DC MOTOR CONTROL USING GENETIC ALGORITHM BASED PID

By

NURUL ASHIKIN BINTI SHAHRONI

FINAL DISSERTATION

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

DC MOTOR CONTROL USING GENETIC ALGORITHM BASED PID

by

Nurul Ashikin binti Shahroni

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

[illegible]

____

Assoc. Prof. Dr. Irraivan Elamvazuthi
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

September 2011

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

Nurul Ashikin binti Shahroni

ACKNOWLEDGEMENT

First and foremost, I would like to express my heartily gratitude to my supervisor, Assoc. Prof. Dr. Irraivan Elamvazuthi for the guidance, motivation, support, inspiration, encouragement and enthusiasm given throughout the progress of this project. With the guidance and motivation, I manage to overcome hardships in completing the project and also keep motivated throughout these two semesters.

My appreciation also goes to my beloved parents and family who have always been there for me, supporting me all these years. Thanks for their encouragement, love and emotional support that they had given to me. Without them, my thesis and project would not have finished on time.

Nevertheless, my honest appreciation is also dedicated towards all my friends and those who have helped me in the completion of the project, either directly or indirectly. Thank you.

ABSTRACT

This dissertation presents the research that had been done on the chosen topic, works done and results acquired throughout the Final Year Project for two semesters, about the **DC Motor Control using Genetic Algorithm based PID.** The objectives of this project are to optimize speed control of the DC motor by using Genetic Algorithm based PID, to improve the performances of the DC motor controller in term of rise time, settling time, maximum overshoot and Integral of Time Absolute Error (ITAE) and to decide the best parameters to be used for Genetic Algorithm that can optimize the performance of a DC Motor (eg: population size, mutation rate and crossover value). First, the report discusses the types of DC motor available in industry nowadays and the origination of Genetic Algorithm itself. Next, the paper sees the implementation of DC motor control and the tuning available that has been researched before. The usage of Genetic Algorithm is briefly explained which comprises of three main phases; reproduction, crossover and mutation. For this study, the software used is MATLAB/Simulink, where the implementation of the chosen DC motor model is represented and Genetic Algorithm is incorporated into the PID controller of the motor, to observe the performance of chosen parameters available. Next, the paper shows the results of PID controller tuning and also the results for the implementation of GA based PID controller. Comparison then is made and discussed to see whether the results are as expected. Lastly, recommendation and conclusion pertaining to the completion of this project are presented.

TABLE OF CONTENTS

LIST OF ABBREVATIONS

DC	Direct Current
GA	Genetic Algorithm
GA-PID	Genetic Algorithm based PID Control
P	Proportional
PI	Proportional-Integral
PD	Proportional-Derivative
PID	Proportional-Integral-Derivative
IAE	Integral of Absolute Error
ISE	Integral of Squared Error
ITAE	Integral of Time Absolute Error
SA	Simulated Annealing
BLDC	Brushless Direct Current
EA	Evolutionary Algorithm

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background of Study

It is known that DC motor is employed in almost all industrial automation due to its excellent speed control characteristics. The application of DC motor varies from small industry to high technology industry.Therefore, the optimization in the application of DC motor controller has been widely discussed and researched for a better performance of the motor. Previously, industry usually used classical method in tuning the controller of a DC motor such as Proportional, Integral and Derivative method (PID). Due to some constraints in the output of the PID tuning, people have come out with ideas and researches on how to optimize the controller. In this paper, the DC motor model is designed and tuned by using Genetic Algorithm (GA) based PID, where GA is a stochastic global search method that emulated the process of natural evolution.

GA has proven to be one of the most preferable methods in an optimization of a plant or project, to obtain the best results. Research had been done on this technique from the year of 1989, to see the implication of using GA in a system, and how does GA actually effect the performance of a plant. GA has been shown to be capable of locating high performance areas in complex domains without experiencing the difficulties associated with high dimensionality or false optima. For this study, the optimization is done to achieve best result in tuning the speed control of a DC motor.

The MATLAB software is used for simulation and coding of the project. At the end of this study, the system responses are analyzed to see whether the usage of GA based PID bring any significance in improving the performances of the motor's controller. This is done to show the relevancy of the project and the effect of using GA into DC motor for optimization, which can be seen by improving the step response characteristics such as, reducing the steady-states error; rise time, settling time and maximum overshoot in speed control of a DC motor.

1.2 Problem Statements

In industry, the usage of classical method such as Ziegler-Nichols Method for tuning on a conventional Proportional-Integral (PI) controller, Proportional-Derivative (PD) controller or Proportional-Integral-Derivative (PID) controller have been widely used, especially for speed and position control of a DC motor. However, often in practice, tuning is carried out by an experienced operator using a 'trial and error' procedure and some practical rules, which is often a time consuming and portrait as a difficult activity to be carried out [7]. Due to this constraint, studies have been done in order to optimize the performance of a DC motor widely.

In this project, four main aspects in term of the performances of a DC motor are being investigated:

(a) Rise Time

The rise time of a controller needed improvement (by reaching zero ideal values) to obtain a good performance of motor control.

(b) Settling Time

A faster settling time of a system shows that the control system is better.

(c) Maximum Overshoot

Trying to obtain an ideal maximum overshoot of a system to improve its performances (approaching to zero values).

(d) Integral of Time Absolute Error (ITAE)

Trying to decrease the error to gain the best performance for the controller (lesser error means better motor control system is portraited).

1.3 Objective and Scope of Study

The purpose of this study is to achieve three different objectives,which are:

- To optimize speed control of the DC motor by using Genetic Algorithm based PID.
- To improve the performances of the DC motor controller in term of rise time, settling time, maximum overshoot and Integral of Time Absolute Error (ITAE).
- To decide the best parameters to be used for Genetic Algorithm that can optimize the performance of a DC Motor (eg: population size, mutation rate and crossover value)

Genetic Algorithm (GA) is basically a stochastic algorithm based on principal of natural selection and genetics. Using GA will result in the optimum controller being evaluated for the system every time. To prove that desired optimization is achieved, the results of GA based PID method will be compared with PID only technique later.

1.4 Scope of the Project

Base on the objectives of the project, the scope of study is divided into two main categories;

1.4.1 DC Motor

DC motor has been widely used in industry where variable speed and strong torque are required specifically [2]. There are some basic functions that motor control systems perform, such as speed and position control, motor and circuit protection, starting and stopping and also surge protection [2]. In this project, we are going to do ground research on the speed controller of a DC Motor. Some control systems in industry require variable speed, which for the case of DC motor; this can be achieved by controlling the voltage to the armature and the fields of the motor.

When full voltage is applied to both the armature and the field, the motor operates at its base or normal speed. When full voltage is applied to the field and reduced voltage is applied to the armature, the motor operates below normal speed. However, when full voltage is applied to the armature and reduced voltage is applied to the field, the motor operates above normal speed. This is the basic applications of a speed control of a DC motor.

Due to this excellent speed control characteristics, speed and position control of DC motor have attracted considerable research and several methods have evolved, even though the maintenance costs of DC motor are higher than the induction motor. One of the most famous researched methods is Genetic Algorithm.

1.4.2 Genetic Algorithm (GA)

Genetic Algorithm (GA) is a search heuristic method that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. It has 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. Therefore, in this project, this optimization technique, GA, will be utilized to tune the PID controller of a specific DC motor to find the optimum controller for the motor accordingly.

CHAPTER 2
LITERATURE REVIEW

In the process of undertaking the project, to understand and to achieve the desired objectives of the project, thorough reading regarding DC Motor Control, PID Controller and Genetic Algorithm is needed. Hence comprehensive and depth research for resources are needed that somehow have contributed to the required accomplishment of the project. To understand the project better, the literature review is done based on different parts essential for the projects.

2.1 DC MOTOR CONTROL

In spite of the development of power electronics resources, the direct current (DC) machine became more and more useful. Nowadays their uses is not limited in the car applications (electric vehicle), in applications of weak power using battery system (motor or toy) but also in the electric traction in the multi-machine too [5]. In the application of DC motor, the three most common speed control methods are resistance control, armature voltage control, and armature resistance control [4]. According to Ayasun and Karbayez, feedback control system is also discussed for implementation of speed control system for DC motor drives [4].

In the field resistance control method, a series resistance is inserted in the shunt-field circuit of the motor in order to change the flux by controlling the field current [4]. However, in the armature voltage control method, the voltage applied to the armature circuit is varied without changing the voltage applied to the field circuit of the motor [4]. For the armature resistance control, an increase in the armature resistance results in a significant increase in the slope of the torque-speed characteristics of the motor while the no-load speed remains constant [4]. These cases are true for a separately-excited DC motor system.

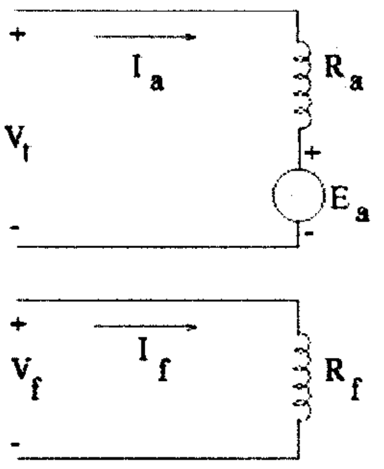


Figure 2.1: Model for Separately Excited DC Motor

For the case of position control of a DC shunt motors, Neenu Thomas and Dr. P. Poongodi had found out that, the designed PID with GA has much faster response than the response of a classical method [3] which is by using Ziegler-Nichols method. With GA, the response is better in term of rise time, settling time and also the error associated with the methods proven to be lesser than conventional method [3].

In other research [9], where the motor discussed is a Brushless DC Motor (BLDC), it is observed that GA performed better than Simulated Annealing (SA). (From the simulation results, it is observed that GA performed better than SA for BLDC motor design). Both GA and SA optimization techniques are proven to be efficient powerful tools for obtaining global optimal solutions of the BLDC motor compared to traditional design procedures. Optimal solutions show that GA is more efficient than SA and it is proven that the accuracy of the optimal parameters is similar in both the methods.

From all the readings, it has been found out that the brush DC motor model portrait good electrical and mechanical performances more than other DC motor models [5]. Therefore, for this study, the brush DC motor model is chosen for research.

2.2 SPEED CONTROL OF DC MOTOR AND ITS TUNING

Approaching the control of speed in DC motor in different perception, we can also discuss in term of the controllers of the speed itself. The controller types comprise of several conventional and numeric controller, which can be: PI, PD or PID Controller, Fuzzy Logic Controller; or the combination between them Fuzzy-Genetic Algorithm, Fuzzy-Neural Networks, Fuzzy-Ants Colony or even Fuzzy-Swarm [5]. In other research, the controllers of a DC motor can also be in other types, which is the combination of GA and PID as in [6].

However, some of the methods show some major drawbacks in term of their implementation. For instance the conventional PID controller is the most widely used in industry, because of its remarkable effectiveness, simplicity of implementation and broad applicability [7]. Nevertheless, often in practice, tuning is carried out by an experienced operator using a 'trial and error' procedure and some practical rules, which is often a time consuming and portrait as a difficult activity to be carried out [7].

On the other hand, the application of other method which is fuzzy controller also has major drawback, which is the insufficient analytical design technique (choice of the rules, the membership functions and the scaling factors) [5]. Therefore, the fuzzy controller is often associated with the genetic algorithms approach to improve the performance of the controller as in [5], using Fuzzy Logic-Genetic Algorithms optimization.

2.3 OPTIMIZATION AND GENETIC ALGORITHM

From readings, optimization of a system is proven to be very important and has become a major research topic. The goal of global optimization is to find the global optima, that is, global maxima or minima of the objective function. Optimization problems are used to find good parameters or designs for components or plants to be put into action by the human beings or machines [8].

Observing the trend of optimization technique that is chosen to improve a performance of a controller, Genetic Algorithm is one of the most popular optimization technique used. GA is often used in collaboration with other tuning method available. It has been proven in many studies that Genetic Algorithm gives a very good result. According to [6], GA is however 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' method. Based on these criteria and a lot of readings, Genetic Algorithm is chosen as an interesting topic of study for optimization technique in control engineering.

Genetic Algorithms (GAs) are actually subclass of evolutionary algorithms (EAs). What is Evolutionary Algorithms? They are population-based meta heuristic optimization algorithms that use biology-inspired mechanisms and survival of the fittest theory in order to refine a set of solution iteratively. Therefore, GA is its subclass where the elements of the search space are binary strings or arrays of other elementary types [8].GA is computer based search techniques patterned after the genetic mechanisms of biological organisms that have adapted and flourished in changing highly competitive environment [8].

Genetic Algorithm is a search heuristic that mimics the process of evoluation [8]. From all the readings above, it is stated that GAs are powerful and broadly applicable stochastic search and optimization techniques that really work for many problems that are very difficult to solve by conventional techniques. In general, GA has five basic components, which are;

1. A genetic representation of solutions to the problem
2. A way to create an initial population of solutions
3. An evaluation function rating solutions in terms of their fitness
4. Genetic operators that alter the genetic composition of children during reproduction
5. Values for the parameters of genetic algorithms [1].

Basically, GA consists of three main stages: reproduction, crossover and mutation. The application of these three basic operations allows the creation of new individuals which may be better than their parents [2].

This algorithm is repeated for many generations and finally stops when reaching individuals that represents the optimum solution to the problem [2]. A GA is typically initialized with a random population consisting of between 20-100 individuals. Determining the number of population is the one of the important step in GA [6]. The decision is still based on trial and error.

2.3.1 Introduction

The usage of GA is proven to be more efficient than the traditional methods due to some reasons. This Genetic Algorithm is substantially different to traditional search and optimization techniques because of five main differences. They are:

1. GA searches a population of possible solutions in parallel, not in term of single point [6].

2. To run a Genetic Algorithm optimization, one does not need to have derivative information or other auxiliary knowledge. One only needs to define the objective function for the optimization problem and the corresponding fitness levels that will influence the directions of the search for best population [6].

3. Genetic Algorithm uses probabilistic transition rules, not deterministic rules [6].

4. Genetic algorithm may provide a number of potential solutions (optimum solutions) and the final choice is up to the user [6].

2.3.2 Reproduction

The principle behind genetic algorithms is essentially Darwinian natural selection [1]. Selection provides the driving force in GA. Typically; a lower selection pressure is indicated at the start of a genetic search in favor of a wide exploration of the search space, while a higher selection pressure is recommended at the end to narrow the search space [1]. Common types for reproduction phase are as follows:

- Roulette wheel selection
- $(\mu + \lambda)$-selection
- Tournament selection
- Steady-state reproduction
- Ranking and scaling
- Sharing [1]

For this study, we will concentrate of Roulette wheel selection as it is a common selection technique and the best known selection type still until today.

2.3.2.1 Roulette Wheel Selection

The basic idea behind this selection is to determine selection probability or survival probability for each chromosome proportional to the fitness value. Then a model roulette wheel can be made displaying these probabilities. The selection process is based on spinning the wheel the number of times equal to population size, each time selecting a single chromosome for the new population [1]. An example of a roulette wheel selection is as below;

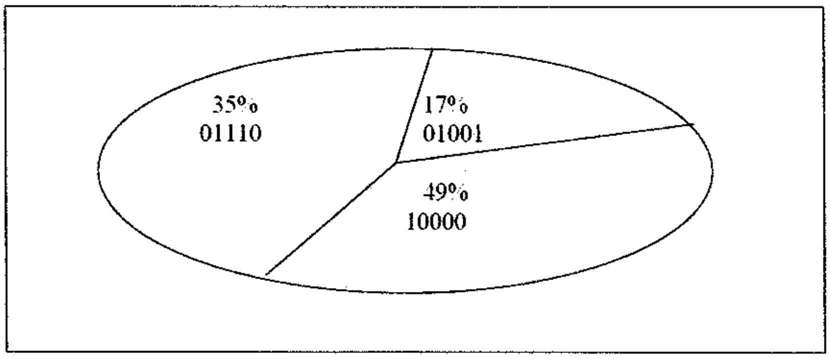


Figure 2.2: Depiction of roulette wheel selection

2.3.3 Crossover

Once the selection process is completed, the crossover algorithm is initiated. The crossover operations swap certain parts of the two selected strings in a bid to capture the good parts of old chromosomes and create better new ones [6]. Genetic operators manipulate the characters of a chromosome directly, using the assumption that certain individual 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.

There are Single-Point Crossover, Multi-Point Crossover and Uniform Crossover Algorithms available for the chosen technique. For this study, Multi-point Crossover is chosen because it is the extension of Single-point Crossover and operates on the principle that the parts of a chromosome that contribute most to its fitness might not be adjacent [6]. Besides, 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.

There are three main stages involved in a Multi-Point Crossover:

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

Example: If the string *11111* and *00000* were selected for crossoverand the multipoint crossover positions were selected to be 2 and 4 then the newlycreated strings will be *11001* and *00110* as shown in Figure 3.

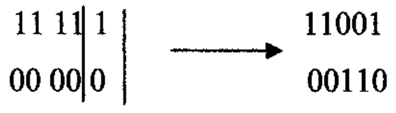


Figure 2.3: Illustration of Multi-Point Crossover.

2.3.4 Mutation

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

- 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.
- 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 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 [6].

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 4.

$$10000 \longrightarrow 10010$$

Figure 2.4: Illustration of Mutation Operation

After all these three steps of Genetic Algorithm are done, we hope to find the optimum solution to the speed controller of the chosen separately excited DC motor in the study.

CHAPTER 3
PROJECT METHODOLOGY

3.1 Genetic Algorithm

Figure 3.1 shows the flowchart of the Genetic Algorithm.

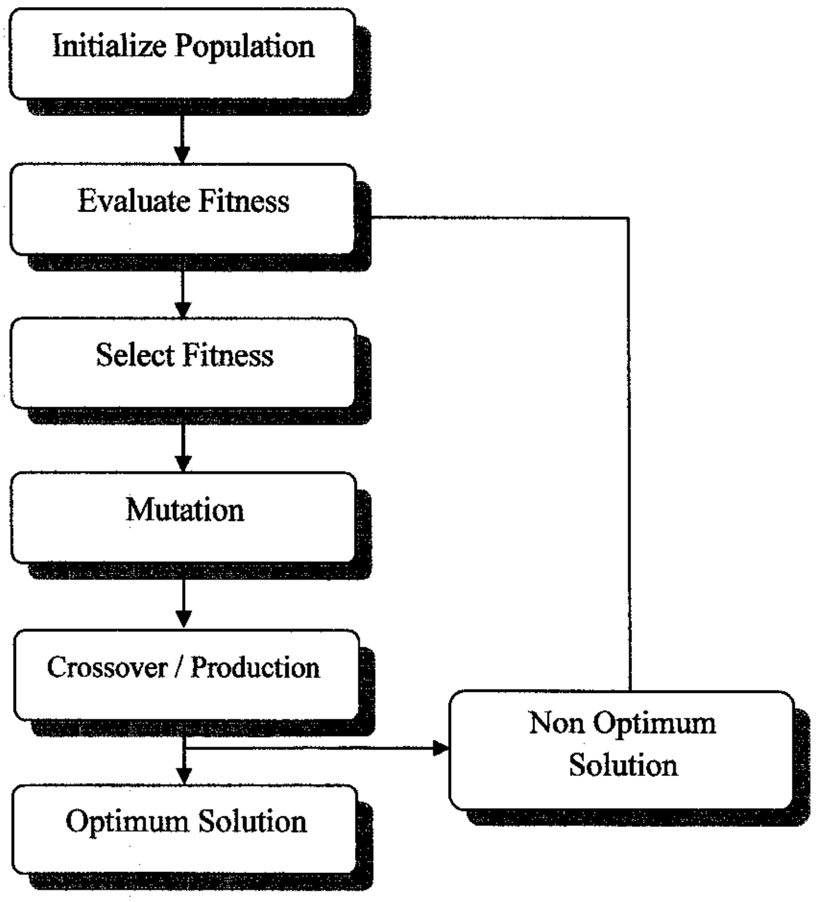


Figure 3.1: Flow for Genetic Algorithm

The detailed explanation of this technique is provided in Chapter 2. The flow of the entire project and the Gantt charts for the entire project starting from Final Year Project I up until Final Year Project II are shown in Appendix C, Appendix D and Appendix E accordingly.

3.2 DC Motor Application (FAULHABER)

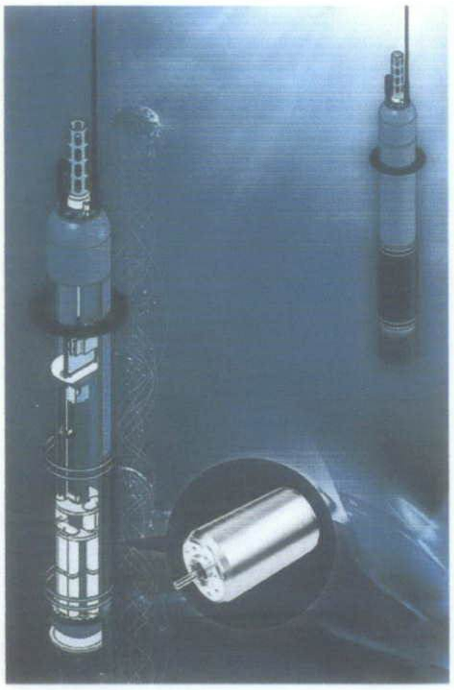


Figure 3.2: Robust DC Motor Drives Deep-Sea Sensors
by FAULHABER

For this project, the chosen DC motor to be applied with the optimization technique for the controller using Genetic Algorithm is a motor by FAULHABER which is Robust DC Motor Drives Deep-Sea Sensors. This motor is used to cope with extreme conditions to retrieve the right data especially for marine research. This kind of underwater sensor plumbs the depths of the world's ocean trying to withstand not only temperature differences but also extreme pressures, yet still having to deliver reliable readings. The objectives are to achieve reliability and absolutely maintenance-free depth control and also to come up with a robust, compact system for precise depth control of the underwater sensors.

For these purposes, according to Timo Witte, Project Manager for developing the NEMO floats, the properties of the DC brush motor are ideal. Therefore, FAULHABER have come out with the best solution to satisfy all the requirements; where the motor need to start up at minimal voltage, simple to install and reliable with precision control, with the usage of its DC Micromotor product.

To achieve this purpose, a biological principal is put to technical use by the DC Micromotor. Since the beginning of time, many species of fish have used a gas bladder to control their buoyancy in the water. This allows them to float in the water without expanding any extra energy, by a simple process of regulating the amount of gas in the bladder. This is also precisely what is required of the monitoring floats if they are to record data for the longest possible periods of time.

A hydraulic piston filled with oil is the main floatation device. In order to be able to steer the underwater sensor to the desired depth, the amount of oil in the swim bubble is varied via the piston. Depending on how well the bladder is filled, the overall density of the sensor changes and it sinks, floats and rises to the surface. A control piston serves as the drive element, by varying the gas pressure. To add the necessary muscles to these sensors, a DC micromotor performing at around 26W drives the piston. This is where our optimization technique is put in use; to better improve the system for these sensors.

3.3 MATLAB/Simulink Software

For this project, we are doing the simulation plus coding work to achieve our objectives. MATLAB with its tool boxes such as Simulink and SimPowerSystems is one of the most popular software packages used to run the simulation or testing of a plant or process system before the real implementationonto the equipments.Therefore, the MATLAB/Simulink software is fully utilized by embedding it into the DC motor model system for the optimization. The block diagram that is incorporated into the system is as shown below:

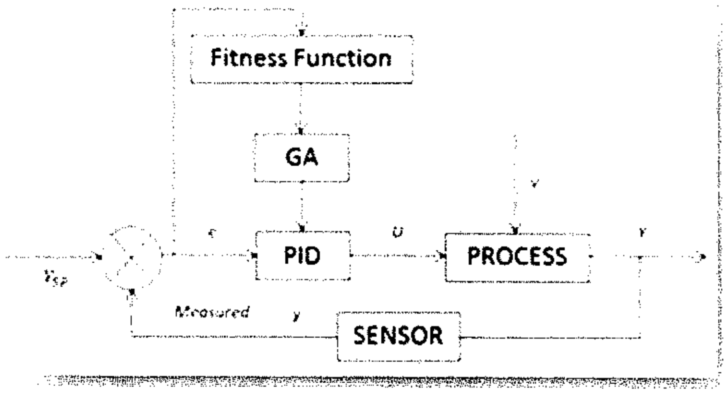


Figure 3.3: Block diagram of the overall system [14]

To achieve the complete Genetic Algorithm optimization, the usage of M-file with programming is needed. Results of the system will then be observed and analysed to find the best conclusion to the study of the system.

CHAPTER 4
RESULT AND DISCUSSION

4.1 DC Motor Transfer Function

For this project, a DC Micromotor with Graphite Commutation from FAULHABER (Series 3557...CS) is chosen based on the criteria given; compact dimensions of 35mm in diameter and 57mm in length teamed with high-performance output and motor performing at around 26W to drive the piston. Using the standard equation for the DC motor and the reference from FAULHABER datasheet (see Appendix A), the transfer function obtain is as shown below.

$$v = Ri + L\frac{di}{dt} + e_b \qquad (1)$$

$$T_e = \frac{L}{R} \qquad (2)$$

$$T_m = K_T i_a(t) \qquad (3)$$

$$T_m = J\frac{d^2\theta(t)}{dt^2} + B\frac{d\theta(t)}{dt} \qquad (4)$$

$$e_b = e_b(t) = K_b\frac{d\theta(t)}{dt} \qquad (5)$$

$$G(s) = \frac{\frac{1}{K_b}}{(T_m s+1)(T_e s+1)} \qquad (6)$$

Equation 1 up until equation 6 are the standard equation that are used to find the transfer function of the motor. With reference to the datasheet that we obtained for the micromotor (as in Appendix A), the values that are used into the calculation of the motor are;

$$K_T = 20.9\, mV/rpm$$

$$K_B = 2.19\, mV/rpm$$

$$J = 52 \times 10^{-7}\, kgm^2$$

$$L = 220\mu H$$

$$R = 1.34\, ohm$$

$$T_m = 16ms$$

Transfer function that is obtained for the motor is;

$$G(s) = \frac{0.46}{2.63s^2 + 16.16s + 1}$$

4.2 Simulation Work

To do all the PID simulation, the project is based on this block diagram;

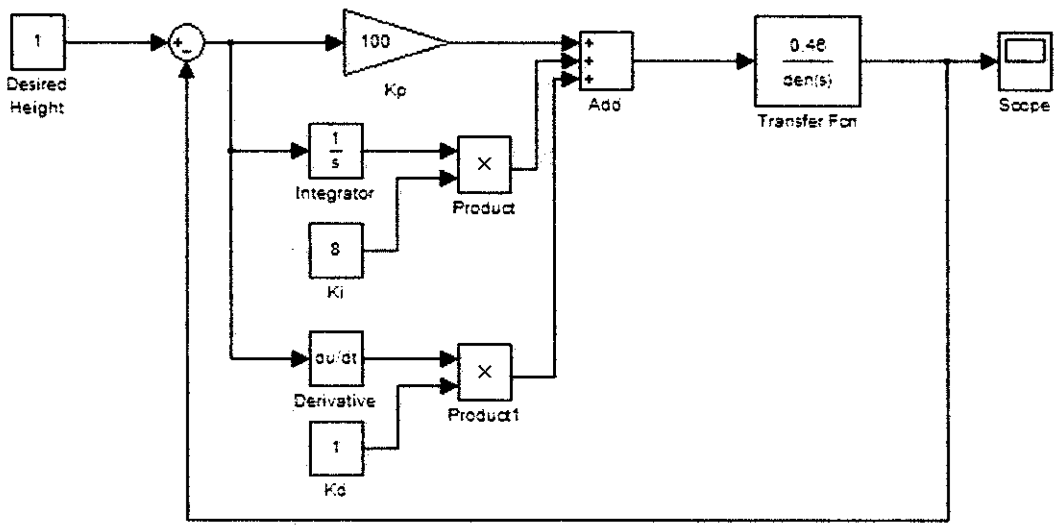


Figure 4.1: PID Tuning Block Diagram

4.2.1 PID Controller Tuning

4.2.1.1 Basic Controller Tuning Block Diagram

The transfer function of the PID Controller is;

$$G_c(s) = K_p(1 + \frac{1}{T_i s} + T_d s)$$

The system under study above has the following basic block diagram;

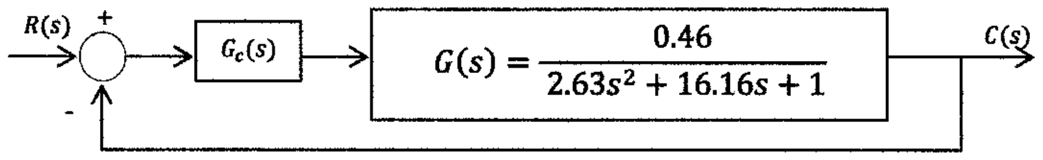


Figure 4.2: Block Diagram of the System with PID

4.2.2 PID Controller Tuning Response

Using the transfer function of the motor, tuning of PID is done to see the effect of tuning onto the system. The choices for the parameters are done by trial and error method. Figure below shows the chosen PID controller tuning response to the system;

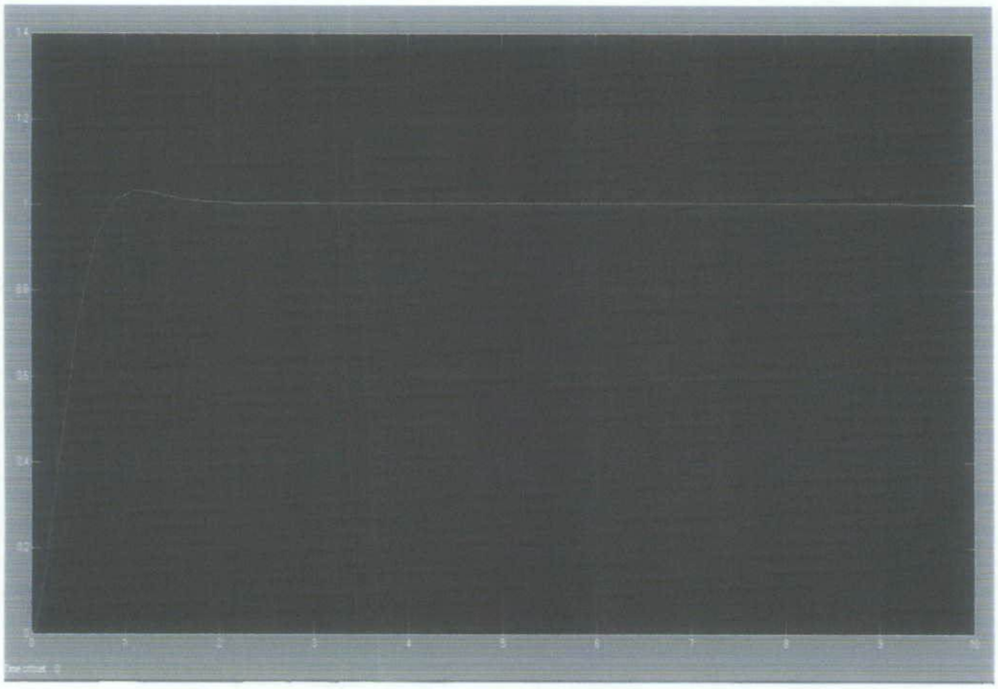


Figure 4.3: System Response for PID Controller Tuning

Controller Parameters:	Corresponding System Performances:	
$K_p = 100$	Rise time (sec)	= 0.85
$K_i = 8$	Settling time (sec)	= 2.14
$K_d = 1$	Overshoot (%)	= 4

From this controller gain values, and the step response that is obtain pertaining to the gain values, the system performances are analyzed. The rise time is taken to reach 10% to 90% of the final value which is about 0.85 seconds. The settling time for the response is about 2.14 seconds while the maximum overshoot is about 4%.

To obtain the best response of the system, a lot of trial and error in the changes of the parameters for P, I and D need to be done. This is proven to be time consuming. The results that we obtained also cannot be proven to be the best value for PID tuning, since it is done manually. However, the system response is acceptable and a good reading is manageable from the tuning done. From this, we try to optimize the plant model to get a better system performances. The system requirements are given below:

Table 1 : System Requirements

System Specs.	**Maximum overshoot**	**Rise time (sec)**	**Settling time (sec)**
Values	< 4%	< 0.85	< 2.14

4.3 Designing of PID Using Genetic Algorithm

4.3.1 Initializing the Population of the Genetic Algorithm

In the first step of implementing Genetic Algorithm technique into a desired plant, the Genetic Algorithm has to be initialized first. The initializations that are vital include the population size, variable bounds and the evaluation function. The following code is utilized for this step. This 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_ITAE';
evalOps=[];
options=[1e-6 1];
initPop=initializega(populationSize,variableBounds,evalFN,...
evalOps,options);
```

Figure 4.4: Coding to Initialize the Genetic Algorithm

Explanations on basic codes that are used in the project are explained below.

- ***PopulationSize***- The first stage in writing codes for Genetic Algorithm is to initialize and determine the number of population to be used for the project. Usually, the population size ranges from 20-100. The best value of population size differs pertaining to the plant model for the project used. Generally, the bigger the population size is the better for the final approximation since that, a bigger population size means a bigger possible optimum solution for the problem.
- ***VariableBounds***– The three row matrix used in the coding are corresponding to the gains of PID controller, since that the project is to optimize the gains of it. The coding illustrate to us that, a population of eighty members are being initialized with values randomly selected between -100 and 100.
- ***evalFN***– This function will fetch the objective function for the project, which is the 'PID_objfun_ITAE' m.file. It will execute the codes and return the value back to the main codes.
- ***Options*** – A real (floating point) numbers is used to encode the population. The value of '1e-6' is the term for floating point precision and the '1' term is used to indicate that the real numbers are being used for this project.
- ***Initializega***– This command correspond to the GAOT toolbox. This command will combine all terms before and creates initial population of 80 real valued members between -100 and 100 with 6 decimal place precision.

4.3.2 Setting The Parameters for The Genetic Algorithm

After initialization of the GA is done, setting of the parameters for the GA is needed to be carried out. The following are codes that are used to set up the GA.

```
%Setting the parameters for the genetic algorithm
bounds=[-100 100;-100 100;-100 100];
evalFN='PID_objfun_ITAE';
evalOps=[];
startPop=initPop;
opts=[1e-6 1 0];
termFN='maxGenTerm';
termOps=200;
selectFN='normGeomSelect';
selectOps=0.08;
xOverFNs='arithXover';
xOverOps=4;
mutFNs='unifMutation';
mutOps=8;
```

Figure 4.5: Coding to Set The Parameters for GA

- ***Bounds*** – The variable bound is used for the Genetic Algorithm to search within a specified area. For this project, the variable bound that is used is same to the one that is used to initialize the population. This bound define the entire search space for the technique.
- ***startPop***– This function is defined as same as the previous function 'initPop'. It serves the same function as 'initPop'.
- ***Opts*** –It consists of the precision for the string value.
- ***TermFN***– This is the code for the termination function for the genetic algorithm. This code will call to another m.file entitled maxGenTerm.m. This is used to terminate the iteration done by Genetic Algorithm once a certain criterion is met.
- ***TermOps***– This command will define the options for the termination function. For the project, a number of 100 termination option is set. This means that the GA will reproduce one hundred generations before terminating the code. This number can be altered to best suit the convergence criteria of the Genetic Algorithm.

- ***SelectFN***–Normalised geometric selection ('normGeomSelect') is used for this project. This function will call to the normGeomSelect.m file in the source code. This m.file is provided by the GAOT toolbox. The GAOT toolbox also gives the user other choices of selection function, which are Tournament selection and Roulette wheel selection. However, tournament selection is not suitable for the MATLAB simulation because it has a longer compilation time than the rest, and this is a big issue for MATLAB software.
- ***SelectOps***– When 'normGeomSelect' is chosen, probability of the fittest chromosomes is declared in this function, which is decided to be 0.08 for the project.
- ***XOverFN***- Arithmetic crossover was chosen as the crossover procedure. Single point crossover is too simplistic to work effectively on a chromosome with three parameters (P, I and D), 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 pointnumbers and is the ideal crossover option for use in this project.
- ***XOverOptions***-This is where the number of crossover points is specified.
- ***mutFNs***- The .multiNonUnifMutation., or multi non-uniformly distributedmutation operator, was chosen as the mutation operator as it is considered tofunction 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 themaximum number of generations and the third parameter is the shape of thedistribution. This last parameter is set to a value of two, three or four wherethe number reflects the variance of the distribution.

4.3.3 Performing The Genetic Algorithm

```
%Iterating the genetic algorithm

[x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,o
pts,termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFN
s,mutOps);
```

Figure 4.6: Coding to Perform the GA

The coding above will iterate the Genetic Algorithm for the project. This iteration will be done up until it fulfills the termination function and the program is terminated. This iteration will return four variables to the user:

x = the best population
endPop = the GA's final population
bestPop = the GA's best solution that is tracked over generation
traceInfo = the best value and average value for each generation.

Figure 4.7 shows how the Genetic Algorithm is converged into its final best solution for the project. The graph used values of $K_p = 98.09443$, $K_i = 6.1111$ and $K_d = 8.70368$

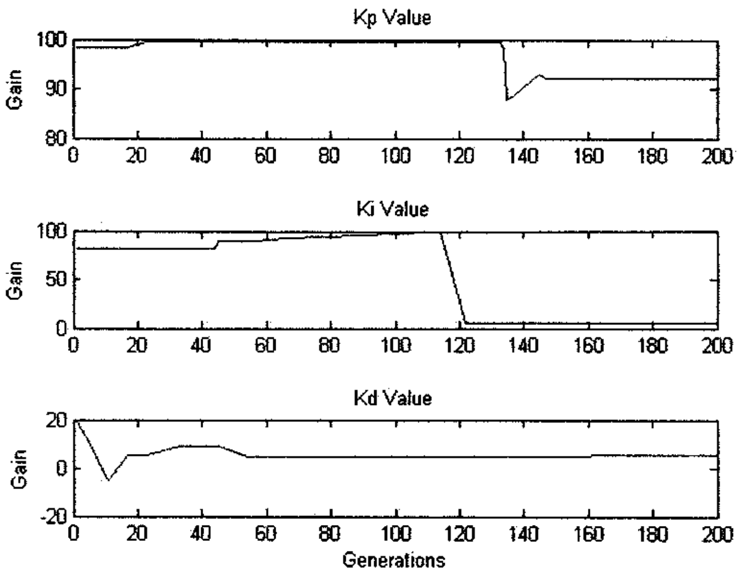


Figure 4.7: Convergence of the PID Chromosomes by GA

4.3.4 Objective Function Of The Genetic Algorithm

One of the most important steps in performing the Genetic Algorithm is determining the best suited objective function prior to the objective of the project itself. After much consideration, the objective function that is chosen to be minimized is the Integral of Time Absolute Error (ITAE) performance criterion. This ITAE is defined as:

$$ITAE = \int_0^T t|e(t)|dt$$

This objective function is created to find a PID controller that gives the smallest overshoot, fastest rise time or quickest settling time. Therefore, in order to combine all these three objectives, the simplest way is to minimize the error of the controlled system instead. Therefore, ITAE is chosen as the objective function.

Apart from that, this ITAE performance index has the advantages of producing smaller system overshoots and oscillations when comparing with other technique which is the IAE (integral of absolute error) and ISE (integral of squared error) performance indices. ITAE is also proven to be the most sensitive of the three, thus having the best selectivity among those. Coding below shown the steps in performing the objective function for the Genetic Algorithm iteration.

```
function [x_pop, fx_val]=PID_objfun_ITAE(x_pop,options)
globalsys_controlled
global time
globalsysrl
%_____________________________________________________________
_______
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=[KdKp Ki];
pid_sys=tf(pid_num,pid_den); %overall PID controller
%Creating PID feedback loop
sys_series=series(pid_sys,sysrl);
sys_controlled=feedback(sys_series,1);
```

Figure 4.8: Coding to Perform the Objective Function for GA

For this coding, each chromosome from the population is passed into the objective function one at a time. Then, the fittest value is evaluated through the objective function. After that, a new population is created consisting of the fittest members for the population beforehand. The chromosomes for this project consist of three separate string of P, I and D term. When the chromosomes pass the evaluation function, they are split into three terms of P, I and D. The gains are used to create a PID controller according to the equation below:

$$C_{pid} = \frac{K_d s^2 + K_p s + K_i}{s}$$

Then, the newly formed PID controller is placed in a unity feedback loop with the system transfer function as shown in the coding before. The controlled system will be given a step input and the error which is based on ITAE for this project is assessed. Below is the code for the implementation of ITAE:

```
%calculating the error
for i=1:301
error(i) = (abs(1-y(i)))*t(i);
end
%Integral of Squared Error
ITAE=sum(error);
```

Figure 4.9: Coding to Implement ITAE

To ensure that the genetic algorithm will converge to a controller that will produce a stable system, a coding which will ensure the stability of the system is written. The coding below shows that the poles of the controlled system is assessed and if they are found to be instable (which is on the s-plane of the system), the error will be assigned an extremely large value as to make sure that the same chromosome (possible solution) will not be selected the other time.

```
%to make sure controlled system is stable
poles=pole(sys_controlled);
if poles(1)>0
ITAE=100e300;
elseif poles(2)>0
ITAE=100e300;
elseif poles(3)>0
ITAE=100e300;

end
fx_val=1/ITAE;
```

Figure 4.10: Coding to Ensure Stabilization of the Controlled System

4.3.6 Results For The GA Based PID Controller

This part of the report will discuss the results that we obtained from the implementation of Genetic Algorithm into the PID controller for the plant model. In the process of completing the project, a lot of parameters for GA need to be decided to use in the programming itself. Table 2 shows the parameters of GA that are used in the project. The selections of all the parameters have been validated in the previous explanation of the main coding for the project.

Table 2: Parameters for GA

GA Property	Value/Method
Maximum number of generation	100
Fitness function	Integral of time absolute error
Selection Method	Normalized Geometric Selection
Probability of selection	0.08
Crossover method	Arithmetic crossover
Number of crossover points	4
Mutation method	Uniform mutation
Mutation probability	0.08

The GA based PID is initially initialized with population size of 20. Then, the system response is observed with different initialization of population size, starting from 30 up to 80 to find the optimum solution. In this project, other parameters (mutation rates and crossover value) are chosen first and set to be the same throughout the testing (after much consideration and observation is done on the implication of other parameters onto the system response) with only changes applied in the population size. The responses of GA based PID are then analyzed to find the smallest overshoot, fastest rise time and the fastest settling time.

The following is the system response of GA based PID with population size of 20, 30, 40, 50, 60, 70 and 80 with their system performances accordingly.

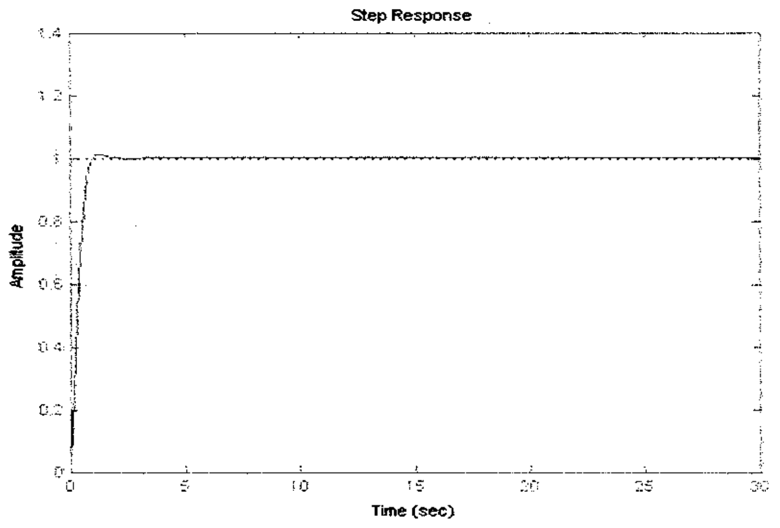


Figure 4.11: GA-PID Response with Population Size 20

Controller Parameters:	Corresponding System Performances:	
$K_p = 99.15879$	Rise time (sec)	= 0.893
$K_i = 6.1941$	Settling time (sec)	= 2.33
$K_d = 3.98398$	Overshoot (%)	= 1

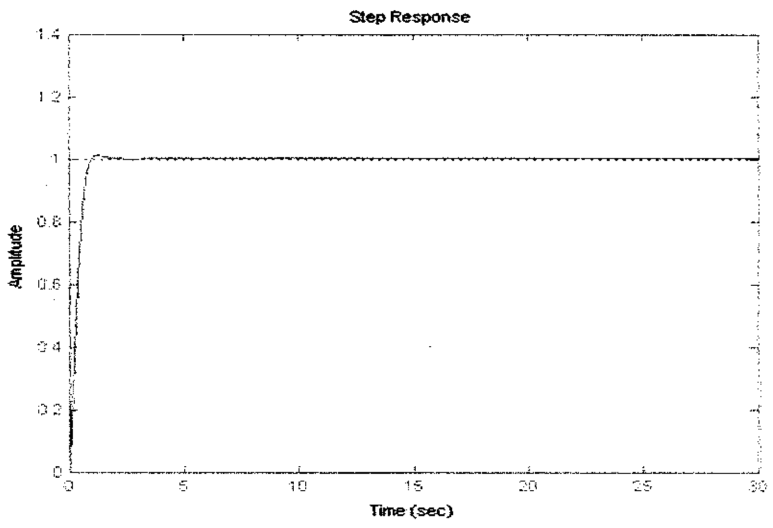


Figure 4.12: GA-PID Response with Population Size 30

Controller Parameters:	Corresponding System Performances:	
$K_p = 99.12296$	Rise time (sec)	= 0.813
$K_i = 6.1892$	Settling time (sec)	= 2.28
$K_d = 4.14580$	Overshoot (%)	= 1

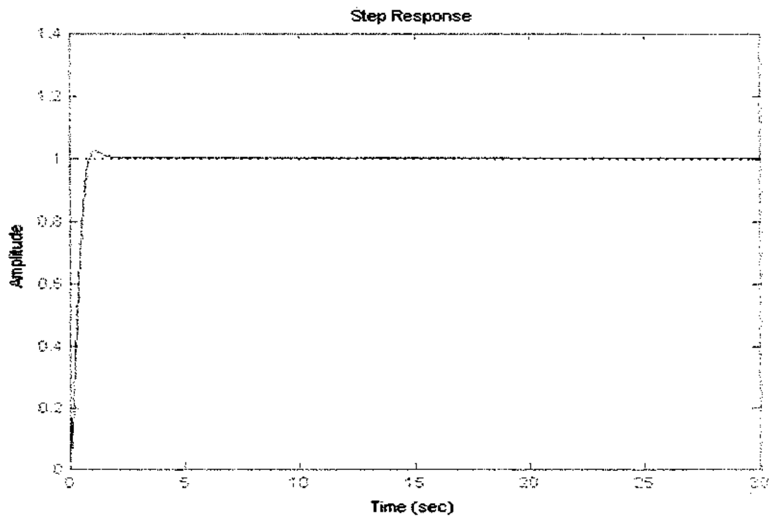


Figure 4.13: GA-PID response with population size 40

Controller Parameters:	Corresponding System Performances:	
$K_p = 99.98597$	Rise time (sec)	= 0.767
$K_i = 6.7199$	Settling time (sec)	= 2.28
$K_d = 2.28531$	Overshoot (%)	= 2

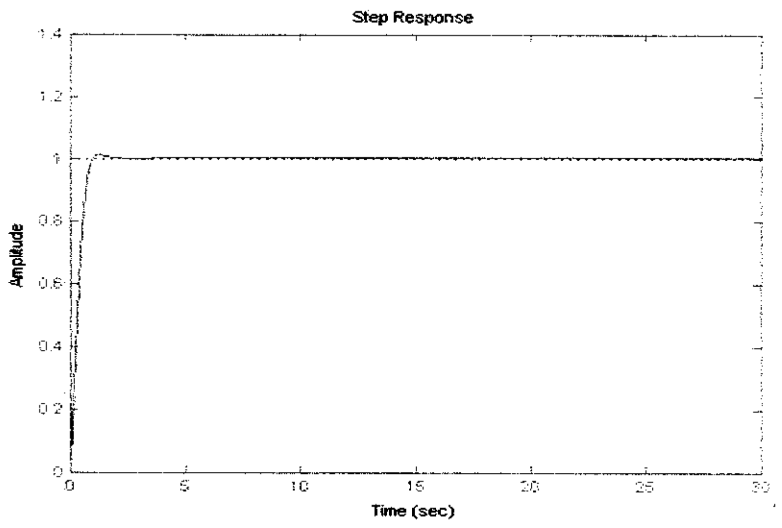


Figure 4.14: GA-PID response with population size 50

Controller Parameters:		Corresponding System Performances:	
$K_p = 98.62480$		Rise time (sec)	$= 0.818$
$K_i = 6.1503$		Settling time (sec)	$= 2.13$
$K_d = 4.06390$		Overshoot (%)	$= 1$

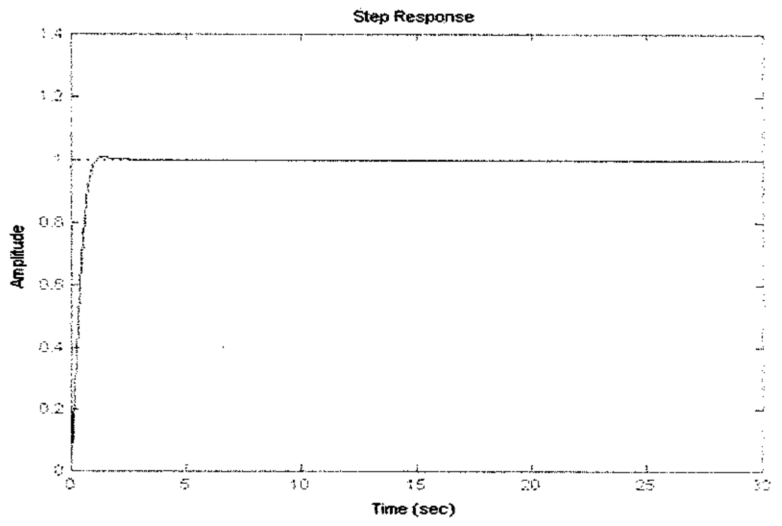


Figure 4.15: GA-PID response with population size 60

Controller Parameters:		Corresponding System Performances:	
$K_p = 94.38349$		Rise time (sec)	$= 0.872$
$K_i = 5.8824$		Settling time (sec)	$= 2.16$
$K_d = 4.60941$		Overshoot (%)	$= 1$

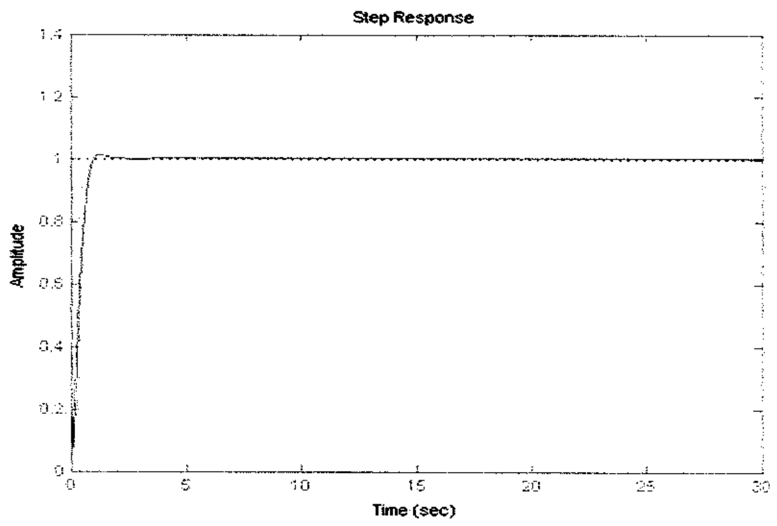


Figure 4.16: GA-PID response with population size 70

Controller Parameters:	Corresponding System Performances:	
$K_p = 95.78919$	Rise time (sec)	= 0.829
$K_i = 5.9774$	Settling time (sec)	= 2.06
$K_d = 3.29274$	Overshoot (%)	= 1

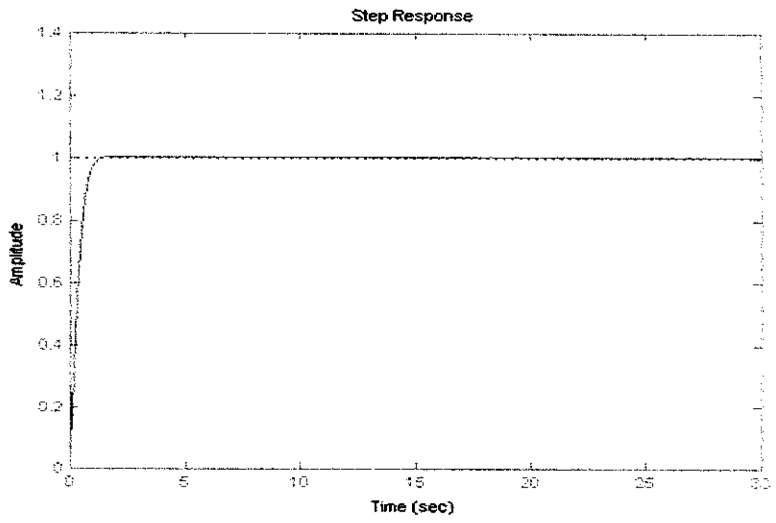


Figure 4.17: GA-PID response with population size 80

Controller Parameters:	Corresponding System Performances:	
$K_p = 98.09443$	Rise time (sec)	= 0.842
$K_i = 6.1111$	Settling time (sec)	= 1.44
$K_d = 8.70368$	Overshoot (%)	= 0

Analysis is done on the system response of GA based PID with differences in the population size. Table 3 shows the results that are obtained.

Table 3: Analysis on GA - PID System Response

Population Size	**Controller Parameters**			**Measuring Factors**		
	K_p	K_i	K_d	**Rise time (sec)**	**Settling Time (sec)**	**Maximum overshoot (%)**
20	99.15879	6.1941	3.98398	0.893	2.33	1
30	99.12296	6.1892	4.14580	0.813	2.28	1
40	99.98597	6.7199	2.28531	0.767	2.28	2
50	98.62480	6.1503	4.06390	0.818	2.13	1
60	94.38349	5.8824	4.60941	0.872	2.16	1
70	95.78919	5.9774	3.29274	0.829	2.06	1
80	98.09443	6.1111	8.70368	0.842	1.44	0

From all the above analysis done on the GA based PID system response for the plant model with the difference in population size, the best system response that is being recognized and detected is the system response of GA based PID with population size of 80. This system response gives rise time of 0.842 seconds, settling time of 1.44 seconds and maximum overshoot of zero percentage.

Comparison is then made between the PID only controller and GA based PID controller for their system performances. Table 4 shows the comparison done on the two techniques:

Table 4: Comparison between PID and GA-PID Controller

Controller Parameters	**PID Controller**	**GA – PID Controller**	**Percentage Improvement**
K_p	100	98.09443	NA
K_i	8	6.1111	NA
K_d	1	8.70368	NA
Measuring Factors			
Rise time (sec)	0.85	0.843	0.82 %
Settling Time (sec)	2.14	1.44	32.71%
Maximum overshoot (%)	4	0	100 %

From table below, results show that, GA based PID controller gives better system performances than PID only controller technique. On average, the percentage of improvement of GA-PID controller against PID controller is around 44.51%. The biggest improvement can be seen in term of the maximum overshoot, where with PID technique, we can find around 4% of maximum overshoot however with GA-PID technique, the plant is optimized such that the maximum overshoot for the plant model is 0%, which is happen to be an ideal case.

The results that we obtained here are correlated with the previous works that are done so far on the GA based PID controller. For instance, Neenu Thomas and Dr. P. Poongodi had found out from their research that, the designed PID with GA has much faster response (improvement about 50%) than the response of classical method for motor tuning [3].

Apart from that, Yingfa Wang, Changliang Xia, Machua Zhang and Dan Liu from Tianjin University in China had also found out that the GA technique that is being applied onto their BLDCM servo system shows that the method not only robust, but also improve dynamic performance of the system (less overshoot, less response time and better stability) when it is compared with PID technique only [13].

Therefore, the results for this project can be testified in accordance to the previous works done so far based on the same optimization technique. Genetic Algorithm is proven to be one of the best optimization techniques available for the tuning of a system and can be applied into the system nowadays.

CHAPTER 5
RECOMMENDATION AND CONCLUSION

5.1 Recommendation

To prove the works done for the PID tuning are correct, conventional method of tuning using Ziegler-Nichols method can be utilized with this model. This conventional method can give better explanation and understanding in term of observing the effect of changing the controller parameters into the motor and finding the best tuning controller method to be used for this motor.

Apart from that, an online system can be applied with this technique to prove the relevancy of the optimization technique online. A higher understanding of the GA itself are needed to ensure the effectiveness of the technique and research can also be conducted to find the best way possible to run GA technique with lesser difficulties.

5.2 Conclusion

In conclusion, the project had shown that the PID controller with GA optimization technique has much faster response than by using the classical method which is PID technique. Apart from that, GA based PID technique also proven to be time savers as they are much faster to be conducted than PID technique which is basically based on trial and error in getting the best PID values before the system can be narrowed down in getting the closest to the 'optimized' value.

As shown in the Table 1 before where comparison is made on the system performances between PID controller and GA-PID controller, a GA-PID controller can give a faster settling time, faster rise time and no overshoot, which is an ideal and stable case. Aside from that, the project has also shown to us that, the initialization made for the population size bring significance effect onto the GA-PID responses of the plant model. A bigger population size gives better chances to the Genetic Algorithm to find the best optimum solution amongst all the chromosomes being initialized earlier. The objectives of the project are successfully achieved.

REFERENCES

[1] Mitsuo Gen, Runwei Cheng, "Genetic Algorithms & Engineering Optimization", JohnWiley & Sons, Inc. 2000

[2] Stephen L. Herman, "Industrial Motor Control", Delmar, Cangage Learning 2010.

[3] Neenu Thomas, Dr. P. Poongodi, "Position Control of DC Motor Using Genetic Algorithm Based PID Controller", Proceedings of the World Congress on Engineering 2009 Vol II,July 1-3, 2009, London, U.K

[4] Saffet Ayasun, Gultekin Karbeyaz, "DC Motor Speed Control MethodsUsing MATLAB/Simulink and Their Integration into Undergraduate Electric Machinery Courses", Wiley Periodicals Inc. 2007

[5] Boumediene Allaoua, Abdellah Laoufi, Brahim Gasbaoui, Abdelfatah Nasri and Abdessalam Abderrahmani, "Intelligent Controller Design for DC Motor Speed Control based on Fuzzy Logic-Genetic AlgorithmsOptimization", Leonardo Journal of Sciences, Issue 13, July-December 2008.

[6] Saifudin bin Mohamed Ibrahim, "The PID Controller Design Using Genetic Algorithm", University of Southern Queensland, 27th October 2005.

[7] M.B.B Sharifian, R. Rahnavard and H. Delavari, 'Velocity Control and DC Motor Based Intelligent Methods and Optimal Integral State Feedback Controller", International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009.

[8] Rahul Malhotra, Narinder Singh and Yaduvir Singh, "Genetic Algorithms: Concepts, Design for Optimization for Process Controllers", Canadian Center of Science and Education, Vol. 4, No. 2, March 2011.

[9] Kondapalli Siva Rama Rao and Azrul Hisham bin Othman, "Design Optimization of a BLDC Motor by Genetic Algorithm and Simulated Annealing", Department of Electrical and Electronics Engineering, Universiti Teknologi PETRONAS.

[10] B. Nagaraj and N. Murugananth, "Soft Computing-Based Optimum Design of PID Controller for a Position Control of DC Motor", Mediamira Science Publisher, 2010.

[11] Ismail K. Bouserhane, Abdeldjebar Hazzab, Abdelkrim Boucheta, Benyounes Mazari and Rahli Mostefa, "Optimal Fuzzy Self-Tuning of PI Controller Using Genetic Algorithm for Induction Motor Speed Control", Int. J. of Automation Technology Vol. 2, No. 2, 2008

[12] Indranil Pan, Saptarshi Das and Amitava Gupta, "Tuning of an optimal fuzzy PID controller with stochastic algorithms for networked control systems with random time delay", Science Direct, ISA Transactions 50 (2011) 28-36

[13] Yingfa Wang, Changliang Xia, Maohua Zhang and Dan Liu, "Adaptive Speed Control for Brushless DC Motors Based On Genetic Algorithm and RBF Neural Network", IEEE International Conference on Control and Automation, Guangzhou, China, May 30 to June 1, 2007.

[14] B. Nagaraj, P. Vijayakumar, "A Comparative Study of PID Controller tuning using GA, EP, PSO and ACO." Journal of Automation, Mobile robotics & Intelligent Systems, vol. 5, no.2, pp.42-48, 2011.

APPENDIX A

Datasheet for Robust DC Motor Drives Deep-Sea Sensors

FAULHABER

DC-Micromotors — **50 mNm**

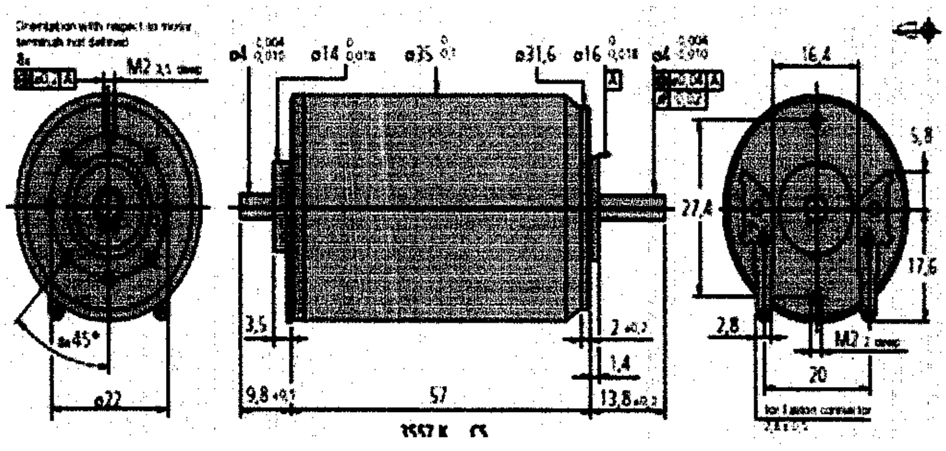
Graphite Commutation

For combination with
Gearheads:
30/1, 30/1 S, 32/3, 32/3 S, 38/1, 38/1 S, 38/2, 38/2 S, 35A
Encoders:
HEDL 5540, HEDM 5500, HEDS 5500, HEDS 5540

Series 3557 ... CS

		3557 K		009 CS	012 CS	020 CS	024 CS	048 CS	
1	Nominal voltage	U_N		9	12	20	24	48	V
2	Terminal resistance	R		0,7	1,34	4	5,5	23	Ω
3	Output power	$P_{2\,max}$		28,1	26,1	24,5	25,4	24,1	W
4	Efficiency, max.	η_{max}		78	79	79	78	76	%
5	No-load speed	n_0		5 700	5 400	5 500	5 500	5 200	rpm
6	No-load current (with shaft ø 4 mm)	I_0		0,19	0,125	0,07	0,065	0,04	A
7	Stall torque	M_H		188	185	169	176	177	mNm
8	Friction torque	M_R		2,8	2,6	2,4	2,7	3,5	mNm
9	Speed constant	k_n		643	456	279	233	110	rpm/V
10	Back-EMF constant	k_E		1,56	2,19	3,59	4,3	9,05	mV/rpm
11	Torque constant	k_M		14,9	20,9	34,2	41	86,5	mNm/A
12	Current constant	k_I		0,067	0,048	0,029	0,024	0,012	A/mNm
13	Slope of n-M curve	$\Delta n/\Delta M$		30,3	29,2	32,5	31,3	29,4	rpm/mNm
14	Rotor inductance	L		100	220	630	850	3 430	µH
15	Mechanical time constant	τ_m		16	16	16	16	16	ms
16	Rotor inertia	J		50	52	47	49	52	gcm²
17	Angular acceleration	α_{max}		37	35	36	36	34	·10³rad/s²
18	Thermal resistance	R_{th1}/R_{th2}	1,5 / 9						K/W
19	Thermal time constant	τ_{w1}/τ_{w2}	15 / 900						s
20	Operating temperature range								
	– motor		–30 ... +125						°C
	– rotor, max. permissible		+125						°C
21	Shaft bearings		ball bearings, preloaded						
22	Shaft load max.:								
	– with shaft diameter		4						mm
	– radial at 3 000 rpm (3 mm from bearing)		30						N
	– axial at 3 000 rpm		5						N
	– axial at standstill		50						N
23	Shaft play								
	– radial	≤	0,015						mm
	– axial	=	0						mm
24	Housing material		steel, zinc galvanized and passivated						
25	Weight		275						g
26	Direction of rotation		clockwise, viewed from the front face						
Recommended values - mathematically independent of each other									
27	Speed up to	$n_{e\,max}$		5 000	5 000	5 000	5 000	5 000	rpm
28	Torque up to ¹⁾	$M_{e\,max}$		50	50	50	50	50	mNm
29	Current up to (thermal limits)	$I_{e\,max}$		3,15	2,26	1,3	1,1	0,54	A

¹⁾ thermal resistance R_{th2} by 40% reduced



3557 K ... CS

APPENDIX B

Full MATLAB Codes Used for Genetic Algorithm Based PID in the Project

PID_GA.m

```
%This programme is being modified from a journal entitled 'The PID
Controller Design Using Genetic Algorithm' by Saifudin bin Mohamed
Ibrahim from University of Southern Queensland in 2005.

clc
clear
close all
globalsys_controlled
global time
globalsysrl
%_______________________________________________________________________
_
den1=[2.63 16.16 1];
num1=[0.46];
sysrl=tf(num1,den1);
%_______________________________________________________________________
_
%Initialising the genetic algorithm
populationSize=80;
variableBounds=[-100 100;-100 100;-100 100];
evalFN='PID_objfun_ITAE';
evalOps=[];
options=[1e-6 1];
initPop=initializega(populationSize,variableBounds,evalFN,...
evalOps,options);
%_______________________________________________________________________
_
%Setting the parameters for the genetic algorithm
bounds=[-100 100;-100 100;-100 100];
evalFN='PID_objfun_ITAE';
evalOps=[];
startPop=initPop;
opts=[1e-6 1 0];
termFN='maxGenTerm';
termOps=200;
selectFN='normGeomSelect';
selectOps=0.08;
xOverFNs='arithXover';
xOverOps=4;
mutFNs='unifMutation';
mutOps=8;
%_______________________________________________________________________
_
%Iterating the genetic algorithm
[x,endPop,bPop,traceInfo]=ga(bounds,evalFN,evalOps,startPop,opts,...
termFN,termOps,selectFN,selectOps,xOverFNs,xOverOps,mutFNs,mutOps);
%_______________________________________________________________________
_

%Creating the optimal PID controller from GA results
ga_pid=tf([x(1) x(2) x(3)],[1 0]);
ga_sys=feedback(series(ga_pid,sysrl),1);
figure(1)
```

```
hold on;
step(ga_sys,'g',0:0.01:30);%Green-genetic algorithm

%_____________________________________________________________________
_
%Printing to screen the PID values
fprintf(' Genetic Algorithm values : Kd = %7.5f  Kp = %7.5f  Ki =
%7.4f ',x(1),x(2),x(3));
%disp( x );
%_____________________________________________________________________
_
%Plotting best population progress
figure(2)
subplot(3,1,1),plot(bPop(:,1),bPop(:,3)),...
title('Kp Value'),, ylabel('Gain');
subplot(3,1,2),plot(bPop(:,1),bPop(:,4)),...
title('Ki Value'),, ylabel('Gain');
subplot(3,1,3),plot(bPop(:,1),bPop(:,2)),...
title('Kd Value'),xlabel('Generations'), ylabel('Gain');
%_____________________________________________________________________
_
```

PID_objfun_ITAE.m

```
function [x_pop, fx_val]=PID_objfun_ITAE(x_pop,options)
globalsys_controlled
global time
globalsysrl
%_______________________________________________________________________
_
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=[KdKp Ki];
pid_sys=tf(pid_num,pid_den); %overall PID controller
%Creating PID feedback loop
sys_series=series(pid_sys,sysrl);
sys_controlled=feedback(sys_series,1);
%_______________________________________________________________________
_
time =0:0.1:30;
[y t] = step(sys_controlled,time); % Step response of closed-loop
for i=1:301
error(i) = (abs(1-y(i)))*t(i);
end
%Integral of Squared Error
ITAE=sum(error);

%_______________________________________________________________________
_
%to make sure controlled system is stable
poles=pole(sys_controlled);
if poles(1)>0
ITAE=100e300;
elseif poles(2)>0
ITAE=100e300;
elseif poles(3)>0
ITAE=100e300;

end
fx_val=1/ITAE;
%___________________________________
```

maxGenTerm.m

```
function [done] = maxGenTerm(ops,bPop,endPop)
% function [done] = maxGenTerm(ops,bPop,endPop)
%
% Returns 1, i.e. terminates the GA when the maximal_generation is
reached.
%
% ops    - a vector of options [current_genmaximum_generation]
% bPop   - a matrix of best solutions
[generation_foundsolution_string]
% endPop - the current generation of solutions

% Binary and Real-Valued Simulation Evolution for Matlab
% Copyright (C) 1996 C.R. Houck, J.A. Joines, M.G. Kay
%
% C.R. Houck, J.Joines, and M.Kay. A genetic algorithm for function
% optimization: A Matlab implementation. ACM Transactions on
Mathmatical
% Software, Submitted 1996.
%
% This program is free software; you can redistribute it and/or
modify
% it under the terms of the GNU General Public License as published
by
% the Free Software Foundation; either version 1, or (at your
option)
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details. A copy of the GNU
% General Public License can be obtained from the
% Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,
USA.

currentGen = ops(1);
maxGen     = ops(2);
done       = currentGen>= maxGen;
```

normGeomSelect.m

```
function[newPop] = normGeomSelect(oldPop,options)
% NormGeomSelect is a ranking selection function based on the
normalized
% geometric distribution.
%
% function[newPop] = normGeomSelect(oldPop,options)
% newPop  - the new population selected from the oldPop
% oldPop  - the current population
% options - options to normGeomSelect [gen
probability_of_selecting_best]

% Binary and Real-Valued Simulation Evolution for Matlab
% Copyright (C) 1996 C.R. Houck, J.A. Joines, M.G. Kay
%
% C.R. Houck, J.Joines, and M.Kay. A genetic algorithm for function
% optimization: A Matlab implementation. ACM Transactions on
Mathmatical
% Software, Submitted 1996.
%
% This program is free software; you can redistribute it and/or
modify
% it under the terms of the GNU General Public License as published
by
% the Free Software Foundation; either version 1, or (at your
option)
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details. A copy of the GNU
% General Public License can be obtained from the
% Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,
USA.

q=options(2);                % Probability of selecting the best
e = size(oldPop,2);              % Length of xZome, i.e. numvars+fit
n = size(oldPop,1);              % Number of individuals in pop
newPop = zeros(n,e);             % Allocate space for return pop
fit = zeros(n,1);            % Allocates space for prob of select
x=zeros(n,2);                    % Sorted list of rank and id
x(:,1) =[n:-1:1]';           % To know what element it was
[y x(:,2)] = sort(oldPop(:,e));      % Get the index after a sort
r = q/(1-(1-q)^n);           % Normalize the distribution, q prime
fit(x(:,2))=r*(1-q).^(x(:,1)-1);     % Generates Prob of selection
fit = cumsum(fit);           % Calculate the cumulative prob. func
rNums=sort(rand(n,1));           % Generate n sorted random numbers
fitIn=1; newIn=1;            % Initialize loop control
whilenewIn<=n               % Get n new individuals
if(rNums(newIn)<fit(fitIn))
newPop(newIn,:) = oldPop(fitIn,:);   % Select the fitIn individual
newIn = newIn+1;             % Looking for next new individual
else
fitIn = fitIn + 1;           % Looking at next potential selection
end
end
```

arithXover.m

```
function [c1,c2] = arithXover(p1,p2,bounds,Ops)
% Arith crossover takes two parents P1,P2 and performs an
interpolation
% along the line formed by the two parents.
%
% function [c1,c2] = arithXover(p1,p2,bounds,Ops)
% p1      - the first parent ( [solution string function value] )
% p2      - the second parent ( [solution string function value] )
% bounds  - the bounds matrix for the solution space
% Ops     - Options matrix for arith crossover [gen #ArithXovers]

% Binary and Real-Valued Simulation Evolution for Matlab
% Copyright (C) 1996 C.R. Houck, J.A. Joines, M.G. Kay
%
% C.R. Houck, J.Joines, and M.Kay. A genetic algorithm for function
% optimization: A Matlab implementation. ACM Transactions on
Mathmatical
% Software, Submitted 1996.
%
% This program is free software; you can redistribute it and/or
modify
% it under the terms of the GNU General Public License as published
by
% the Free Software Foundation; either version 1, or (at your
option)
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details. A copy of the GNU
% General Public License can be obtained from the
% Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,
USA.

% Pick a random mix amount
a = rand;

% Create the children
c1 = p1*a     + p2*(1-a);
c2 = p1*(1-a) + p2*a;
```

unifMutation.m

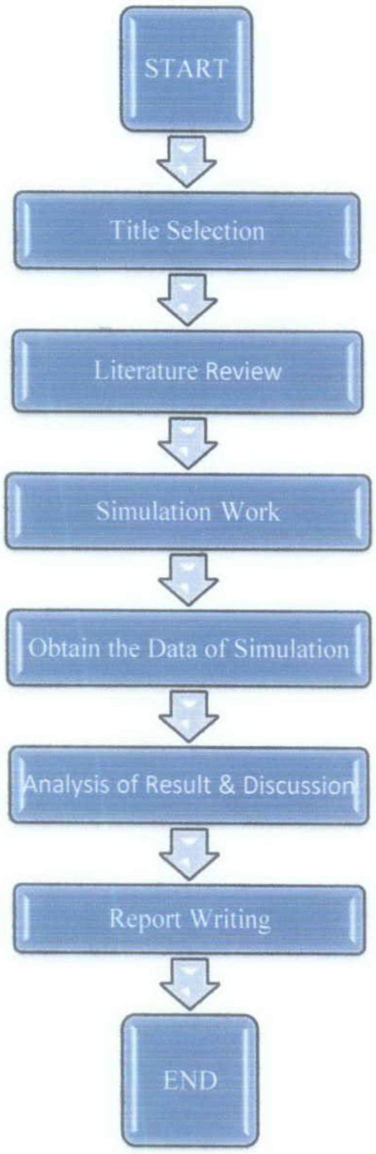
```
function [parent] = uniformMutate(parent,bounds,Ops)
% Uniform mutation changes one of the parameters of the parent
% based on a uniform probability distribution.
%
% function [newSol] = multiNonUnifMutate(parent,bounds,Ops)
% parent  - the first parent ( [solution string function value] )
% bounds  - the bounds matrix for the solution space
% Ops     - Options for uniformMutation [gen #UnifMutations]

% Binary and Real-Valued Simulation Evolution for Matlab
% Copyright (C) 1996 C.R. Houck, J.A. Joines, M.G. Kay
%
% C.R. Houck, J.Joines, and M.Kay. A genetic algorithm for function
% optimization: A Matlab implementation. ACM Transactions on
Mathmatical
% Software, Submitted 1996.
%
% This program is free software; you can redistribute it and/or
modify
% it under the terms of the GNU General Public License as published
by
% the Free Software Foundation; either version 1, or (at your
option)
% any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details. A copy of the GNU
% General Public License can be obtained from the
% Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,
USA.

df = bounds(:,2) - bounds(:,1);      % Range of the variables
numVar = size(parent,2)-1;       % Get the number of variables
% Pick a variable to mutate randomly from 1-number of vars
mPoint = round(rand * (numVar-1)) + 1;
newValue = bounds(mPoint,1)+rand * df(mPoint); % Now mutate that
point
parent(mPoint) = newValue;       % Make the child
```

APPENDIX C

Project Activity Flow



APPENDIX D

Gantt Chart for FYP I

No	Detail / Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Selection of Project Topic	🟥	🟥												
2	Preliminary research work			🟥	🟥	🟥									
3	Submission of extended proposal defense						🟥								
4	Developing the model for DC motor control							🟥							
5	Proposal Defense								🟥	🟥					
6	Applying PID controller into the model of the motor										🟥				
7	Tuning of the controller using conventional method											🟥			
8	Implementing Genetic Algorithm into the motor control (reproduction phase)												🟥		
9	Submission of Interim Draft Report													🟥	
10	Submission of Interim Report														🟥

APPENDIX E

Gantt Chart for FYP II

No	Detail / Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	Project Work Continue (second phase of GA)	🟥	🟥	🟥	🟥	🟥	🟥	🟥								
2	Submission on Progress Report								🟥							
3	Project Work Continue (third phase of GA)								🟥	🟥	🟥	🟥	🟥			
4	Pre-EDX											🟥				
5	Submission of Draft Report												🟥			
6	Submission of Dissertation (soft bound)													🟥		
7	Submission of Technical Paper													🟥		
8	Oral Presentation														🟥	
9	Submission of Project Dissertation (Hard Bound)															🟥