A Major Project Report On

"TUNING OF PID CONTROLLER USING GENETIC ALGORITHM"

Submitted in Partial Fulfillment of Requirements

for the Award of the Degree of

MASTER OF ENGINEERING

(CONTROL & INSTRUMENTATION)

By

IVAN CAROL BHENGRA

Roll No. 13801

Under the Supervision of

SH. RAM BHAGAT

Department of Electrical Engineering

Delhi College of Engineering

Bawana Road, Delhi-110042

2012

A

Major Project Report

On

"TUNING OF PID CONTROLLER USING GENETIC ALGORITHM"

Submitted in Partial Fulfillment of Requirements

for the Award of the Degree of

MASTER OF ENGINEERING

(CONTROL & INSTRUMENTATION)

By

IVAN CAROL BHENGRA

Roll No. 13801

Under the Supervision of

SH. RAM BHAGAT

Department of Electrical Engineering

Delhi College of Engineering

Bawana Road, Delhi-110042

2012

<u>CERTIFICATE</u>

It is certified that **Mr. IVAN CAROL BHENGRA**, Roll No. 13801, student of M.E. (Control & Instrumentation), Department of Electrical Engineering, Delhi College of Engineering, has submitted the dissertation entitled **"TUNING OF PID CONTROLLER"** under our guidance towards partial fulfillment of the requirements for the award of the degree of Master of Engineering (Control & Instrumentation).

This dissertation is carried out by him under my guidance and supervision. His work is found to be outstanding and has not been done earlier.

I wish him success in all his endeavours.

Sh. Ram Bhagat

Electrical Engineering Department

Delhi College of Engineering

<u>ACKNOWLEDGEMENT</u>

Place: DCE, Delhi   **IVAN CAROL BHENGRA**

Contents

List of Figures

List of Tables

Abstract

It is interesting to note that more than half of the industrial controllers in use today utilize PID or modified PID control scheme. Most PID controllers are adjusted on-site, many different type of tuning rules have been proposed in the literature. Using these tuning rules, delicate and fine tuning of PID controllers can be made on site. Also, automatic tuning methods have been developed and some of the PID controllers may possess on-line automatic tuning capabilities. Modified forms of PID control, such as I-PD control and two-degrees-of-freedom PID control, are currently in use in industry. Many practical methods for bump less switching (from manual operation to automatic operation) and gain scheduling are commercially available. Hence how do we optimize the PID controller?

The usefulness of PID controls lies in their general applicability to most control systems. In particular, when the mathematical model of the plant is not known and therefore analytical design methods cannot be used, PID controls prove to be most useful. In the field of process control systems, it is well known that the basic and modified PID control schemes have proved their usefulness in providing satisfactory control, although in many given situations they may not provide optimal control.

In this dissertation, it is proposed that the controller be tuned using the Genetic Algorithm technique. The Genetic Algorithm (GA) is an optimization and stochastic search technique based on the principles of genetics and natural selection. The Genetic Algorithms (GAs) are a stochastic global search method that emulates the process of natural evolution. The Genetic Algorithm starts with no knowledge of the correct solution and depends entirely on responses from its environment and evolution operators (i.e. reproduction, crossover and mutation) to arrive at the best solution. Genetic Algorithms have been shown to be capable of locating high performance areas in complex domains without experiencing the difficulties associated with high dimensionality or false optima as may occur with gradient decent techniques. 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 Genetic Algorithm method. The results of both designs will be compared, analyzed and conclusion will be drawn out.

Chapter-1

INTRODUCTION

1.1 Project Aims and Objectives

The aim of this project is to design a plant using Genetic Algorithm. What is Genetic Algorithm? Genetic Algorithm or in short GA is a stochastic algorithm based on principles of natural selection and genetics. Genetic Algorithms (GAs) are a stochastic global search method that mimics 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 or false optima as may occur with gradient decent techniques. Using genetic algorithms to perform the tuning of the controller will result in the optimum controller being evaluated for the system every time.

The objective of this project is to show that by employing the GA method of tuning a plant, an optimization can be achieved. This can be seen by comparing the result of the GA optimized plant against the classically tuned plant.

1.2 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.
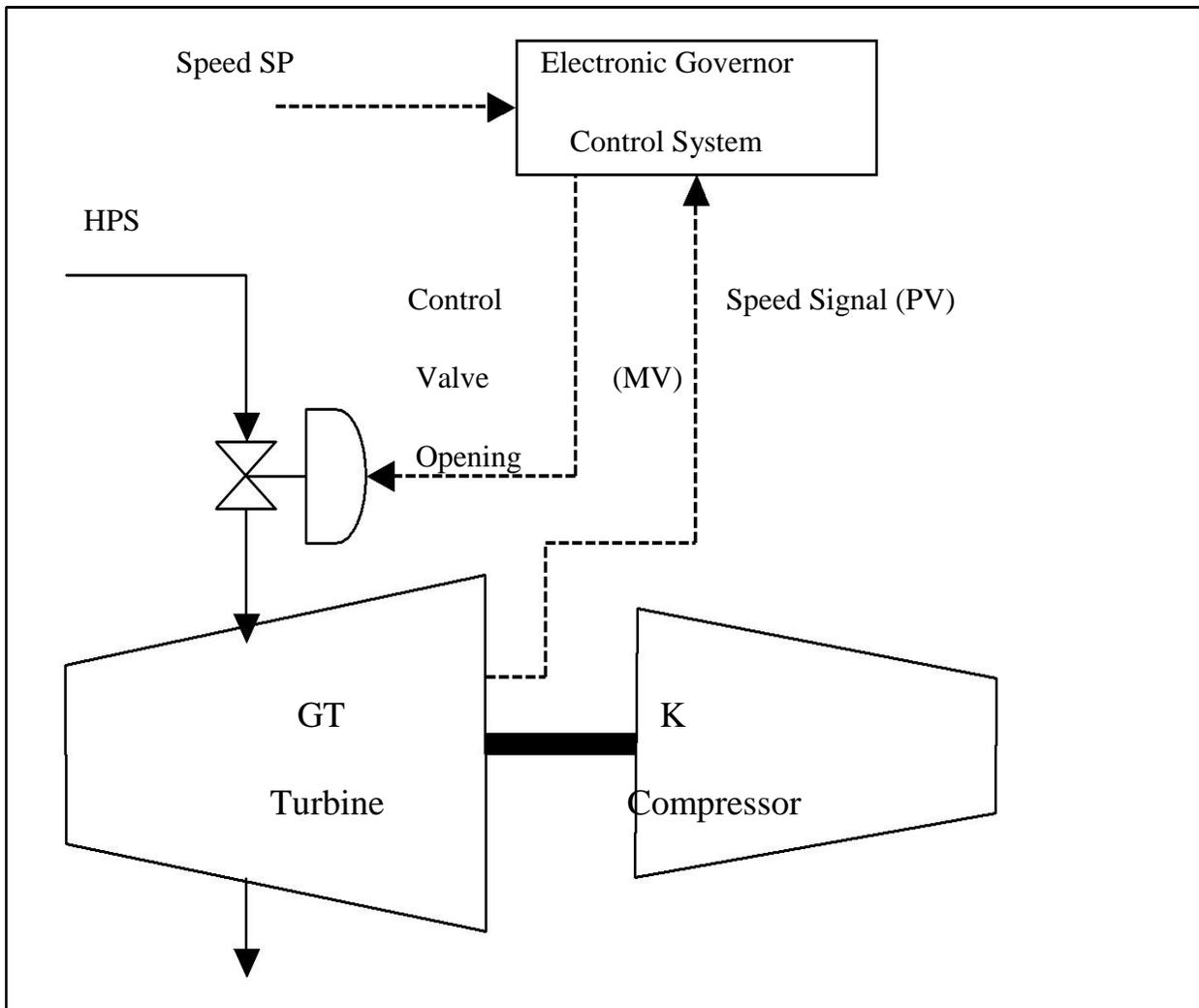
Figure 1.1. Typical Turbine Speed Control

In the above 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.

It is a known fact that the PID controller is employed in every facet of industrial automation. The application of PID controller spans 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 has 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 above in figure I. 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.

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 areal-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 call edits 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 operation 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. 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:

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.

1.3 Literatures Reviews

The followings are the few books and papers that were referred to, in the process of undertaking this project. For the undertaking of his project, thorough reading of Genetic Algorithm is required before the project can commences. Hence a comprehensive research for resources is required and the following are some of the literatures that have somehow contributed to my understanding of the control system and the genetic algorithm in specific.

Books

- David E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning. The University of Alabama, Addison-Wesley Publishing Company Inc, 1989.

- John Leis, Digital Signal Processing, A MATLAB-Based Tutorial Approach, University of Southern Queensland, Research Studies Press Limited, 2002.

- K. Astrom and T Hagglund, PlD Controllers: Theory, Design and Tuning, Prentice Hall, 1987.

- K. Ogata, Discrete-Time Control Systems, University of Minnesota, Prentice Hall, 1984.

- K. Ogata, Modern Control Engineering, University of Minnesota, Prentice Hall, 2007.

- S.K.Singh, Computer-Aided Process Control, Automation Division, Tata Steel, Jamshedpur, Prentice Hall of India Private Limited, 2007.

- B. Wayne Bequette, Process Control Modeling, Design and Simulation, Rensselaer Polytechnic Institute, Prentice Hall of India Private Limited, 2006.

Journals

- T O. Mahony, C J Downing and K Fatla, Genetic Algorithm for PlD Parameter Optimization: Minimizing Error Criteria, Process Control and Instrumentation 2000 26-28 July 2000, University of Stracthclyde, pg 148- 153.

- Chipper field, A. J., Fleming, P. J. Pohlheim, H. and Fonseca, C. M., A Genetic Algorithm Toolbox for MATLAB, Proc. International Conference on Systems Engineering, Coventry, UK, 6-8 September, 1994.

- Q Wang, P Spronck and R Tracht, An Overview Of Genetic Algorithms Applied To Control Engineering Problems, Proceedings of the Second International Conference on Machine Learning And Cybernetics, 2003.

- K. Krishna Kumar and D. E. Goldberg, Control System Optimization Using Genetic Algorithms, journal of Guidance, Control and Dynamics, Vol. 15,No. 3, pp. 735-740, 1993.

- A. Varsek, T. Urbacic and B. Filipic, Genetic Algorithms in Controller Design and Tuning, IEEE Trans. Sys. Man and Cyber, Vol. 23, No. 5, PP1330-1339, 1994.

From the reading of the above and not inclusive, it is found that GAs are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding acceptably good solutions to problems in acceptably quickly Where specialized techniques exist for solving particular problems, they are likely to out-perform GAs in both speed and accuracy of the final result, so there is no black magic in evolutionary computation.

Therefore GAs should be used when there is no other known efficient problem solving strategy.

However in this project, GA is still use as the preferred optimized method in optimizing the turbine speed control system. You will see that some other optimization method can be better in certain areas of application and GA can be better in another application. Hence there is no fast and quick rule to which optimization methods to use. It all depends on application and the complication in the implementation of the optimized algorithm.

Chapter 2

GENETIC ALGORITHM

2.1 Introduction

Genetic Algorithms (GAs) 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 have 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.

In this way, GAs have been shown to be capable of locating high performance areas in complex domains without experiencing the difficulties associated with high dimensionality, as may occur with gradient decent techniques or methods that rely on derivative information.

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 Chromosome 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 chromosome.

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.

2.3 Population Size

Determining the number of population is the one of the important step in GA. There are many research papers that dwell in the subject. Many theories have been documented and experiments recorded.

However the matter of the fact is that more and more theories and experiments are conducted and tested and there is no fast and thumb rule with regards to which is the best method to adopt. For a long time the decision on the population size is based on trial and error.

In this project the approach in determining the population is rather unsciencetific. From my reading of various papers, it suggested that the safe population size is from 30 to 100. In this project initial

populations of 20 were used and the result observed. The result was not promising. Hence initiatives of 40, 80 and 90 size of population were experimented. It was observed that the population of 80 seems to be a good guess. Population of 90 and above does not results in any further optimization.

2.4 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 as shown in 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, it is not implausible for the weakest string (01001) to dominate the selection process.
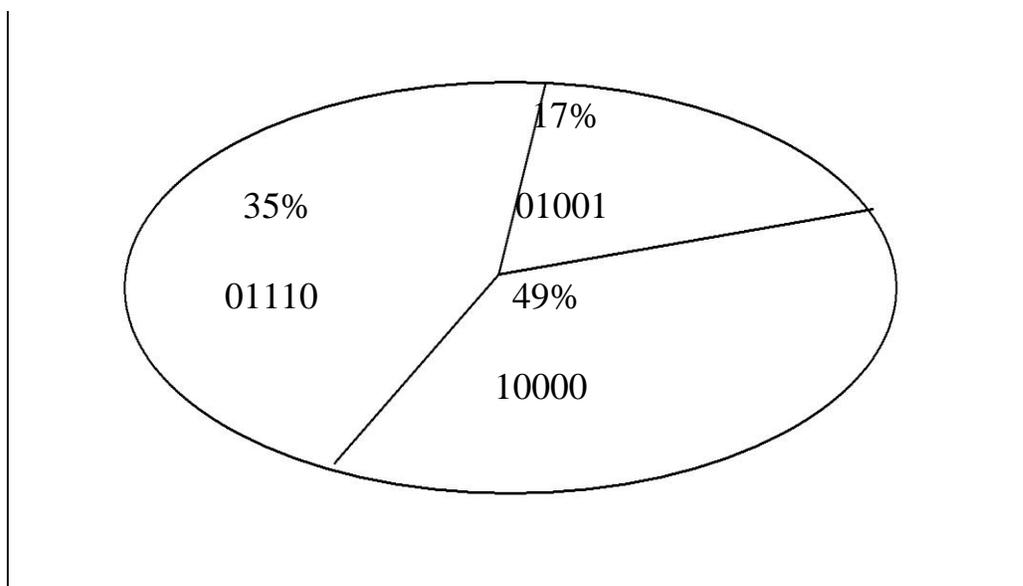
Figure 2.1. Depiction of Roulette Wheel Selection

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 that is giving fitter chromosomes a larger probability of selection.

Four common methods for selection are:

1.   Roulette Wheel selection

2.   Stochastic Universal sampling

3.   Normalized geometric selection

4.   Tournament selection

Due to the complexities of the other methods, the Roulette Wheel method is preferred in this project.

2.5 Crossover

Once the selection process is completed, the crossover algorithm is initiated. The crossover operation 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:

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.

```
100 │ 00 ─────────────► 10010
011 │ 10              01100
```
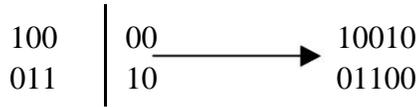
Figure 2.2. Illustration of Crossover.

More complex crossover techniques exist in the form of Multi-point and Uniform Crossover Algorithms. In Multi-point crossover, it 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.

```
11│11│ 1 ─────────────► 11001
00│00│ 0              00110
```
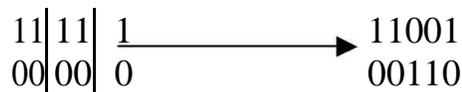
Figure 2.3. Illustration of Multi-Point Crossover.

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.

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 figure 2.4.

```
10101                    11111

        ──────▶10101──────▶

01010                    00000
```
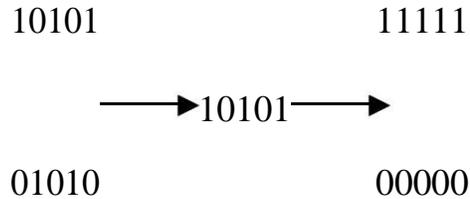
Figure 2.4. Illustration of a Uniform Crossover.

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 and Multi-Point Crossover is the preferred choice.

2.6 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 Genetic Algorithm searches the entire problem space.

2. The Genetic Algorithm 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 Genetic Algorithm. 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 values55represent 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.

10000 ———————— 10010

Figure 2.5. Illustration of Mutation Operation

2.7 Summary of 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.

```
┌─────────────────────────────────────────────────────────┐
│                                                           │
│        ┌────────────────────┐                             │
│        │  Create/Initialize │                             │
│        │     Population      │                             │
│        └────────────────────┘                             │
│                  │                                         │
│                  ▼                                         │
│        ┌────────────────────┐◄──────────────────┐         │
│        │  Measure/Evaluate  │                    │         │
│        │      Fitness       │                    │         │
│        └────────────────────┘                    │         │
│                  │                               │         │
│                  ▼                               │         │
│        ┌────────────────────┐                    │         │
│        │    Select Fittest  │                    │         │
│        └────────────────────┘                    │         │
│                  │                               │         │
│                  ▼                               │         │
│        ┌────────────────────┐                    │         │
│        │      Mutation      │                    │         │
│        └────────────────────┘                    │         │
│                  │                               │         │
│                  ▼                               │         │
│        ┌────────────────────┐     ┌──────────────────┐    │
│        │Crossover/Production│────►│   Non Optimum    │    │
│        └────────────────────┘     │    Solution      │    │
│                  │                └──────────────────┘    │
│                  ▼                                         │
│        ┌────────────────────┐                             │
│        │  Optimum Solution  │                             │
│        └────────────────────┘                             │
│                                                           │
└─────────────────────────────────────────────────────────┘
```
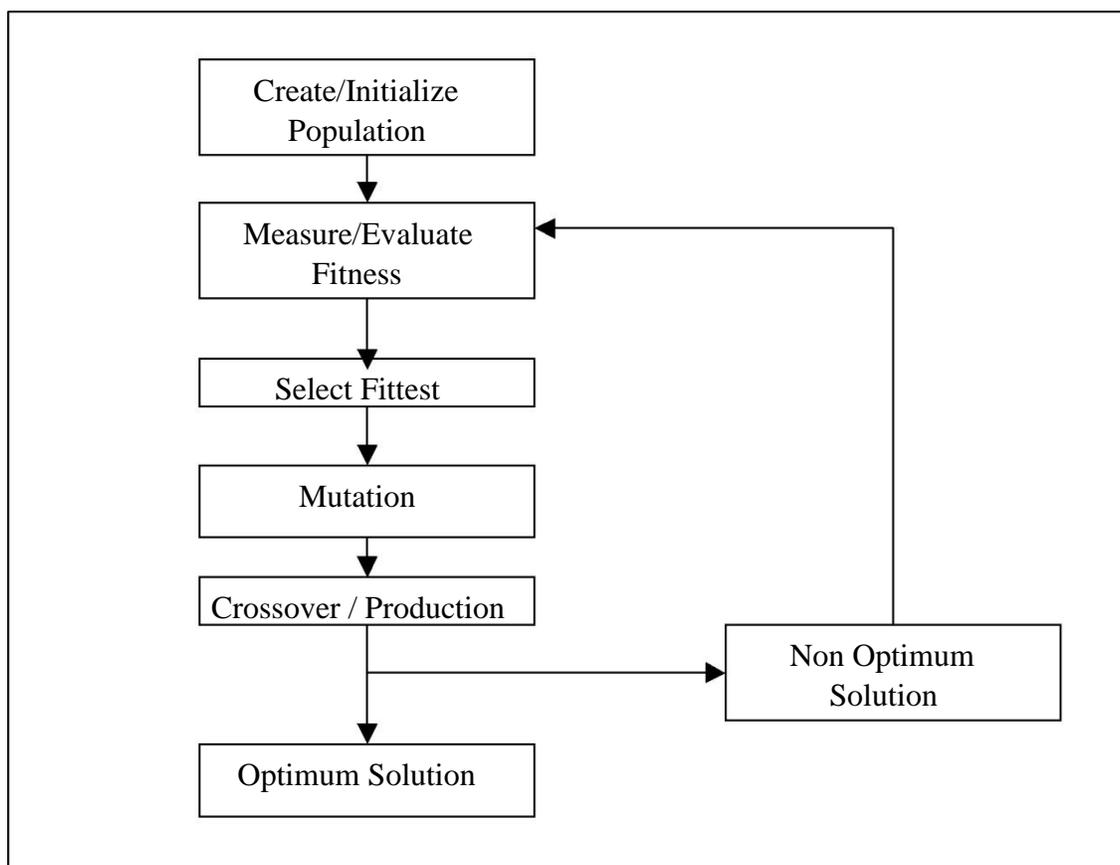
Figure 2.6. Genetic Algorithm Process Flowchart

The steps involved in creating and implementing a genetic algorithm:

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 conditions for example the maximum number of generations or when the string fitness value exceeds a certain threshold.

2.8 Elitism

In the process of the crossover and mutation-taking place, there is high chance that the optimum solution could be lost. There is no guarantee that these operators will preserve the fittest string. To avoid this, the elitist models are often used in this model; the best individual from a population is saved before any of these operations take place. When a new population is formed and evaluated, this model will examine to see if this best structure has been preserved, if not the saved copy is reinserted into the population. The GA will then continue on as normal.

2.9 Objective Function or Fitness Function

The objective function is used to provide a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the most fit individuals will have the lowest numerical value of the associated objective function. This raw measure of fitness is usually only used as an intermediate stage in determining the relative performance of individuals in a GA.

Another function that is the fitness function is normally used to transform the objective function value into a measure of relative fitness, thus where f is the objective function, g transforms the value of the objective function to a non negative number and F is the resulting relative fitness. This mapping is always necessary when the objective function is to be minimized as the lower objective function values correspond to litter individuals. In many cases, the fitness function value corresponds to the number of offspring that an individual can expect to produce in the next generation. A commonly used transformation is that of proportional fitness assignment.

2.10 Application of Genetic Algorithms ln Control Engineering

Presently GA has been receiving a lot of attention and more research has been done to study its applications. Application in the area of Control Engineering has also developed tremendously. Even though in control system design, issues such as performance, system stability, static and dynamic index and system robustness have to be taken into account. However each of these issues strongly depends on the controller structure and parameters. This dependence usually cannot be expressed in a mathematical formula but often a trade-off has to be made among conflicting performance issues.

The following are some GA applications in use in control engineering:-

- Multi objective Control.

- PID control.

- Optimal Control.

- Robust Control.

- Intelligent Control.

Chapter 3

PID Controller

3.1   Introduction

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

PID controllers' algorithms are mostly used in feedback loops. PID controllers can be implemented in many forms. It can be 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 schematic diagram illustrating the PID controller. Such set up is known as non-interacting form or parallel form.
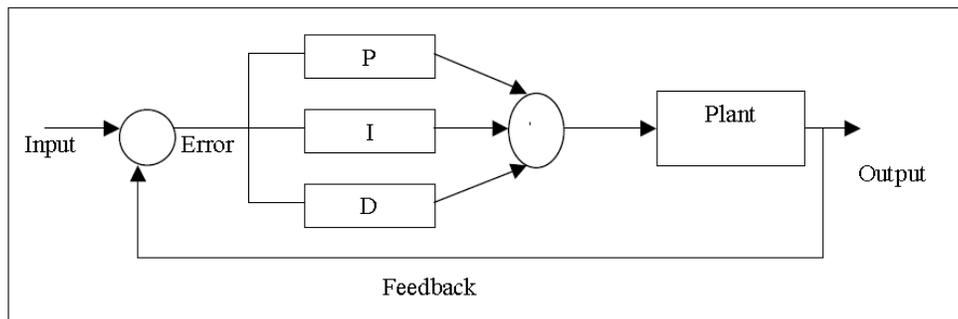

Figure 3.1. Schematic Diagram of the PID Controller, Non-Interacting Form

3.2 PID Controller

In Proportional control,

$P_{term} = K_p \times Error$

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

In Integral control,

$I_{term} = K_I \times \int Error\ dt.$

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 earlier on by the P-action.

In Derivative control,

$D_{term} = K_D \times \dfrac{d(Error)}{dt}$

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.

3.3 Continuous PID

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

$$G_c(s) = K_p(1 + \tfrac{1}{sT_i} + sT_d)$$

This can be illustrated below in the following block diagram.
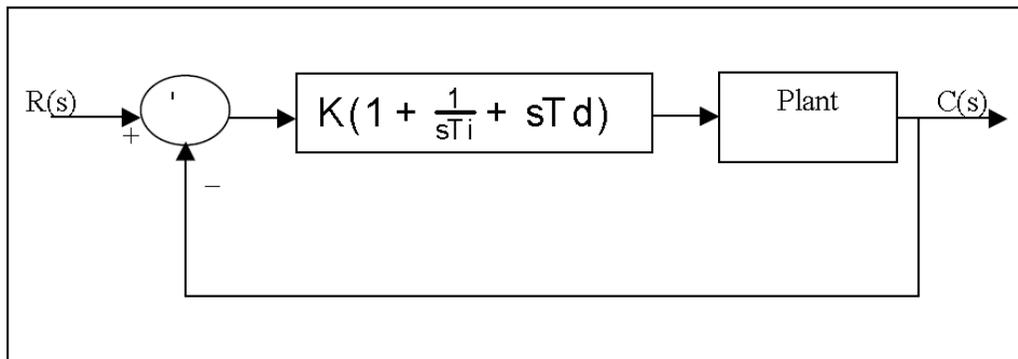

Figure 3.2. Block Diagram of Continuous PID Controller.

What the PID controller does is basically is to act on the variable to be manipulated through a proper combination of the three control 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 of the 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 have been discovered in the recent years.

Chapter 4

Tuning of PID controller

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 increased 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.

4.2   Designing PID Parameters

From the response below, the system under study is indeed oscillatory and hence the Z-N tuning rule based on critical gain K_{cr} and critical period P_{cr} can be applied.
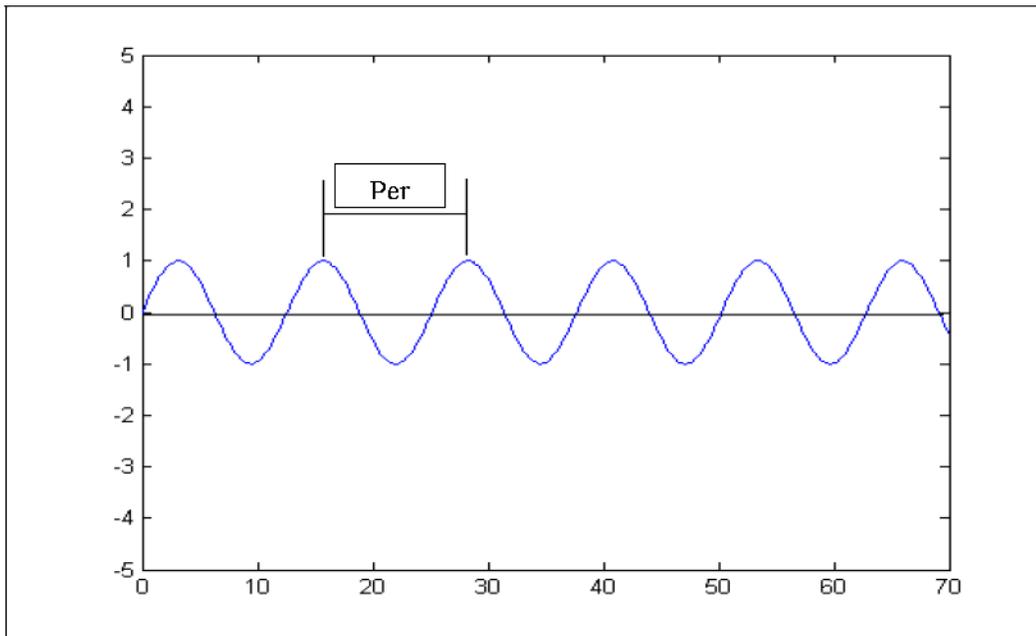
Figure 4.1. Illustration of Sustained Oscillation with Period P$_{cr}$.

The transfer function of the PID controller is

$$G_c(s) = K_p(1 + \frac{1}{sT_i} + sT_d)$$

The objective is to achieve a unit-step response curve of the designed system that exhibits a maximum overshoot of 25%. If the maximum overshoot is excessive says about greater than 40%, fine tuning should be done to reduce it to less than 25%.

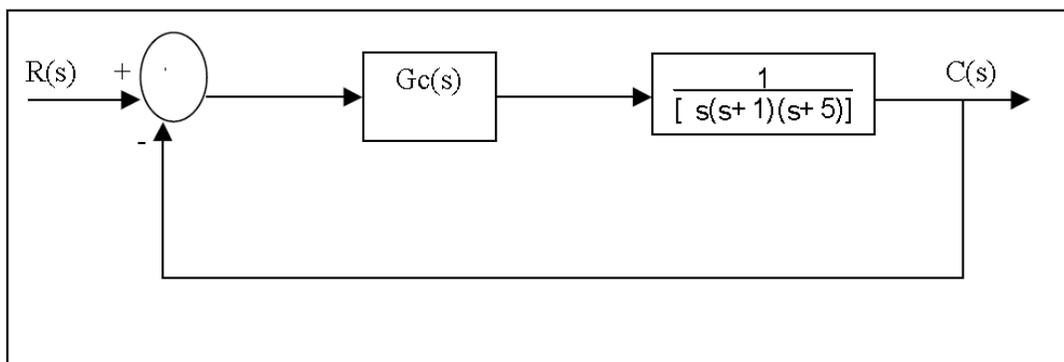The system under study above has a following block diagram

Figure 4.2. Block Diagram of Controller and Plant.

Since the $T_i = \infty$ 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 be determined.

The table below illustrates the Routh array.

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

Table 1. Routh Array

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 jω term. Hence the new equation becomes

$$(j\omega)^3 + 6(j\omega)^2 + 5(j\omega) + 30 = 0$$

This can be simplified to

$$5(5-\omega^2) + j\omega(5-\omega^2) = 0$$

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

$$\omega^2 = 5$$

or $$\omega = \sqrt{5}$$

The period of the sustained oscillation can be calculated as

$$P_{cr} = 2\pi/\sqrt{5}$$
$$= 2.8099$$

From Ziegler-Nichols frequency method of the second method, the table 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.

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}

Table 2. Recommended PID Value Setting.

Hence from the above table, the values of the PID parameters K$_p$, T$_i$ and T$_d$ will be

K$_p$ = 30

T$_i$= 0.5 X 2.8099

   = 1.405

T$_d$= 0.125 X 2.8099

   = 0.351

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

$$G_c(s) = K_p(1 + \tfrac{1}{sT_i} + sT_d)$$

$$= 18(1 + \frac{1}{1.405s} + 0.35124s)$$

$$= \frac{6.3223(s + 1.4235)^2}{s}$$

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 as follows:-
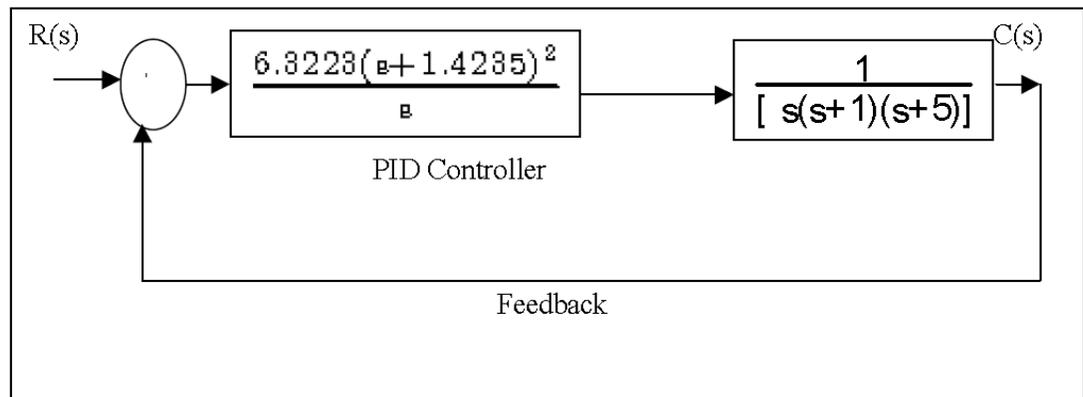

Figure 4.3. Illustration of the Close Loop Transfer Function

Using the MATLAB function, the following system can be easily calculated. The above system can be reduced to single block by using the following MATLAB function. Below are 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
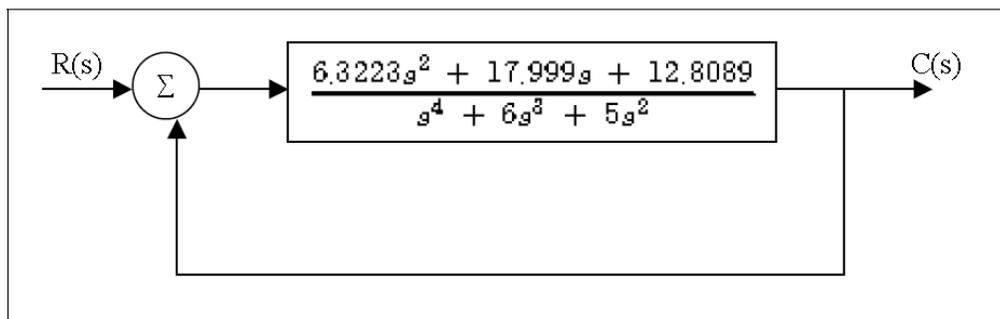
Hence the above block diagram is reduced to,

Figure 4.4. Simplified System

Using another MATLAB function, the overall function with its feedback can be calculated as follows:-

```matlab
% 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:-

```
num/den =

      6.3223 s^2 + 17.999 s + 12.8089

  ---------------------------------------------

   s^4 + 6 s^3 + 11.3223 s^2 + 17.999 s + 12.8089
```

Therefore the overall close loop system response of

$$\frac{C(s)}{R(s)} = \frac{6.3226s^2 + 17.999s^2 + 12.808}{s^4 + 6s^3 + 11.3223s^2 + 18s + 12.8089}$$

The unit step response of this system can be obtained with MATLAB,

```matlab
%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 Design System')
```
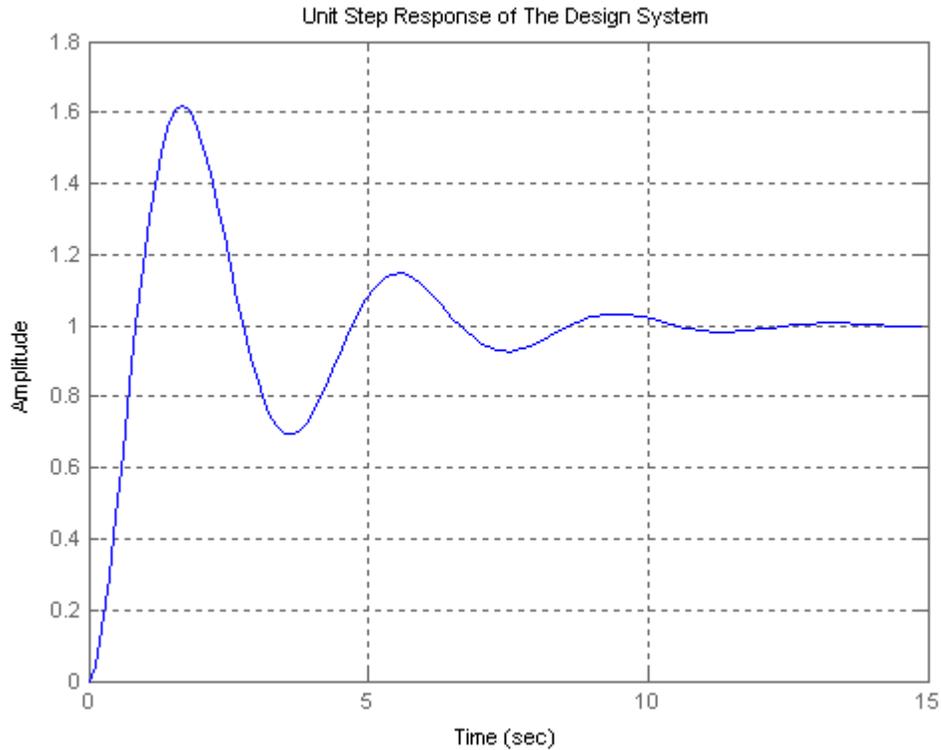
Figure 4.5. Unit Step Response of the Designed System

The figure above is the system response of the designed system. From the above response it is obvious that the system can be further improved.

4.3   Analysis of the Classically Designed Controller

From the above diagram, we can analyze 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: The delay time is the time required for the response to reach half the final value the very first time.

- Rise time, t_r: The rise time is the time required for the response to rise from 10% to 90%, 5% to 95% or 0% to 100% of its final value. For underdamped second order systems, the 0% to 100%

rise time is normally used. For overdamped systems, the 10% to 90% rise time is commonly used.

- Peak time, t_p: The peak time is the time required for the response to reach the first peak of the overshoot.

- Maximum Overshoot, M_p: The maximum overshoot is the maximum peak value of the response curve measured from unity. If final steady-state value of the response differs from unity, then it is common to use the maximum percent overshoot. It is defined by,

Maximum percent overshoot
$$= \frac{c(t_p) - c(\infty)}{c(\infty)} \times 100\%$$

The amount of the maximum (percent) overshoot directly indicates the relative stability of the system.

- Settling time, t_s: The settling time is the time required for the response curve to reach and stay within a range about the final value of size specified by absolute percentage of the final value (usually 2% or 5%). The settling time is related to the largest time constant of the control system. Which percentage error criterion to use may be determined from the objectives of the system design in question?

The delay time, t_d of the above system which 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 1.75 sec.

The peak time, t_p is the time taken for the system to reach the first peak of overshoot is about 2.0 sec.
The maximum Overshoot, M_p of the system is approximately 60%.

Finally, the setting time, t_s, is about 10.2 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:-

```matlab
% 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)
```

The result is as follows,

z =

  -1.4387
  -1.4082

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,

$$G_c(s) = K_p(1 + \frac{1}{sT_i} + sT_d)$$

$$= 18(1 + \frac{1}{1.405s} + 0.35124s)$$

$$= \frac{6.3223(s + 1.4235)^2}{s}$$

The PID controller has a double zero of -1.4235. By trial and error, let us keep 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,

$$G_c(s) = 18(1 + \frac{1}{3.077s} + 0.7692s)$$

$$= 13.846 \frac{(s + 0.65)^2}{s}$$

$$= \frac{13.846s + 17.998s + 5.85}{s}$$

The PID transfer function and plant transfer function in series can be calculated by MATLAB and the result is as follows,

```
%calculation of series system response using matlab
num1=[0 13.846 17.998 5.85];
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 result to,

```
num/den =

   13.846 s^2 + 17.998 s + 5.85

  ---------------------------

      s^4 + 6 s^3 + 5 s^2
```

$$= \frac{13.846s^2 + 17.998s + 5.85}{s^4 + 6s^3 + 5s^2}$$

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

```
%calculation of feedback system response using matlab
num1=[0 0 13.846 17.998 5.85];
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,

```
num/den =

       13.846 s^2 + 17.998 s + 5.85

  -----------------------------------------

  s^4 + 6 s^3 + 18.846 s^2 + 17.998 s + 5.85
```

Therefore the overall close loop system response,

$$\frac{R(s)}{C(s)} = \frac{13.846s^2 + 17.998s + 5.65}{s^4 + 6s^3 + 18.846s^2 + 17.998s + 5.65}$$

The unit step response of this system can be obtained with MATLAB,

```
%MATLAB script of the Designed PID Controller System
num=[0 0 13.846 17.998 5.85];
den=[1 6 18.846 17.998 5.85];
step(num,den);
grid;
title('Unit Step Response of The Design System')
```
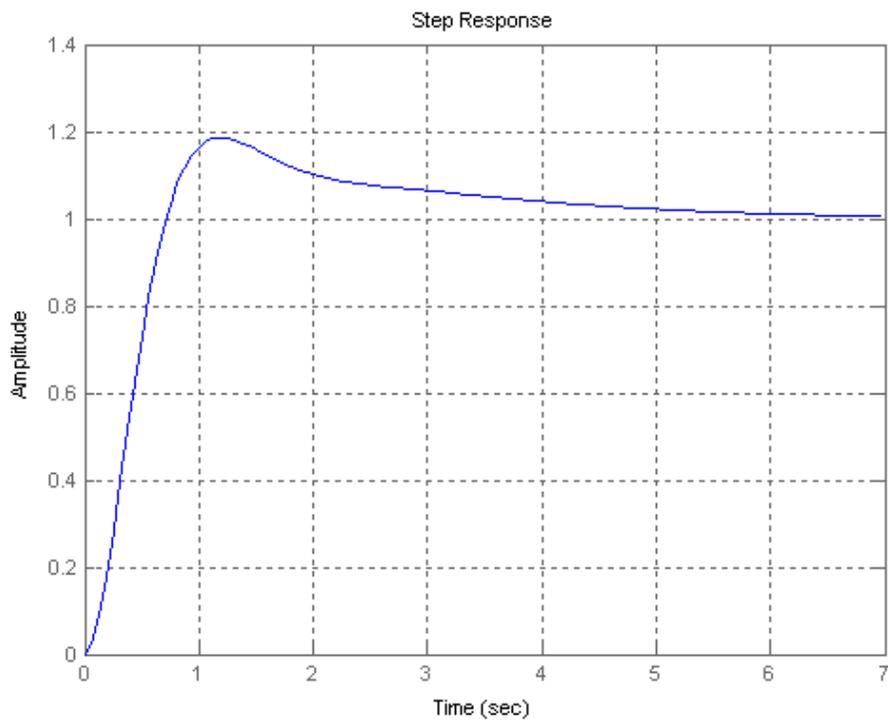

Figure 4.6. Improved System Response

The new system response has somehow improved. The Maximum Overshoot, M_P has 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

$K_p = 18$, $T_i = 3.077$ and $T_d = 0.7692$.

To improve the system further, let's 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,

$$G_c(s) = 39.42(1 + \frac{1}{3.077s} + 0.769s)$$

$$= 30.311\frac{(s+0.65)^2}{s}$$

$$= \frac{30.311s^2 + 39.42s + 12.81}{s}$$

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

```
%calculation of series system response using matlab
num1=[0 30.311 39.42 12.81];
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)
```

The result is,

num/den =

   30.311 s^2 + 39.42 s + 12.81

  ----

     s^4 + 6 s^3 + 5 s^2

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

```
%calculation of feedback system response using matlab
num1=[0 0 30.311 39.42 12.81];
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,

num/den =

      30.311 s^2 + 39.42 s + 12.81

  ----

  s^4 + 6 s^3 + 35.311 s^2 + 39.42 s + 12.81

$$G_c(s) = \frac{30.311s^2 + 39.42s + 12.81}{s^4 + 6s^3 + 35.311s^2 + 39.42s + 12.81}$$

The system response can be obtained with MATLAB,

```
%MATLAB script of the Designed PID Controller System
num=[0 0 30.311 39.42 12.81];
den=[1 6 35.311 39.42 12.81];
step(num,den);
grid;
title('Unit Step Response of The Design System');
```
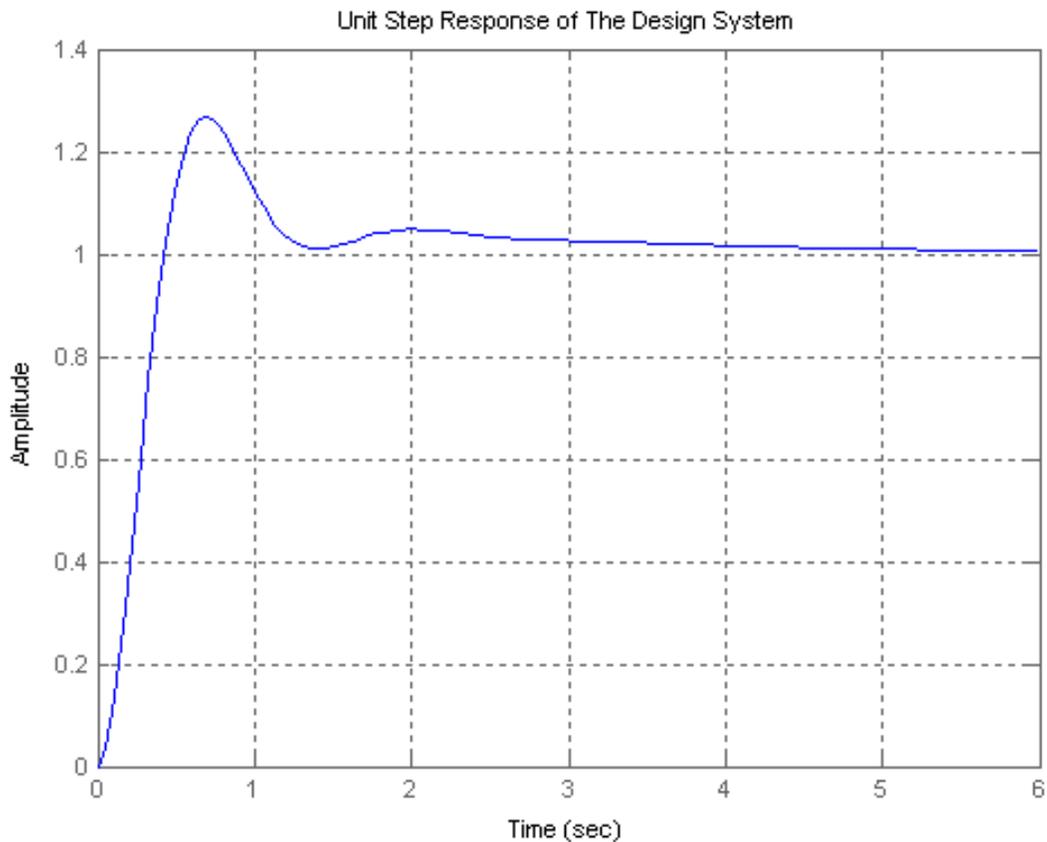

Figure 4.7. "Optimized" System Response.

The above response shows that the system has improved. The response is faster than the one shown in figure 8. The Maximum Overshoot, M_p has increased to about 22%. This is still acceptable since the Maximum Overshoot allowable is less than 25%.The Settling Time, t_s remains the same i.e. 6 sec. The Peak Time, t_p and Delay Time, td has improved. The new PID parameters 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 then 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 25% Maximum Overshoot allowable. The settling time, t_s of the second and the third responses 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 Z-N tuning rule. Hence we can conclude that Z-N tuning rule 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 form -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. The system described above can be further improved or optimized.

Chapter-5

Designing of PID

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 inputs 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=80;


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)
```

Figure 5.1. Initialising the GA.

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 of the GA. Generally the bigger the population size the better is the final approximation.

VariableBounds - Since this project is using genetic algorithms to optimize the gains of a P1D 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 processes. 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. Figure 19 illustrates a population of eighty members being initialized with values randomly selected between - 100 and 100.

EvalFN - The evaluation function is the matlab function used to declare the objective function. it will fetch the file name of the objective function and execute the codes and return the values back to the main codes.

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 and from binary.

2. Binary strings will lose its precision during the conversion process. 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 Figure 19, where the 'le-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 is from the GAOT toolbox. It will combine all the previously described terms and creates an initial population of 80 real valued members between 100 and 100 with 6 decimal place precision.

5.3 Setting the Genetic Algorithm 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-61 0];
termFN='maxGenTerm';
termOps=100;
selectFN='normGeomSelect';
selectOps=0.08;
xOverFNs='arithXover';
xOverOps=4;
mutFNs='unifMutation';
mutOps=8;
```

Figure 5.2. Parameters Setting Of GA.

Bounds - The variable hound 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, 'star-Pop', is defined as the population described in the previous section i.e. 'initPop', see Figure 5.1.

opts - The options for the Genetic Algorithm consist of the precision of the string values i.e. le-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 criterion 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 2.4.

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.

X0verFN - 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.

rnutFNs - 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 in figure.

The function '**ga.m**' will evaluate and iterate the genetic algorithm until it fulfils the criteria described by its termination function.

```
%Performing the genetic algorithm
[x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,opts,..

termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFNs,mutOps);
```

Figure 5.3. Performing the GA.

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.

The best population may be plotted to give an insight into how the Genetic Algorithm converged to its final values as illustrated in figure 5.4.
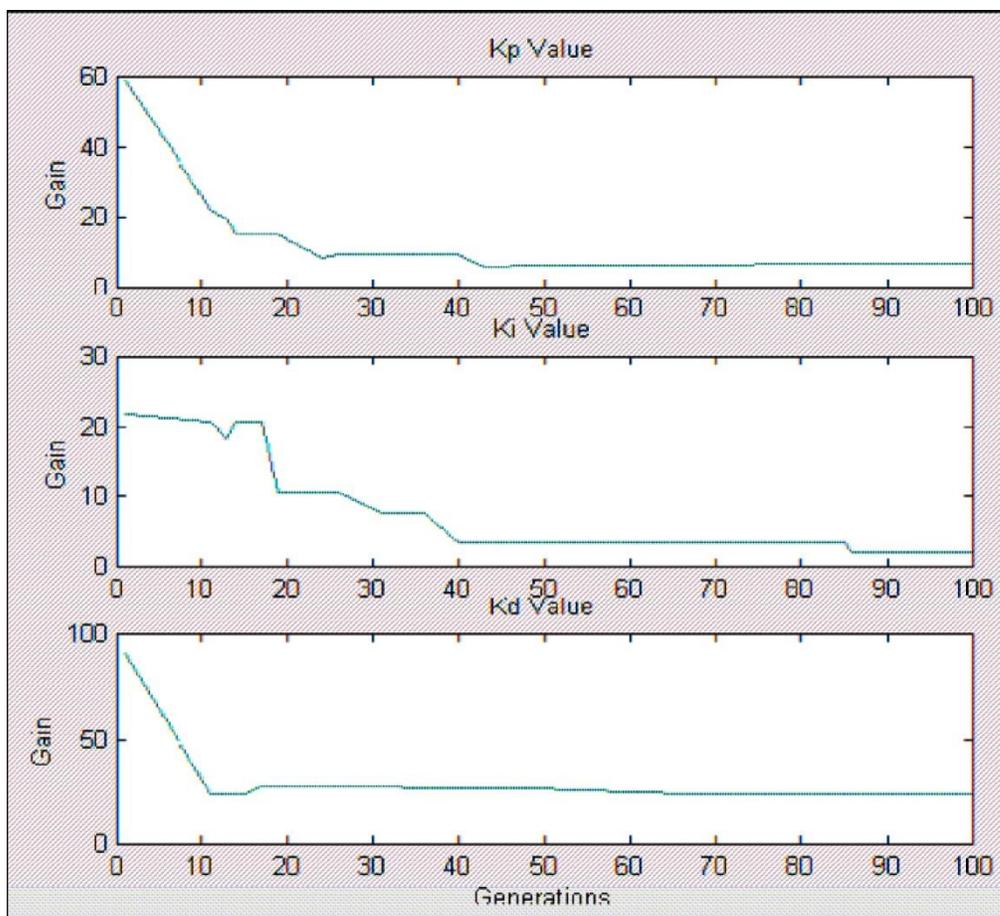

Figure 5.4. Illustration of Genetic Algorithm Converging Through Generations.

5.5 The Objective Function of the Genetic Algorithm

This is the most challenging part of creating a genetic algorithm is writing the objective functions. In this project, the objective function is required to evaluate the best PlD 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 chromosones 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
```

Figure 5.5. Objective Function

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. The P, I and D gains are used to create a PID controller according to the equation below:

$$C_{pid} = \frac{K_D s^2 + K_P s + K_I}{s}$$

The newly formed ND controller is placed in a unity feedback loop with the System's transfer function. This will result in a reduction 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. The chromosome is assigned an overall fitness value according to the magnitude of the error, the smaller the error the lamer the fitness value. Below is the codes used to implement the MSE performance criteria.

```
%Calculating the error for i=1:301

error(i) =1-y(i);

end

%Calculating the MSE error_sq =
error*error';

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

Figure 5.6. Calculating the Error of the System Using MSE Criteria.

Additional code was added to ensure that the genetic algorithm converges to a controller that produces a stable system. The code, shown in figure 5.7, 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=100e300;

elseif poles(2)>0
MSE=100e300;
elseif poles(3)>0
MSE=100e300;
elseif poles(4)>0
MSE=100e300;
elseif poles(5)>0
MSE=100e300; end
fx_val=1/MSE;
```

Figure 5.7. Stability of the Controlled System.

5.6 Results of the Implemented Genetic Algorithm PID Controller

In the following section, the results of the implemented Genetic Algorithm PID Controller will be analyzed. The GA designed PID controller is initially initialized with population size of 20 and the response analyzed. It was then initialized with population size of 40 and 80. The response of the GA designed PID will then be analyzed for the smallest overshoot, fastest rise time and the fastest settling time. The best response will then be selected.

The following is the plot of the GA designed PID with the population size 20. From the figure below,
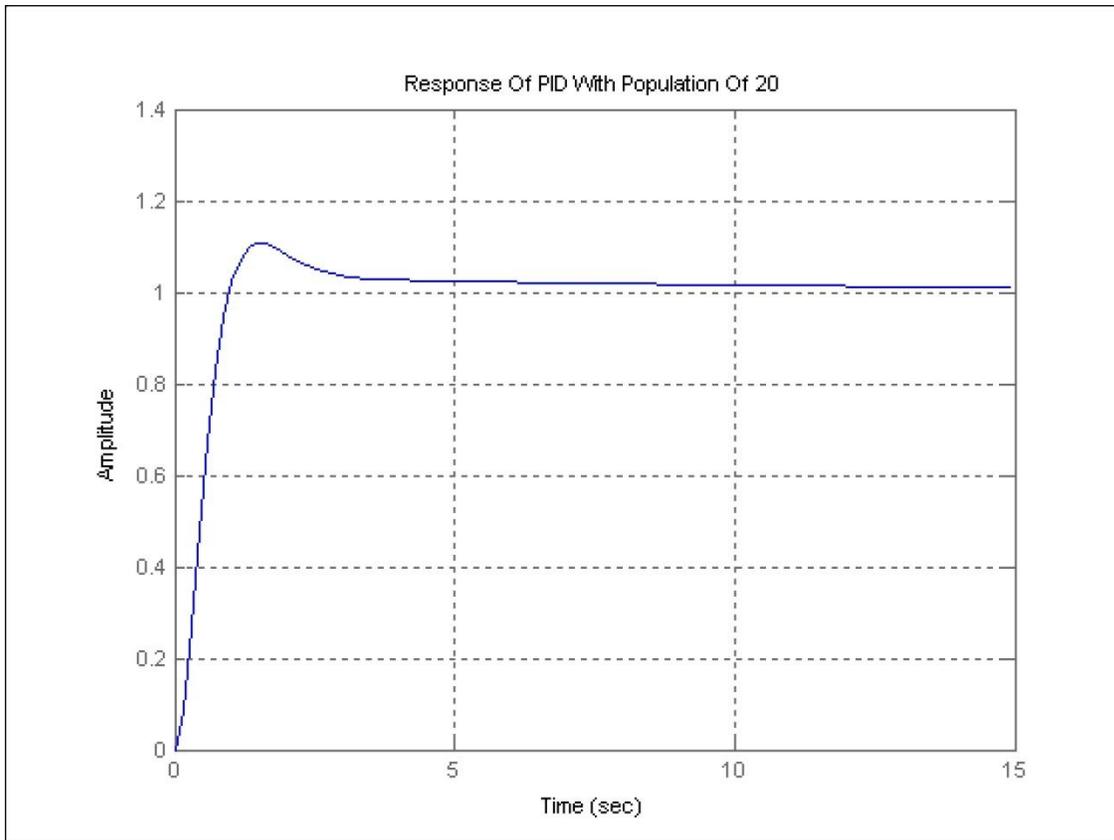
the response of the GA PID will be analyzed.

Figure 5.8. PID Response with Population Size of 20.

From the figure 5.8, the response of system looks reasonable stable. However it can be seen in the above plot that there is an offset in the response. Let observe if the offset can be removed with a bigger population size. These can be observed in the future plots.

From the above figure, the details of the system response will be analyzed. The peak amplitude of the response is 1.11. The overshoot of the response is 10.6%.The settling time of the response is 6.97 seconds and finally the response of the rise time is 0.666 seconds.

From one look, the above response is definitely much better than the classical PID tuning method as shown in the chapter 4.

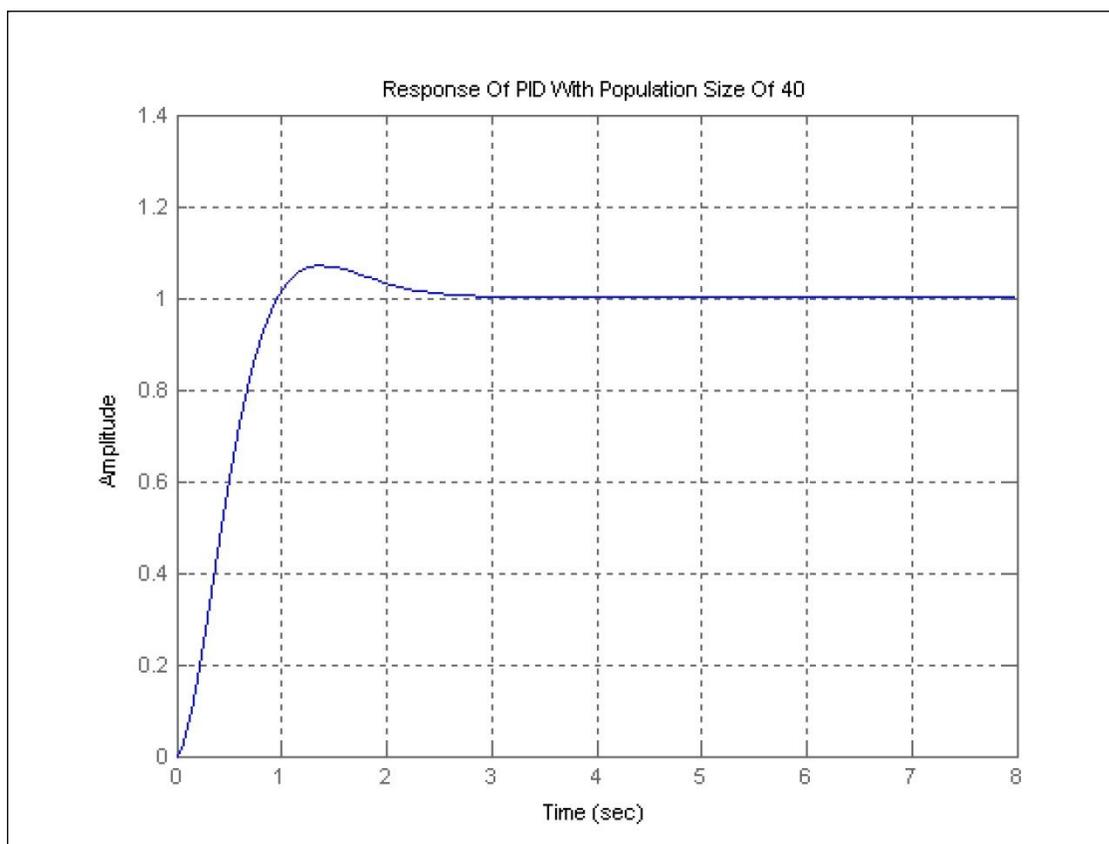The following figure depicts the response of GA designed PID with the population size of 40.

Figure 5.9. PID Response with Population Size of 40.

From the following figure 5.9 above, the system response is much better than the one simulated with the population size of 20. It can be observed that the system offset has been removed.

From the above figure, the details of the response will be analyzed. The peak amplitude of the response is 1.07. The overshoot of the response is 6.98%. The settling time of the response is 2.2 seconds and

the rise time of 0.64 seconds. From the following results, it is obvious that the population of size 40 has returned a better result than the one with the population size of 20.

In this response, the overshoot value has improved. The settling time has reduced from 6.97 seconds to 2.2 seconds. The rise time has improved slightly that is 0.64seconds as compared to 0.666 seconds. The overall response is that it has improved as compared to the one in figure 5.8.
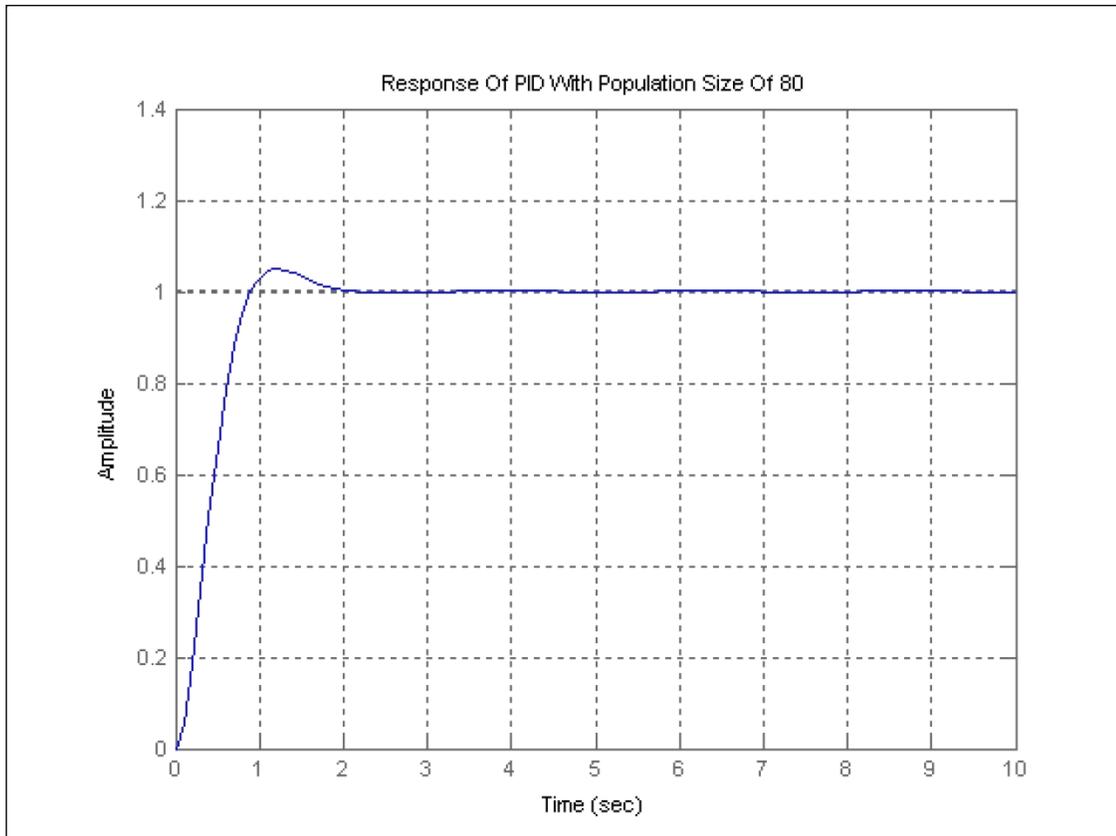

Figure 5.10. PID Response with Population Size of 80.

Finally let's look at the system response of the population size of 80. From observation, the system returned a much better response. The GA designed PID with population size of 80 has the following response factors, peak amplitude of 1.05, overshoot of 4.86%, rise time of 0.818 seconds and settling time of 1.545 seconds.

The population size of 90 and above were tried and the program has not shown any sign of improvement in the optimization. Hence a decision was made to stick to the population size of 80 and analyze it against the Conventional PID Controller. Proceeding with the higher population size will take up a lot of computer memory space. Since the Genetic Algorithm designed PID with population size of 80 seems to have the best response as compared to the others responses.

The above analysis is summarized in the following table.

Measuring Factors	Conventional PID Controller	GA Controller
Rise Time, t_r (sec.)	0.445	0.818
Peak Time, t_p (sec.)	0.695	1.182
Maximum Overshoot, M_p (%)	1.28	1.05
Settling Time, t_s (sec.)	1.667	1.545

Table 3. Results of Conventional PID Controller and GA Designed Controller.

Chapter - 6

Further Works and Conclusions

6.1 Further Works

It is hoped that this project can be improved to include the implementation of tuning the MD 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.

6.2 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 approach 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.

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 other methods available.

Finally this project has made me more appreciative of the Control Engineering and its contribution to the improvement of the industry 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 bio-medical 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.

References

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

[2]   Goldberg, David E. *" Genetic Algorithm in Search, Optimization and Machine Learning"* Addison –Wesley Pub. Co.1989.

[3]   Chris Houck, Jeff Joines, and Mike Kay, *"A Genetic Algorithm for Function Optimization: A Matlab Implementation"* NCSU-IE TR 95-09, 1995. http://www.ie.ncsu.edu/mirage/GAToolBox/gaot/

[4]   G.J. Battaglia and J.M. Maynard, *"Mean square Error: A Useful Tool for Statistical Process Management,"* AMP J. Technol. 2, 47-55, 1992.

[5]   Q.Wang, P. Spronck & R. Tracht, *" An Overview of Genetic Algorithms Applied To Control Engineering Problems,"* Proceedings of the Second Conference on Machine Learning and Cybernetics, 2003.

[6]   Luke,S., Balan, G.C. and Panait, L*, "Population Implosion In Genetic Programming",* Department Of Computer Science, George Mason University. http://www.cs.gmu.edu/~eclab

[7]   Gotshall,S. and Rylander,B., *"Optimal Population Size And The Genetic Algorithm."* , Proc On Genetic And Evolutionary Computation Conference, 2000.

[8]   Naeem, W., Sutton,R. Chudley. J,Dalgleish, 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.

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

[10] Herrero, J.M., Blasco, X, Martinez, M Salcedo, J.V., *"Optimal PID Tuning With Genetic Algorithm For Non Linear Process Models.",* 15th Terminal World Congress, 2002.

[11] O'Dwyer, A. *" PI And PID Controller Tuning Rules For Time Delay Process: A Summary. Part I: PI Controller Tuning Rules."* Proceedings Of Irish Signals And Systems Conference, June 1992.

[12] K. Krishnakumar and D. E. Goldberg, *"Control System Optimization Using Genetic Algorithms",* Journal of Guidance, Control and Dynamics, Vol. 15, No. 3' pp.735-740, 1992.

[13] T O'Mahony, C J Downing and K Fatla, *"Genetic Algorithms for PID Parameter Optimization; Minimizing Error Criteria"*, Process Control and Instrumentation 2000 26-28 July 2000, University of Stracthclyde, pg 148-153, July 2000.

[14] A. Varsek, T. Urbacic and B. Filipic, *"Genetic Algorithms in Control Design and Tuning",* IEEE Trans. Sys. Man and Cyber, Vol.23, No. 5, pp 1330-1339, 1993.

[15] Chipperfield, A. J., Flemming, P.J., Pohlheim, H. and Fonseca, C.M., *" A Genetic AlgorithmToolbox for MATLAB"*, Proc International Conference on Systems Engineering, Coventry, UK, 6-8 September, 1994.

[16] Aigner, P., Pythian, M., Wen, P. and Black, J, *"Computer Controlled System"*, Distance Education Centre, USQ, 2003.