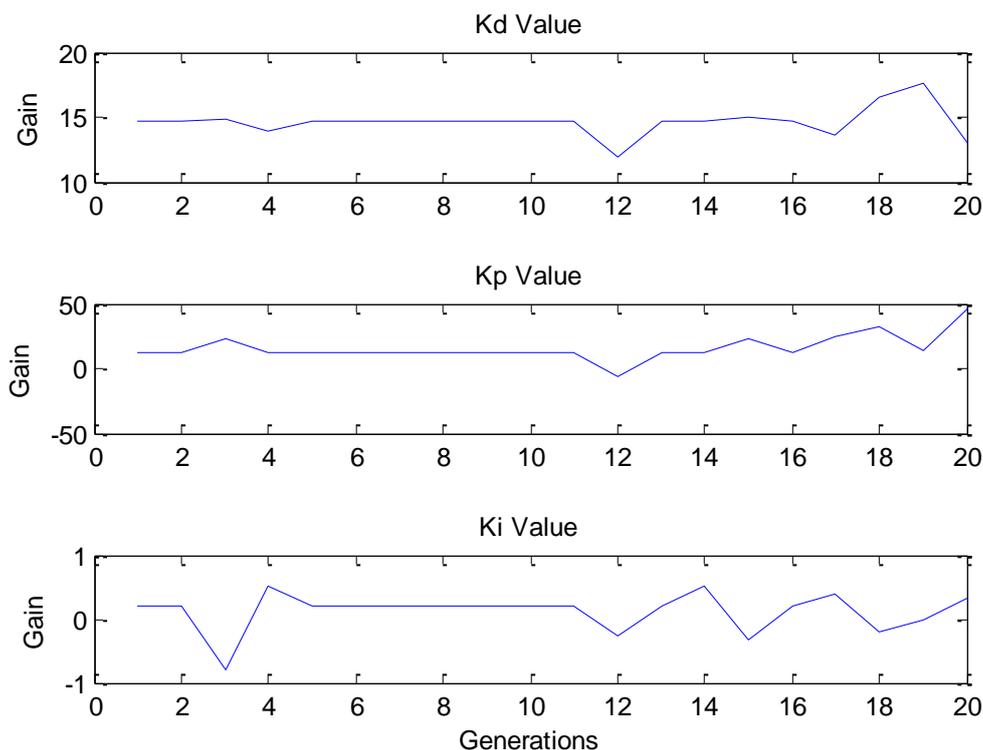


Figure 5.4 Illustration Of Genetic Algorithm Converging Through Generations.

Figure 5.5 depict the response of GA designed PID with the population size of 40.

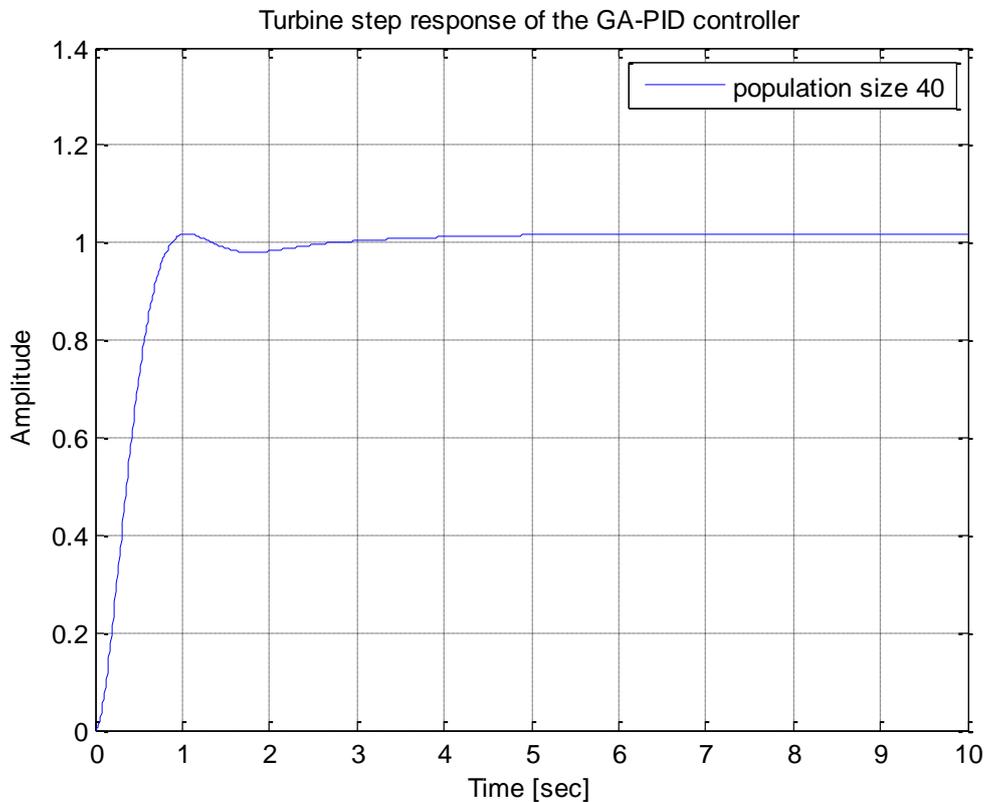


Figure 5.5 PID Response With Population Size Of 40.

The details of the system response will be analyzed. The overshoot of the response is 1.85% The settling time of the response is 0.812 seconds and finally the response of the rise time is 0.543seconds . The closed loop poles are

$$P_1 = -2.662661360979914 + 3.121811944967547i$$

$$P_2 = -2.662661360979914 - 3.121811944967547i$$

$$P_3 = -0.617779255034797$$

$$P_4 = -0.056898023005371$$

The convergence curve for each gain is called as particle for K_p , K_i and K_d is plotted to give an idea how the GA converged to its final value has been illustrated in Figures 5.6 is the outcome of convergence curve for output response.

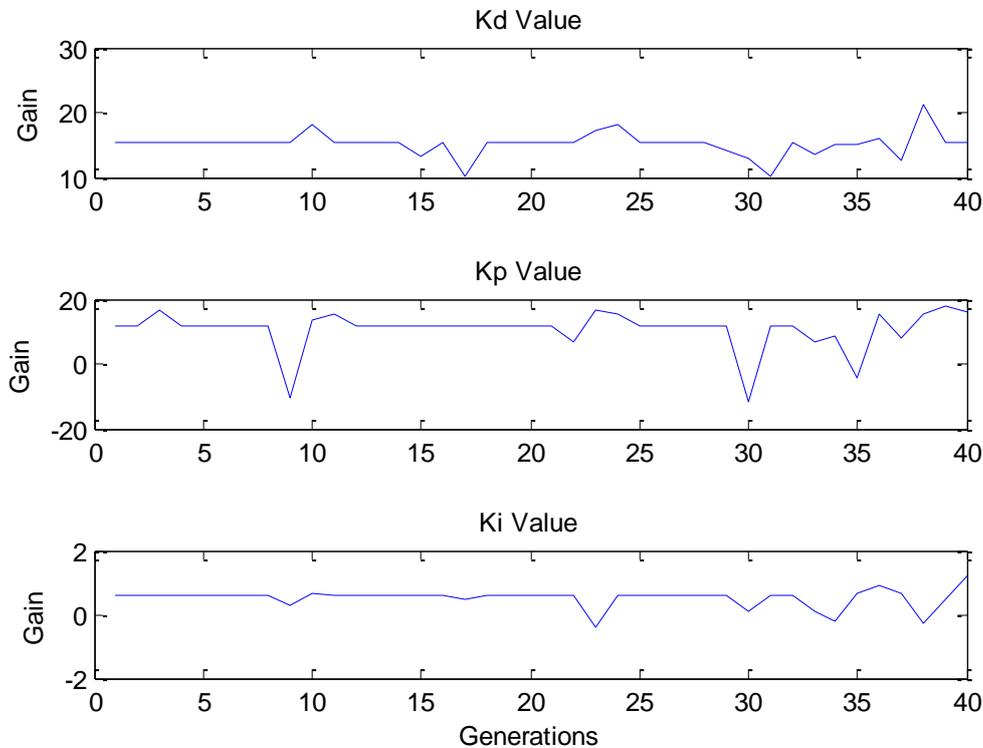


Figure 5.6 Illustration Of Genetic Algorithm Converging Through Generations.

The analysis of both controllers are shown in table 3.

Table 3.Result Of Classical Z-N And Genetic PID Controller.

Type of Controller	K_d	K_p	K_i	M_p %	T_s (sec)	T_r (sec)	Population size
Z-N.	10.53	18	21.35231	61.9	10	0.578	
Opt.Z-N	30.3139	39.42	12.811	26.5	3.65	0.286	
GA-PID	14.7283	11.7569	0.191138	1.98	0.837	0.56	20
	15.4635	11.5457	0.591774	1.85	0.812	0.543	40

From table 3 the system response of the GA-PID controller with population size of 40 have the best response as compared to the others responses. Proceeding with the higher population size will take up a lot of computer memory space.

Chapter 6

Conclusions And Further Works

6.1 Conclusions

In conclusion the responses as shown in chapter 5, had showed to us that the designed PID with GA has much faster response than using the classical method. The classical method is good for giving us as the starting point of what are the PID values. However as shown in chapter 4, the approached in deriving the initial PID values using classical method is rather troublesome. There are many steps and also by trial and error in getting the PID values before you can narrow down in getting close to the .optimized. values. An optimized algorithm was implemented in the system to see and study how the system response is. However the GA designed PID is much better in terms of the rise time and the settling time.

This project has exposed me to various PID control strategies. It has increased my knowledge in Control Engineering and Genetic Algorithm in specific. It has also shown me that there are numerous methods of PID tunings available in the academics and industrial fields. Previously I was comfortable with Z-N classical methods but now I would like to venture into others methods available .However this depends on the opportunity in my work place.

Finally this project has made me more appreciative of the Control Engineering and its contribution to the improvement of the industrial and society. The fact is, in every aspect of our life, Control Engineering is always with us. Let it be in our room, in our car or even in the complex application of the Biomedical field. As our life improves with more

automated system available in our daily life, be conscious that the background of these happening is the working of control engineering.

6.2 Further Works

It is hope that this project can be improved to include the implementation of tuning the PID controller via GA in an online environment. This will have much impact in the optimization of the system under control .As for the subject under study, if the plant or the turbine system can be tuned using GA in an online environment, there will be minimum losses on the process. The steam used to drive the turbine will be fully utilized and the energy transferred maximized. There will be minimum loss since the response shown above is as close to the unit step. Hence in the refineries, this will translate to better profit margin. It is also hope that in the future works, the system to be tested with other Performance Criterion such as IAE, ITAE and ISE. In the current work, due to time constraint, only MSE as the Performance Criterion was implemented. Hence it was not known and proven that the MSE will result a better fitness function measurement.

References

[1] Katsuhiko Ogata, Modern Control Engineering, Upper Saddle River, NJ, Prentice Hall, 2002.

[2] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini, Feedback Control of Dynamic Systems, Upper Saddle River, NJ, Prentice Hall, 2010

[3] Astrom, K., T. Hagglund, .PID Controllers; Theory, Design and Tuning., Instrument Society of America, Research Triangle Park, 1995.

[4] Whitley, A Genetic Algorithm Tutorial, Technical Report CS-93-103, Dept. of Computer Science, Colorado State University, 1993

[5] Liu Fan, Er Meng Joo” Design for Auto-tuning PID Controller Based on Genetic Algorithms” Nanyang Technological University Singapore IEEE Trans on ICIEA 2009

[6] Naeem, W., Sutton, R. Chudley. J, Dalglish, F.R. and Tetlow, S., . An Online Genetic Algorithm Based Model Predictive Control Autopilot Design With Experimental Verification.. International Journal Of Control, Vol 78, No. 14, pg 1076 . 1090, September 2005.

[7] Skogestad, S., .Probably The Best Simple PID Tuning Rules In The World.. Journal Of Process Control, September 2001.

Appendix

Matlab Script and Function

Files

Initial_PID_GA.m

```
%%%%%%%%%% Iterating the genetic algorithm
clear
clc
format long;
options = gaoptimset('PlotFcns',...
    {@gaplotbestf});
[X,endPop,bPop,traceInfo] = ga(@PoleAssign,3);
t=0:0.1:10;
y=PoleAssignY(X,t);
fprintf(' Kd = %g \n',X(1));
fprintf(' Kp = %g \n',X(2));
fprintf(' Ki = %g \n',X(3));
figure,plot(t,y);
xlabel('Time [sec]')
ylabel(' Amplitude')
legend('Genetic PID controlller ')
title(' Turbine step response of the Genetic PID controller ')
grid

%%%%%%%%%%Plotting Genetic algorithm controller %%%%%%%%%%%
function y=PoleAssignY(X,t)
```

```

Kd=X(1);
Kp=X(2);
Ki=X(3);
n = [Kd Kp Ki] *1;
d = [ 1 6 5 0 0];
s = tf(n,d);
s1 = feedback(s,1);
y = step(s1,t);
%%%%% PID_objfun_mse.m
%%%%%%%%%
%%%%%%%%%
%
function F=PoleAssign(X)
Kd=X(1);
Kp=X(2);
Ki=X(3);
t =0:0.1:10;
n = [Kd Kp Ki] *1;
d = [ 1 6 5 0 0];
s = tf(n,d);
s1 = feedback(s,1);
S = stepinfo(s1)
y = step(s1,t);
for i=1:size(t);
error(i)=1-y(i);
end
%Calculating the MSE
error_sq = error*error';

```

```
MSE=error_sq/max(size(error));
```

```
%
```

```
—
```

```
%Ensuring controlled system is stable
```

```
poles=pole(s1);
```

```
if poles(1)>0
```

```
MSE=10e5;
```

```
elseif poles(2)>0
```

```
MSE=10e5;
```

```
elseif poles(3)>0
```

```
MSE=10e5;
```

```
elseif poles(4)>0
```

```
MSE=10e5;
```

```
end
```

```
F=MSE;
```