A Comparison Study of Genetic Algorithms In Feedback Controller Design

by

Nga Hin Benjamin Fong

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Mechanical Engineering

APPROVED:

Prof. H. H. Robertshaw, Chairman

Prof. A. L. Wicks

Prof. H. H. Cudney

December 8, 1994

Blacksburg, Virginia

A Comparison Study of Genetic Algorithms In Feedback Controller Design

by

Nga Hin Benjamin Fong

Committee Chairman: Harry H. Robertshaw

Mechanical Engineering

(ABSTRACT)

This thesis discusses the use of genetic algorithms as a global search technique to solve three optimization problems: a sixth-order polynomial problem, a single-degree-of-freedom spring-mass-damper (SDOF SMD) system problem, and a loading bridge regulator problem. Genetic algorithms are iterative global search techniques based on the principles of natural selection and population genetics. The theory, design and implementation of the algorithm is discussed in detail.

The Simple Genetic Algorithm (SGA) is presented to solve a sixth-order polynomial optimization problem. Results from two traditional numerical techniques will be compared with the SGA results as well as the analytical calculus solution. In addition, the effect of different parametric sizes of the genetic operators are investigated.

In the second problem, genetic algorithms are used to design a two-state feedback optimal gain set for a SDOF SMD model with a given initial condition. An improved selection scheme called the stochastic remainder selection without replacement is

introduced. An improved GA-based (IGA) feedback controller is designed to control the system.

Lastly, a regulator control problem is presented using advanced genetic algorithms (AGA). Two-point crossover and inversion operators are employed. A loading bridge is chosen as the control model. An advanced GA-based full-state feedback controller is designed to control the loading bridge with the given reference input voltage.

The conclusions show that SGA is more robust than traditional numerical techniques to solve multi-modal functions. Among the three GA approaches considered, AGA is the most robust one for the design of adaptive feedback controllers.

Acknowledgements

I would like to take this opportunity to thank those who have helped me in this endeavor. First of all, I would like to thank Dr. Harry Robertshaw for serving as my advisor and committee chairman. It is unquestionable that without his encouragement and motivation, this thesis would not have been completed. I would also like to thank Dr. A. L. Wicks and Dr. H. H. Cudney for being on my committee and for their support.

Special thanks goes to Dan Cole, Darrell McAlister and Tom Snyder for their invaluable assistance. Dan, thanks for your support throughout the entire research. You are the man who guided me towards studying GA. "Hey, man! It's been a long time.....but I won't forget you!" Darrell, thanks for fixing all the hardware and software problems with the computers. Tom, thanks for proofreading my thesis and teaching me how to watch football. Thank you guys for the innumerable hours you took off from your research to help me. I would also like to thank everyone else in the Smart Structures Lab (Peter Tappert, Daniel Miller, Will Saunders and Jim Pascoe) for their help and advice. Hey, Pete ("Mega Man"), best wishes on your new job in Oregon.

I would also like to thank all my friends outside the lab who have provided support throughout my research. Thank you Jesse, Stanley, David and Katherine for your help.

Lastly, I would like to express my appreication to my parents for their unyielding love and support. Thanks also goes to my eldest brother and sister-in-law in Hawaii, Ricky & Susanna. I also thank Joe, my elder brother from Houston, for his continuous support throughout the years. Thanks to God for giving me such a loving family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In conventional control theory, ordinary fixed-gain, linear feedback control works well with systems which are linear and regulated at fixed operating points. However, as changes occur in the plant dynamics due to variations in the environment, ordinary fixed-gain, linear feedback controllers may not be able to obtain the desired performance. In addition, disturbances, nonlinear behavior, changes in the nature of inputs and unknown system parameters may also lead the conventional controller to perform at an unacceptable level. Therefore adaptation may be needed in order to change system parameters as rapidly and as frequently as the situation demands. A system which can be self-adjusted or self-modified under the condition of a changing environment or structure is called an adaptive control system. Adaptive control systems can be divided into two main categories: feedforward adaptive controllers and feedback adaptive controllers.

One of the earliest approaches to feedforward adaptive control is gain scheduling, which was used for the first time in the early 1950s in aircraft control [2]. The idea is to find auxiliary process variables (other than the plant outputs used for feedback) that correlate well with the changes in process dynamics. It is then possible to compensate for plant parameter variations by changing the parameters of the regulator as functions of the

auxiliary variables. The advantage of feedforward adaptive control is its high speed of adaptation in response to process changes. The disadvantage is that it is an open-loop adaptation scheme, with no feedback or learning capability from its output response. Moreover, it requires a large amount of parameter storage to accommodate many operating conditions.

If the process behavior changes cannot be determined directly by measurement of external process signals, feedback adaptive controllers have to be used. Feedback adaptive controllers can be subdivided into dual adaptive and nondual adaptive controllers. Under a dual control system, the optimal regulator maintains a balance between the control activity for learning about the plant it is controlling and the activity for controlling the plant output to its desired value. The design of optimal dual controllers is very complex and is commonly found in the area of stochastic control. It can only be performed numerically by using the principle of dynamic programming which was developed by Bellman in 1961 [23]. The nondual adaptive controllers minimize the design performance criterion which requires only present and past values of the control loop signals, and current information about the process, state or signal estimates. No future estimation is taken into account. The design procedure of nondual adaptive controllers is closely related to the separation and certainty equivalence principles. Several types of nondual adaptive controllers have been developed in the past two decades [2,3,23,41].

For example, in a model reference adaptive control (MRAC) system, the desired closed-loop performance is specified through a given reference model. The adaptive

system attempts to make the output of the plant match asymptotically with the output of the given reference model. The scheme discussed above is called the direct method, because the controller parameters directly update the adjustment between the model and the plant outputs. When the controller parameters are updated indirectly, the system is called indirect. With indirect control, the plant parameters are estimated and the control parameters are adjusted on the basis of these estimates. A controller of this construction is called a self-tuning regulator (STR), which was originally introduced by Åström and Wittenmark in 1973 [3]. It was also classified as a model identification adaptive controller (MIAC) by Isermann in 1991 [23]. In STR, a pre-tune mode is introduced first to help in obtaining the required prior information. The unknown plant parameters are estimated and identified using a recursive estimation technique. The controller parameters are then obtained by the regulator design which represents an on-line solution for a system with known parameters. This design is classified as the underlying design problem. STR is very flexible with respect to its choice of controller design and identification scheme. For the design of the controller, we can use linear quadratic regulator, pole placement, minimum variance, gain-phase margin design, etc. Several methods can be used for the design of the identifier, including recursive least squares, extended Kalman filtering, and maximum likelihood. The advantage of using an adequately designed STR is its capability to adapt to any unmeasurable disturbances. As a remark, MRAC can be used with continuous-time as well as discrete-time systems, whereas STR works only with discrete-time systems.

Although adaptive techniques like MRAC and STR seemed to be ideal tools for performing automatic tuning, they were found insufficient because of their requirement of prior information. One special technique called auto-tuning was developed to fine-tune the parameter settings of classical PID controllers [18]. One of the most well-known methods, Ziegler-Nichols ultimate-cycle tuning, is used to tune the parameters during open-loop experiments. One drawback of this method is the time consumption for operating the process. An improved technique called relay feedback was developed to control the amplitude and frequency of the self-oscillation of a closed-loop system. One advantage of auto-tuning is that the tuning experiment is initiated and can be supervised by an operator.

In the 1970's, some researchers started to investigate a new form of adaptive control system which combined the concept of conventional adaptive control theory and artificial intelligence (AI) with the use of powerful high-speed computers. They called this system an expert system or knowledge-based system. Several knowledge-based techniques have been developed in the past two decades, such as fuzzy logic, neural networks, and genetic algorithms. An expert system has human like decision-making ability to recognize, quantify and adapt to any changing condition of the problem environment. Recently, researchers have focused on applying real-time knowledge-based techniques to adapt to changing control environments.

In most adaptive control models, control gain parameters may be changed due to variations in the system and the presence of disturbances. A self-decisive algorithm will be

adequate to identify and make appropriate adaptations in these different situations. Genetic algorithms are being recognized as one of the robust optimal global search techniques.

1.2 Objective

The objective of this thesis is to investigate the abilities and efficiencies of using genetic algorithms (GA) in the design of an optimal regulator for feedback control systems. Three optimal engineering problems will be solved using genetic algorithms. In addition, improved and advanced genetic algorithms will be employed to address their superiorities over simple genetic algorithms (SGA). Finally, initial studies and investigations into the performing of real-time or on-line adaptive control using genetic algorithms will be discussed.

1.3 Contents of Thesis

The rest of the thesis is outlined as follows. Chapter 2 presents a literature review of the developement of genetic algorithms and their recent applications. Chapter 3 discusses the foundamental concept of simple genetic algorithms (SGA). A general scheme for implementing the algorithm will be presented. Simple genetic algorithms are used to solve a sixth-order polynomial function problem. Selection of genetic parameters to influence the efficiency of the algorithm will be discussd. Chapter 4 presents two control problems

solved using GA-based controllers. The first is the solution of an initial condition problem with a Single-Degree-Of-Freedom (SDOF) Spring-Mass-Damper (SMD) system model. An improved genetic algorithm (IGA) will be introduced to demonstrate its superiority over the simple genetic algorithm. The second problem concerns a control regulator problem using a loading bridge model. Advanced genetic algorithms (AGA) are used to solve this problem. Inversion and two-point crossover operators are discussed in this section. In both sections, the Linear Quadratic Regulator (LQR) method will provide the optimal solutions as a reference to the GA-based solutions. In chapter 5, an overall conclusion on the efficiencies and advantages of using genetic algorithms as an optimal regulator will be stated. Initial studies on the concept and approach of performing real-time GA-based adaptive control will be discussed for further recommendations.

Chapter 2

Literature Review

2.1 Introduction

This section presents the literature review of the development of genetic algorithms and their applications. Genetic algorithms are defined by Goldberg [16] as *search algorithms based on the mechanics of natural selection and natural genetics.* The underlying principles of genetic algorithms were first published by John Holland in 1962 [16]. The mathematical framework was developed in the late 1960's, and was presented in Holland's pioneering textbook, Adaptation in Natural and Artificial Systems [19], which was published in 1975. Genetic algorithms are known to have the capability to seek near-optimal solutions to complex problems. The efficiency of using genetic algorithms for optimizing functions was proved by De Jong in 1975 [7]. De Jong's study showed that genetic algorithms give a robust search method applicable across a wide set of problems, in contrast to conventional calculus-based methods. Genetic algorithms have been used in many diverse areas, such as engineering, computer science, operations research, biology, business and social sciences. In the last decade, research devoted to genetic algorithms has significantly increased as a result of several conferences and publications on the topic. Two excellent references on the concept and the implementation of genetic algorithms are the texts written by David Goldberg and Lawrence Davis, Genetic Algorithms in Search,

Optimization, and Machine Learning [16] in 1989; and Handbook of Genetic Algorithms [5] in 1991, respectively.

2.2 Early Developments in the Theory of Genetic Algorithms

Prior to the use of genetic algorithms for solving system optimization problems, a number of biologists performed simulations on genetic systems using digital computers (Barricelli, 1957, 1962; Fraser, 1960, 1962; Martin and Cockerham, 1960) [16]. In 1962, John Holland established the basis of genetic algorithms with his writings on adaptive systems theory. Holland's main objective was to develop the theory and procedures necessary for creating an algorithm which could adapt to changing environments. During the developing period (1962-1965) [16], Holland was a professor teaching adaptive systems at the University of Michigan. His students had written simulations of applying genetic adaptive algorithms. But unfortunately, most of these written records were lost. For the rest of the decade, Holland and his students investigated and explored the theory of genetic adaptive systems into mathematical foundation. By the end of the decade, the Fundamental Theorem of Genetic Algorithms was established.

The words "genetic algorithm" and its application were first published in Bagley's pioneering dissertation in 1967 [16]. Bagley's genetic algorithm was similar to the one we use today. He created reproduction, crossover, and mutation operators similar to those described in simple genetic algorithms. In addition, he used dominance, inversion, and

diploid string representations which are described as advanced operators by Goldberg. Working at the same time as Bagley, Rosenburg also investigated the capability of genetic algorithms [16]. His simulation was the first application of genetic algorithms to a root-finding problem.

In 1970, Cavicchio [16] applied genetic algorithms to two problems of artificial search: a subroutine selection problem and a pattern recognition problem. He also invented three mutation operators and employed inversion, two-point crossover, and intrachromosomal duplication. One newness mechanism found in his study was called a preselection scheme. Here, a good offspring replaced one of its parents in the hope of maintaining population diversity. A similar scheme was later adopted successfully by De Jong (1975) [7] in an optimization study. In the same year, Weinberg [16] stated in detail but did not simulate an interesting problem of genetic algorithm optimization in his dissertation study. He proposed the use of a multilayered genetic algorithm to select a good set of 15 rate constants that controlled the workings of different simulated Escherichia coli cells.

In 1970, Hollstien's dissertation [16] was the first to apply genetic algorithms to a pure problem of mathematical optimization. The title of his dissertation, "Artificial Genetic Adaptation in Computer Control Systems," suggested the application of genetic algorithms to digital feedback control of some engineering plant. The work was involved with optimizing functions of two variables using dominance, crossover, mutation, and several selection schemes. In 1972, Frantz constructed combined linear-nonlinear

functions over binary haploid chromosomes [16]. He also examined the effects on position of several functions where the order of chromosome was changed to affect the length of particular building blocks. His experiments with the reordering operator, inversion, were inconclusive because the concept of genetic algorithm deception was not developed at the time of Frantz's investigation. (The concept of deception is discussed in the texts by Holland and Goldberg [19, 16].)

Nineteen-seventy-five was a particularly good year for genetic algorithms. Holland [19] published his pioneering textbook, "Adaptation in Natural and Artificial Systems." In the same year, De Jong [7] finished his pivotal dissertation, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems." De Jong's study stands as a breakthrough in the development of genetic algorithms because of its combination of Holland's theory of schemata and his own thorough computational experiments. Similar to Hollstien's earlier study, De Jong's work acknowledged genetic algorithms in the framework of function optimization. He was interested in applying genetic algorithms in data structure design, algorithm design, and computer operating system adaptive control. De Jong constructed a test environment of solving five minimization function problems. (Detailed discussion can be found in De Jong's dissertation or Goldberg's text.)

In order to indicate the effectiveness of different genetic algorithms, De Jong contrived two measures, one to gauge convergence and the other to gauge ongoing performance. He called these measures off-line (convergence) and on-line (ongoing) performance respectively. In an off-line application, many function evaluations were

simulated and the best option was saved and used after some stopping criteria were executed. In on-line application, function evaluations were terminated through real time experimentation. As Goldberg stated in his text [16]: *On-line performance is an average of all function evaluations up to and including the current trial. Off-line performance is a running average of the best performance values to a particular time.*

De Jong's results showed that larger populations guided to better final off-line performance because of the large diversified schemata found in a large population. (The concept of schema will be discussed in next chapter.) A larger population would also lead to poorer initial on-line performance. On the other hand, his results showed that smaller populations had the capability to adapt rapidly and thus experienced a better initial on-line performance.

Finally, in 1983, David Goldberg finished his dissertation [14, 15], "Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning." His work applied genetic algorithms to optimization and machine learning problems in natural gas pipeline control.

2.3 Recent Applications of Genetic Algorithms

After Holland's pioneering textbook had published in the mid seventies, researchers started to focus more on the applications of genetic algorithms. Several conferences have been held on the topic. The first international conference on genetic algorithms was held

at San Mateo in 1985. After the conference, there was an increased number of publications found on the topic. Following is a review of the recent applications using genetic algorithms to solve different types of control problems. The problems are solved using different control approaches combined with the use of modified genetic algorithms.

In 1989, Kristinsson and Dumont [29] applied simple genetic algorithms in system identification control problem. Instead of using recursive and local search techniques for performing system identification, they used genetic algorithms to estimate the poles and zeros of the dynamic system. In 1992, Kristinsson and Dumont [28] further investigated simple genetic algorithms for an adaptive control problem. The algorithm was implemented as an estimator for discrete time and continuous time systems. Both minimum and nonminimum phase plants were used to identify the system parameters. The recursive instrumental variable (RIV) identification method was used to compare the results with genetic algorithms. Genetic algorithms proved to be robust and able to converge toward the actual values of the parameters.

In 1990, Michalewicz, Krawczyk, Kazemi and Janikow [35] used simple genetic algorithms to solve two discrete-time, numerical optimization control problems. The numerical results were compared with a search-based computational package called GAMS. In the same year, Janikow and Michalewicz [25] introduced a specialized genetic algorithm to solve the same numerical optimization problems. The specialized version of the genetic algorithm was designed to work on numerical parameter optimization problems with real valued domains. The main purpose was to move the algorithm closer

to the problem space and made use of some real space specific characteristics. One of the advantages using the specialized genetic algorithm is that it is well suited to perform local fine tuning and is less function dependent. Furthermore, the specialized GA obtained much higher precision solutions and lower time complexity than the simple genetic algorithm approach.

In 1991, Thierens and Vercauteren [48] developed two new genetic operators called activation-masked-crossover and topological-mutation. The modified genetic algorithm was used to solve a classical cart-pole or inverted pendulum control problem. The experimental result showed that the algorithm performs well to control a discrete partitioning of the input space which has nearly 25 times more subspaces than the past result. In the same year, Huang and Fogarty [22] applied two new versions of genetic algorithm to the same cart-pole balancing problem. They introduced the incremental version of genetic algorithm (IGA) to learn the classification of the state-space control process and the batch version of genetic algorithm (BGA) to optimize a set of control actions. The result showed that the algorithm is capable of learning partitioning of the state space and optimizing a complete set of control actions for balancing the pole without any domain knowledge or prior information.

The classical cart-pole model was simulated using a genetic-based technique by McGregor, Odetayo and Dasgupta in 1992 [33]. A structured genetic algorithm was developed to perform the same task. The result showed an improvement over the simple genetic algorithm in its robustness and optimization speed. Similar results were obtained

by simulating the cart-pole system using simple genetic algorithms [51, 53, 9].

Examples of applying genetic algorithms to active controlled structures were done in 1991 by Curtis [4]; Rao, Pan and Venkayya [39]. In 1992, genetic algorithms had been successfully applied to problems in a wide variety of different areas. Ueyama, Fukuda, and Arai [50] developed a distributed genetic algorithm to perform the coordinate planning for a structure configuration of the Cellular Robotic System (CEBOT). Szarkowicz [47] introduced an adaptive-coding genetic algorithm to solve and obtain near-optimal minimum-time solutions to a pair of brachistochrone problems. Maclay and Dorey [32] used the simple genetic algorithm to investigate the nonlinear internal behavior of a vehicle engine and drive train, and its acceleration response to changes in throttle position. The result was compared to a quadratic, gradient based method called the Levenberg-Marquartd (M-L) method. Huang and Chan [21] demonstrated the optimal performance of using simple genetic algorithms to solve two numerical function problems. In 1994, Solano and Jones [44] applied genetic algorithms to control a robot manipulator to find feasible minimum distance paths between two configurations without colliding with obstacles present in its workspace.

In 1992, Krishnakumar and Goldberg [27] applied simple genetic algorithms to two feedback flight control system design problems. The two models were a lateral autopilot and a wind shear controller. Simple genetic algorithms were implemented and the state feedback gains were designed based on a quadratic performance index. Each gain in the gain set was represented by a 7-bit string. All strings were concatenated to

produce a full gain set which represented one feedback design. The fitness function was derived from the cost function. (Detailed work for the optimization of control gains problem will be discussed in later chapters.) Powell's conjugate direction method and the Linear Quadratic Regulator (LQR) method were used to compare the results. It showed that the genetic algorithm solution matched with the LQR results better than Powell's method did.

Tuning of the proportional-integral-derivative (PID) control parameters is a common example found in most control optimization problems. In the past few years, several researchers had focused on the investigation of designing genetic-based fine-tuning PID controllers. Several conference publications were found under this topic [38, 45, 54, 51, 30, 10]. Again, a detailed discussion on the design of control gain parameters using genetic algorithms is discussed in later chapters. Genetic algorithms have recently been applied to the fuzzy control design problem. In 1992, Wang and Kwok [55] used the simple genetic algorithm technique to design a classical PID controller with the fuzzy logic principle. In the same year, Homaifar and McCormick [20] developed the genetic-based fuzzy logic controller to solve the cart-centering problem.

In the next few chapters, the fundamental theory of genetic algorithms will be discussed. Traditional optimization techniques are used to compare the results with the GA method. Two control optimization problems will be solved using advanced genetic operators and an improved selection scheme.

Chapter 3

Genetic Algorithms

3.1 Introduction

Many techniques are used today for optimizing engineering design problems. Most of these techniques fall under two general categories: calculus-based methods and enumerative schemes. Although these techniques are well developed, they maintain significant drawbacks. Calculus-based methods are local in scope, i.e., the optima they seek are the ones closer to the current point. Obviously, starting the search near a lower peak region can lead us to miss the global peak. Once the lower peak is reached, further improvement must be sought through a random restart. In addition, calculus-based techniques rely on the existence of derivatives. Calculus-based methods are insufficiently robust over the broad spectrum of optimization functions. Many enumerative schemes have been suggested to overcome the deficiency of calculus-based methods. However these schemes lack efficiency because many practical search spaces are too large to seek one at a time. Another type of technique that has achieved increasing popularity is the random search algorithm. Nevertheless, the algorithm must also be discounted because its lack of efficiency. Random searches, in the long run, can be expected to do no better than enumerative schemes.

There is one technique, called the genetic algorithm (GA), which is both global and robust over a broad spectrum of problems. The genetic algorithm is an iterative global search technique based on the principles of population genetics and natural selection. It combines the Darwinian survival-of-the-fittest principle and introduces a structure which permits efficient exchange of genetic information. The differences between the working of genetic algorithms and other optimization techniques will be discussed in next section.

In this chapter, the fundamental theory of simple genetic algorithms will first be discussed. It explains the inner workings of the algorithm and describes the mechanism of genetic operators. The fundamental theorem of genetic algorithms will then be stated into mathematical expression. A later section of the chapter will demonstrate the use of GA to solve a higher order polynomial optimization problem. Two numerical techniques are used to compare the results with the GA and the analytical solution. Finally the importance of determining appropriate GA parameters, which have a dominant influence on the effectiveness of the algorithm, is discussed.

3.2 Differences Between Genetic Algorithms & Other Optimization Techniques

The genetic algorithm differs from other search techniques due to its use of natural genetics and natural selection. First, it works with a coding of a parameter set, not the parameters themselves. Second, the genetic algorithm works with a population of strings.

It simultaneously evaluates many points in the parameter space and searches many peaks concurrently. By applying genetic operators, it exchanges information between the peaks, therefore reducing the possibility of converging at a local peak and missing the global peak. Third, the genetic algorithm only requires payoff information (evaluations of the objective function) to guide its search. It does not require derivatives or other auxiliary information. The fitness is the objective function that should be maximized or minimized. As analogy to the optimal control problem, the fitness is working like the cost function or the performance index. The only accessible information from the system is the evaluated fitness value of the current population. Finally, the genetic algorithm uses probabilistic transition rules rather than deterministic rules. The randomized search is guided by the comparison of each individual string fitness value. By employing the genetic operators, it explores different parts of the parameter space where there is a high probability of finding improved performance.

3.3 Fundamental Theory of Simple Genetic Algorithms

Genetic algorithms are known to have the flexibility and capability to find an optimal or near-optimal solution of a given objective function with respect to a set of variable parameters. GAs work with a population of a finite set of strings just like nature works with chromosomes. The variable parameters are coded as a fixed length string of characters or alleles which is called a chromosome. Each chromosome has an associated fitness which is the value of the objective function for that set of parameters. The fitness

must be a positive function which reaches a maximum for the optimal chromosome. A chromosome and its fitness define an individual. The chromosome will contain sub-strings or genes which as units, contribute in different ways to the overall fitness of the individual. Genetic algorithms proceed by taking a population of different individuals and creating a new population, or generation, by combining features of the chromosomes of individuals from the old population with the highest fitness.

The strings are made from a coding of a parameter set. A typical binary set of code is used as a sequence of ones and zeros. Each parameter has a finite length string of ℓ bits $[0,..., 2^{\ell}-1]$. The value of each substring is mapped onto an interval of real numbers $[x_{low}, x_{up}]$. With n parameters, the final string consists of n concatenated substrings. The string is decoded by a user-defined procedure which determines the parameter values for a given problem. For example, two 7-bit binary strings could be used to represent the two feedback control gains: K1 and K2. A single 7-bit string represents one of the $2^7 = 128$ alternative gain values. If the coding procedure assumed that the gain values are set in the range of $-5 \le K1 \le 5$, $-2 \le K2 \le 2$, then the following string:

1001110 0100110

K1 K2

would be decoded to the following values using equation 3.1:

$$\text{Decoded X} = \frac{\text{bit value of an individual string}}{\text{maximum bit value of a given string set}} * \left(x_{up} - x_{low}\right) + x_{low} \qquad (3.1)$$

$$K1 = (78/128)*(5-(-5)) + (-5) = 1.09375$$

$$K2 = (38/128)*(2-(-2)) + (-2) = -0.8125$$

A simple genetic algorithm is composed of three fundamental operators: reproduction, crossover, and mutation.

Reproduction: Reproduction is a process by which the strings with larger fitness values can reproduce with higher probabilities large numbers of their copies in the next generation. Selecting good strings for the reproduction operation can be implemented in many different ways. In the method used in this study, strings with higher fitness values, F, get a proportionally higher probability of reproduction selection (i.e. roulette wheel selection) based on

$$P_{(select)} = F_i / \sum F_i$$

where i = string index. This method, in which good strings get more copies in the next generation, emphasizes the survival of the fittest concept of genetic algorithms.

Crossover: Crossover is a process by which the systematic information exchange between two strings is implemented using probabilistic decisions. First, two newly reproduced strings are paired together at random. Then, an integer position n places along every pair of strings is selected uniformly at random. Finally, based on a probability of crossover, P_C, the paired strings submit to cross over at the integer position n places along the string. This results in new pairs of strings that are created by exchanging all of the characters between characters 1 and n inclusively. As an example, consider two strings A and B of length 6 mated at random from the mating pool of the new generation:

A = 1 1 1 0 ^ 0 0

B = 0 0 0 1 ^ 1 1

If the random number generator comes up with a cross site at 4, the bits are interchanged between string A and string B at the position marked after the symbol "^". Then the resulting crossover yields two new strings, A* and B* as follows:

A* = 1 1 1 0 ^ 1 1

B* = 0 0 0 1 ^ 0 0

Although crossover operation is a randomized event, when combined with reproduction it becomes an effective means of exchanging information and combining portions of good quality solutions.

Mutation: Mutation is simply a random alteration of bit values which is based on the probability of mutation, P_m. (The probability of mutation is usually small, say one in a thousand.) In a binary code, the mutation operator simply flips the bit value from a 0 to a 1 or vice versa. As an example, consider a string A=111100 is operated by mutation with P_m=1.0. After mutation is done, the string A becomes 000011. The mutation operator is used as an assurance against the loss of valuable information through the operation of reproduction and crossover.

A general scheme of implementing simple genetic algorithms is shown as below:

1. Initialize the population of strings (i.e. the 0th generation)
2. Decode and evaluate the fitness value of each string within the population
3. Select strings to reproduce based on their fitness

4. Select pairs of strings and exchange portions of the contents between them
5. Mutate occasional single bit values of a string
6. Create a new population set
7. Return to step 2

In explaining the inner workings of genetic algorithms, let us initially make a few definitions [16]. Since we are dealing with binary strings, a notation must be developed to denote similarity subsets (schemata). A schemata (denoted by the symbol "H") is a similarity subset which contains strings that have similarities at some bit positions. We can expand this thinking even further with the introduction of a "don't care" character, *, in addition to the binary set {0,1}. For example, H = 1**0 is a particular schema which matches all strings of length 4 with a 1 in the first position and a 0 in the fourth position. For instance, the strings 1110 and 1000 match this particular schema. The set {1110,1010,1100,1000} can be described by the similarity template 1**0. Using this notation, we can now define a schema's order and defining length. For a given schema (H), its defining length (denoted $\delta(H)$) is the distance between the first and the outermost defining positions of a schema. The order of a schema (denoted o(H)) is defined as the number of fixed bit positions within that schema. In the example given, $\delta(H) = 4-1 = 3$, while $o(H) = 2$.

The reproduction, crossover and mutation operators can be analyzed in terms of their affect on a schema. With the above definitions, we can now present the Fundamental Theorem of Genetic Algorithms, or the Schema Theorem [19, 16]:

$$m(H, t+1) \geq m(H,t) \cdot \frac{f(H)}{f_{avg}} \left[1 - p_c \frac{\delta(H)}{\ell - 1} - o(H) p_m \right] \qquad (3.2)$$

where:

$m(H,t)$ = number of copies of schema H, at generation t

$f(H)$ = fitness function of strings with schema H

f_{avg} = average fitness of population, at generation t

$\delta(H)$, $o(H)$ = defining length and order of the schema

Equation 3.2 gives a lower bound on the number of samples of schema H at generation t+1. If the product of the second and third factors on the right hand side of Equation 3.2 is greater than 1, then that particular schema will receive exponentially increasing trials in consequent generations. Equation 3.2 can be understood in terms of the genetic operators. If the schema fitness, $f(H)$, is better than average, then the schema is more likely to be reproduced. If the defining length, $\delta(H)$, is small, then the crossover operator is less likely to destroy the schema after bits have been exchanged between mated strings. If the order of the schema, $o(H)$, is small, mutation is less likely to destroy the schema by altering a specified bit position. Short, low order, highly fit schemata are referred to as building blocks since they help to guide the genetic algorithms toward optimal solutions by surviving from generation to generation.

3.4 A Sixth-Order Polynomial Function Problem

In this section, a multi-modal function is presented. The problem is to solve for its optimal solution. Genetic algorithms and some other numerical techniques are used to compare and illustrate their individual robust performance. An analytical calculus method is used as the true solution to compare all the numerical results. The Newton-Raphson method and the Golden Section Method are the two numerical techniques employed to make a comparison with the GA solution.

3.4.1 Analytical Calculus Method

A sixth-order polynomial function is given as below:

$$y(x) = x^6 + 4x^5 - 54x^4 - 160x^3 + 641x^2 + 828x + 5700 \qquad (3.3)$$

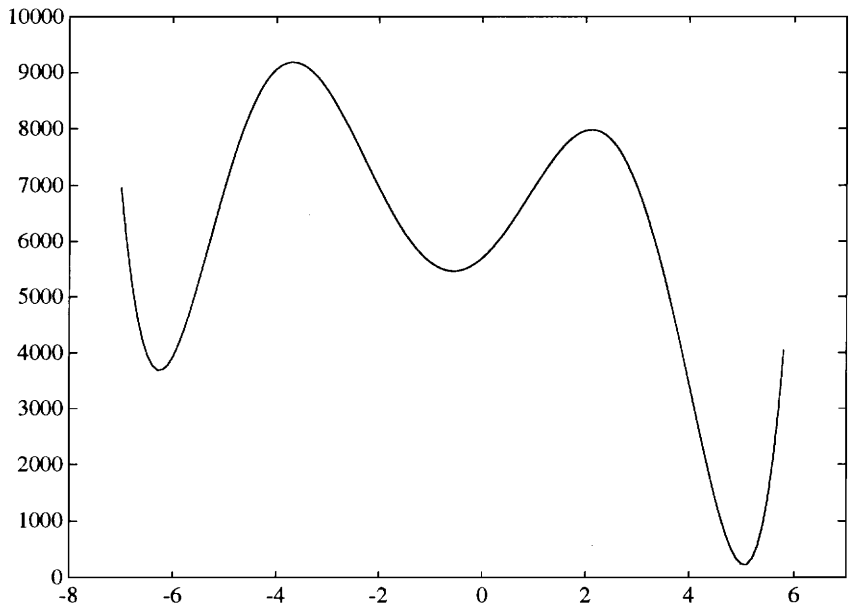

Figure 3.4.1: A Sixth-Order Polynomial Function

Finding the optimal point of this function is actually a process of finding the point at which the derivative of the function with respect to the independent variable (the slope of the curve) is zero. The first step is to take the first derivative of the function with respect to the variable, x, as below:

$$\frac{d}{dt}(y(x)) = 6x^5 + 20x^4 - 216x^3 - 480x^2 + 1282x + 828 \qquad (3.4)$$

Next, by setting equation 3.4 equal to zero, it is possible to calculate the roots of equation 3.3. The roots are found to be at the locations of all the turning points of the curve. MATLAB version 4.0 is used to solve the roots of the equation. In MATLAB, we set:

```
poly = [6 20 -216 -480 1282 828];
```

then, use the "roots" command to find the locations.

```
locate = roots(poly);
```

The location of the turning points are found to be:

x = {-6.2617, -3.6829, -0.5595, 2.1162, 5.0544}

The problem which comes to the mind is at which x-location is the optimal function value? The problem can be solved by applying the Second Derivative Test. It is stated [46] that; suppose a function f is differentiable on an open interval containing x and that $f'(x) = 0$.

(i) If $f''(x) < 0$, then f has a local maximum at x.

(ii) If $f''(x) > 0$, then f has a local minimum at x.

A point P(k, f(k)) on the graph of a function f is a point of inflection if f'' exists on an open interval (a,b) containing k, and f'' changes sign at k.

Taking the second derivative with respect to x from equation 3.3 yields:

$$\frac{d^2}{dt^2}(y(x)) = 30x^4 + 80x^3 - 648x^2 - 960x + 1282 \qquad (3.5)$$

Substituting the located turning point values into equation 3.5, gives the corresponding function values: $y'' = \{8364.8, -2448.8, 1605.2, -2291.7, 9784.7\}$. Obviously, based on the second derivative test, we can tell that only the second and the fourth x variables would give us local maxima. If the located turning point values are then substituted into equation 3.3, the resulting function values will give us a better idea of the location of the global maximum: $y = \{3686.2, 9188.0, 5459.9, 7983.1, 225.9\}$. Clearly, the global maximum of the function is located at $x^* = -3.6829$, with an optimal value, $y(x^*) = 9188.0$.

3.4.2 Numerical Techniques

Newton-Raphson Method

The Newton-Raphson method is employed to find the roots of a function equation [52]. An initial point (X_o) is first picked and substituted into the algorithm. After a number of function iterations, the new point (X_{j+1}) has converged into one of the roots of the equation. The root will be inserted into equation 3.4 to determine the optimal function value.

The algorithm is stated as follows:

$$X_{j+1} = X_j - \frac{f(X_j)}{f'(X_j)} \tag{3.6}$$

and its corresponding objective function comes from equation 3.4.

After 30 random trials, the result shows that choosing the initial point (X_o) is crucial for performing this technique. The root is found to be locally sensitive. The closer a particular root is to the initial point, the higher chance for that particular root to be picked as the solution. As a result, the root which is found may not represent the optimal value. After several trials, there is a safety range found to determine the optimal solution. If the initial point (X_o) is chosen from -5 to -2.65, the method is guaranteed to give the optimal solution within 6 iterations. Therefore, the Newton-Raphson method is concluded to lack robustness in solving a multi-modal function problem.

Golden Section Method

The golden section method is a popular technique for estimating the maximum, minimum, or zero of a one-variable function. It does not require function derivatives and its rate of convergence is well known. If we want to maximize the function as in our case, we need only minimize the negative of the function.

The algorithm begins by setting the lower and upper bounds on the x-direction. We will call them X_{low} and $X_{up.}$ Pick two intermediate points X_1 and X_2 such that $X_1 < X_2$ and evaluate the corresponding function values F_1 and F_2. Since we have assumed the function to be unimodal, it follows that either X_1 or X_2 will form a new bound on the

minimum. In this case, F_1 is greater than F_2, so X_1 forms a new lower bound and a new set of bounds, X_1 and X_{up} is found. In the other case, if F_2 is greater than F_1, X_2 would become a new upper bound to give X_{low} and X_2 as the points which bracket the minimum.

Assume we pick X_1 as the new lower bound and then we pick an additional point, X_3 and evaluate for F_3. Comparing F_2 and F_3, if F_3 is greater, then X_3 replaces X_{up} as the upper bound. The same process is repeated until the bounds converge within the desired tolerance. We only state the major equations to determine the two sets of points, X_1 and X_2 as follows. The detailed scheme of the algorithm is discussed in [52].

$$X_1 = (1 - \tau)X_{low} + \tau X_{up} \tag{3.6}$$

$$X_2 = \tau X_{low} + (1 - \tau)X_{up} \tag{3.7}$$

where $\tau = 0.38197$

Table 3.1: Results of performing golden section method

X_{low}	X_{up}	Optimal Value	# of iterations to converge for the optimal value
-7.0	5.8	9188.0	12
-6.8	5.8	7983.1	14
-6.8	5.7	9188.0	9
-6.5	5.7	7983.1	12
-6.5	5.4	9188.0	3
-6.4	5.4	7983.1	9

Table 3.1 shows that the optimal value is determined under a correct range set of bounded values. The ratio of X_1 to X_2 is called the famous "golden section" number [52].

It directly affects the direction of the search. For a multi-modal function, assigning the lower and the upper bound values is decisive. The algorithm is induced, based on the ratio of the golden section number. A wrong set of lower and upper bound values may lead the algorithm to search for the wrong location. As a result, the final solution may not match with the true solution.

3.4.3 Genetic Based Method

A simple genetic algorithm is used to find the optimal value of a sixth-order polynomial function. The algorithm is written in MATLAB. The population size of the parameter set is chosen to have 30 strings. Each string contains 7 bits and is coded over a given domain of the function, from -7.0 to 5.8. A single 7-bit string represents one of the $2^7 = 128$ alternative solutions. For example, 0000000 represents the minimum value of the parameter set and 1111111 represents the maximum value of the parameter set. If the strings are decoded using equation 3.1, they become -7.0 and 5.8, respectively. The objective function (fitness) to be maximized in this problem is equation 3.2. The probability of crossover, P_C is set to be 0.85; whereas the probability of mutation, P_m is set at 0.01.

A population of 30 different strings are first randomly initialized, to provide a diverse basis for the algorithm to start its search. This initial population corresponds with the 0th generation. Each string is decoded and evaluated for its associated fitness value. By using the available fitness information of the current population, three basic genetic

operators are then applied to produce a new population (next generation) of strings. This process is repeated until a specific number of generations is reached which is given by the user. After the iteration, the population should converge to some high fitness which is near the optimal region. The algorithm is run for 40 generations.

GA Results

Numerical results are shown in Fig. 3.4.2 & 3.4.3. In Fig. 3.4.2, the solid line shows the maximum fitness of each generation run by GA. The optimal value of the function is obtained in the 5th generation and 150 function evaluations are taken. The value remains steady through the end of iterations. The author would mention that in most cases, the optimal value may not remain as steady as shown in Fig. 3.4.2. The reason is based on the probabilistic characteristic of the genetic algorithm. However, as the number of generations increases, the maximum value of each generation should move increasingly towards and stay increasingly near the optimal region. Improvement will be discussed in the next chapter. Fig. 3.4.3 shows the average fitness of each generation run. As shown, there is an upward trend towards the region of maximum fitness as the number of generations increases. In both figures, the dotted line indicates the true solution generated by the analytical calculus method.

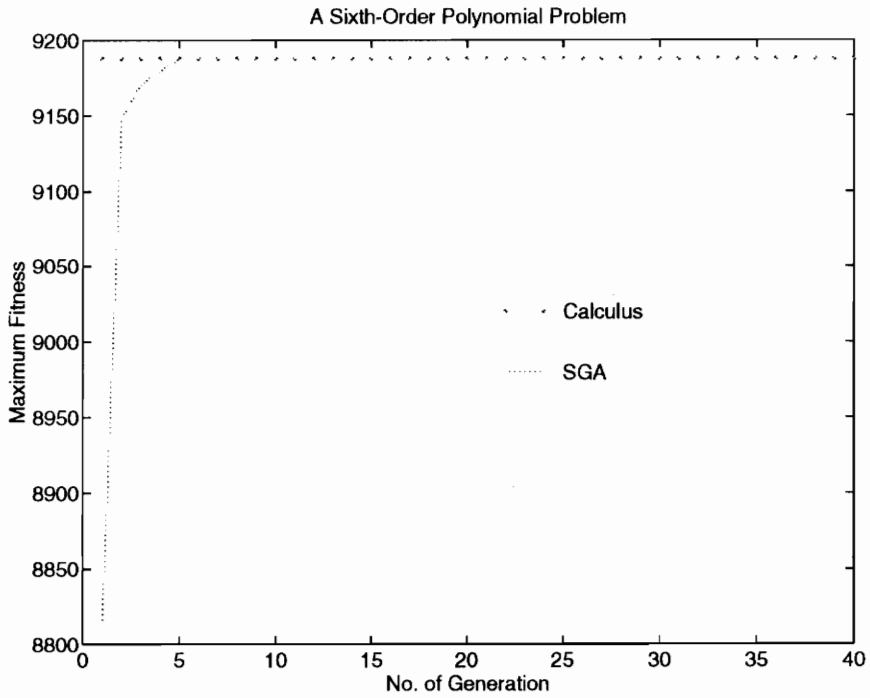

Figure 3.4.2: The Maximum Fitness Curve For the Sixth-Order Polynomial Problem

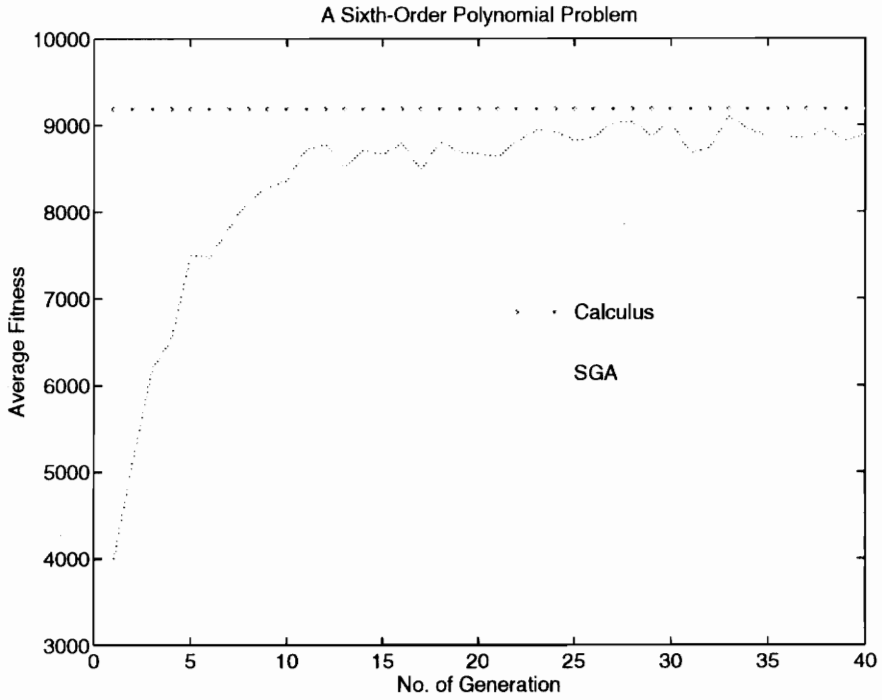

Figure 3.4.3: The Average Fitness Curve For the Sixth-Order Polynomial Problem

Results

Table 3.2: Comparison of different optimization techniques

	OPTIMAL VALUE	CONSTRAINT ON PERFORMANCE
Analytical Calculus	9188	Not Applicable
Newton-Rapshon	9188	Initial Point
Golden Section	9188	Range of the Parameter Set
Simple Genetic Algorithm	9187.7	None

Table 3.2 shows the results generated by the four chosen techniques. It is shown that all three programmable techniques are capable of obtaining an optimal result which is comparable to the true solution found from analytical calculus. However, from among the three techniques, the genetic algorithm is the only one which does not have any constraints on performance. The Newton-Raphson method strongly relies upon the initial point which is selected for performing the iteration. Since it is a local search technique, the algorithm looks for the peak closer to the initial point. Hence, if the initial point is not chosen properly, the global peak may always miss. The golden section method does not work any better than the Newton-Raphson method. Instead of choosing the right initial point, this time a proper range has to be defined well before the iteration starts. The golden section method works similar to the bisection method, but rather than bisecting in half each time, the algorithm searches upon the golden section number. The results show that the set range of the parameters determines the final optimal point in this multi-modal case.

The genetic algorithm works without a constraint. The result shows that it can perform up to 99.9967% accuracy. GA only requires the objective (fitness) function. GA is concluded to be more robust than the traditional numerical techniques.

There may be questions that someone would like to ask the author. How would the population size, the probability of crossover, and the probability of mutation of the genetic algorithm be chosen? What kind of influence do these parameters have on performance?

3.5 Parametric Selection of Genetic Operators

In 1975, De Jong [7] became the first person to start investigating the effect of different sizes of parametric selections of genetic algorithms. He performed several experiments and examined the influence of population size, crossover and mutation rates on the efficiency of the algorithm. In 1986, Grefenstette [17] did the similar work to solve a metalevel optimization problem. In this section, we will emphasize the importance of determining the appropriate size of each simple genetic operator. While one of the operators is being tested at different sizes, the size of the other two operators will be kept constant. Simulation results will be discussed to conclude the size of each genetic operator to be selected.

Size of Crossover Probability

The size of the population is set to have 20 strings. The mutation rate is chosen as P_m = 0.008. The size of the probability of crossover varies at 0.001, 0.01, 0.1 and 0.9. The simple genetic algorithm with four different crossover rates is run simultaneously for 40 generations. The results are shown in Figures 3.5.1-3.5.3.

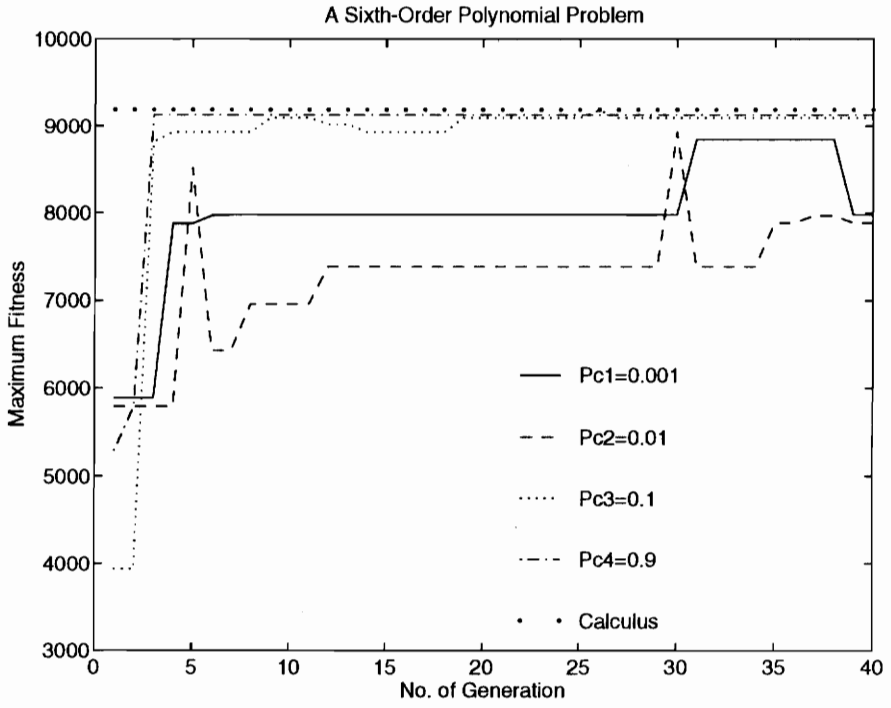

Figure 3.5.1: The maximum fitness at different crossover rates for the sixth-order polynomial problem

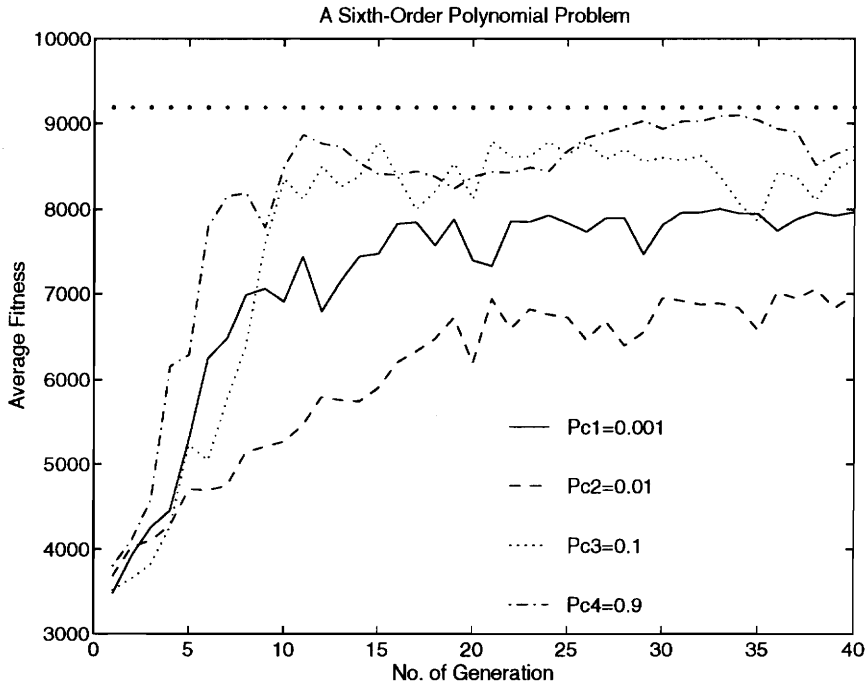

Figure 3.5.2: The average fitness at different crossover rates for the sixth-order polynomial problem

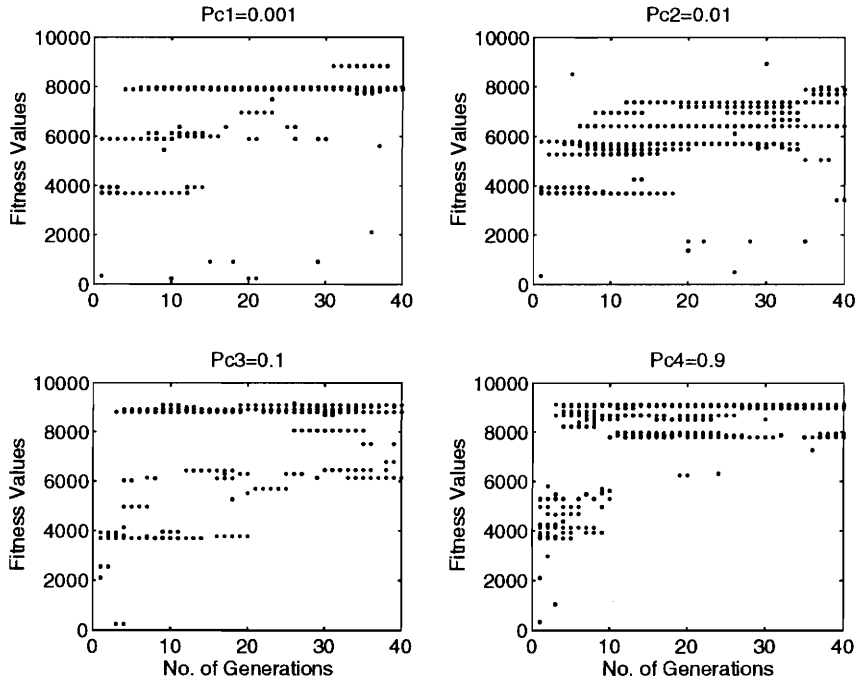

Figure 3.5.3: Population fitness of each generation at different crossover rates for the sixth-order polynomial problem

In Fig. 3.5.1, the maximum fitnesses of each generation are plotted for each individual crossover rate. They indicates that lower crossover rates tend to deteriorate the search and therefore it stops at an undesired region. On the other hand, higher crossover rates make the search tend towards the optimal solution. The entire population fitnesses increase with a higher crossover rate. The results are shown in both Fig. 3.5.2 and Fig. 3.5.3. However, if the crossover rate is chosen too high while the mutation rate is too low, the maximum fitness might stagnate at the wrong region. Thus, this causes the problem of premature convergence.

Size of Mutation Probability

The population size is still set to have 20 strings. The probability of crossover is chosen at $P_c = 0.7$. The mutation rates varies at 0.001, 0.01, 0.1,and 1. The algorithm is run for 40 generations with four different mutation probabilities. The results are shown in Fig. 3.5.4-3.5.6.

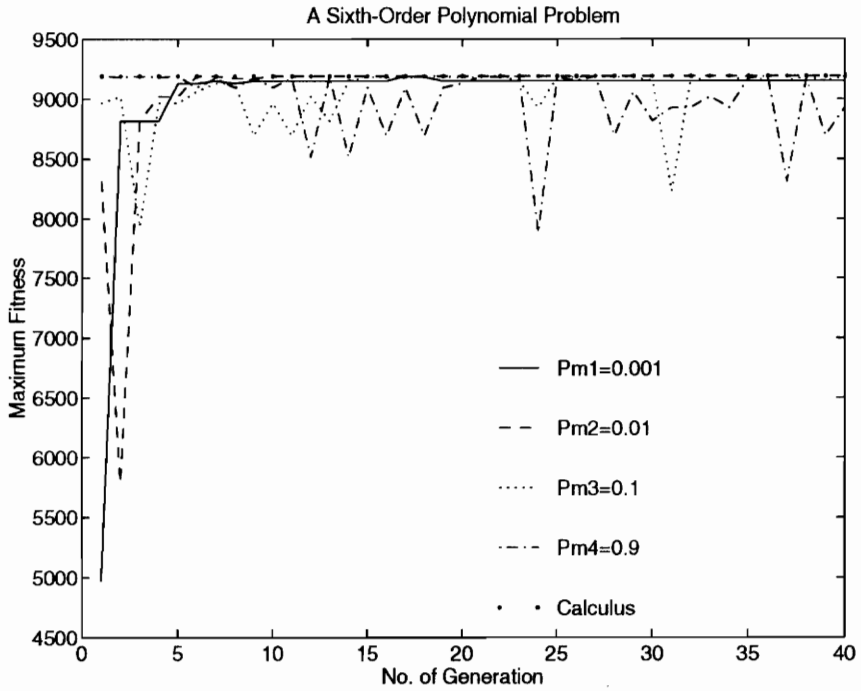

Figure 3.5.4: The maximum fitness at different mutation rates for the sixth-order polynomial problem

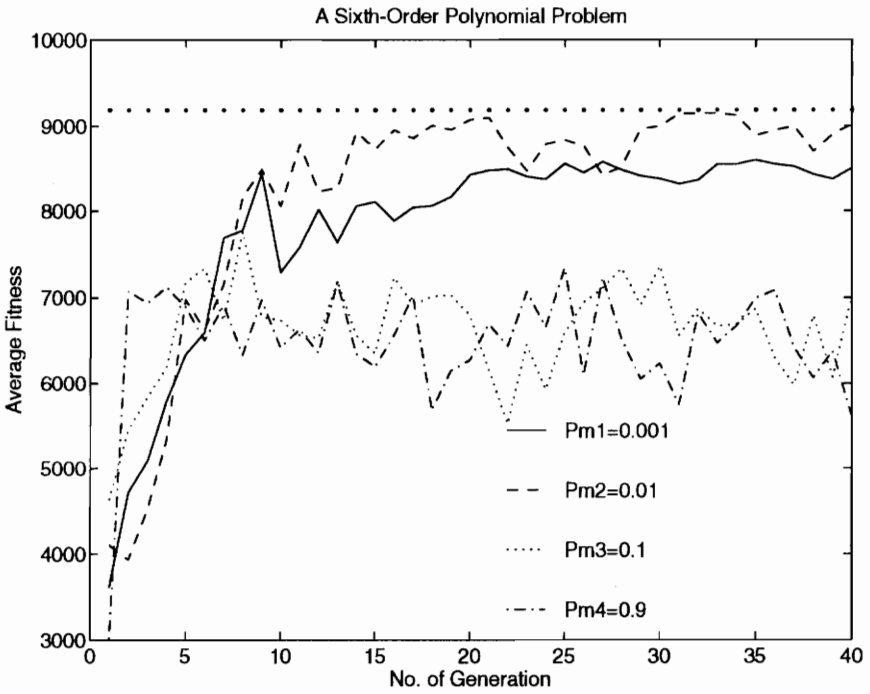

Figure 3.5.5: The average fitness at different mutation rates for the sixth-order polynomial problem

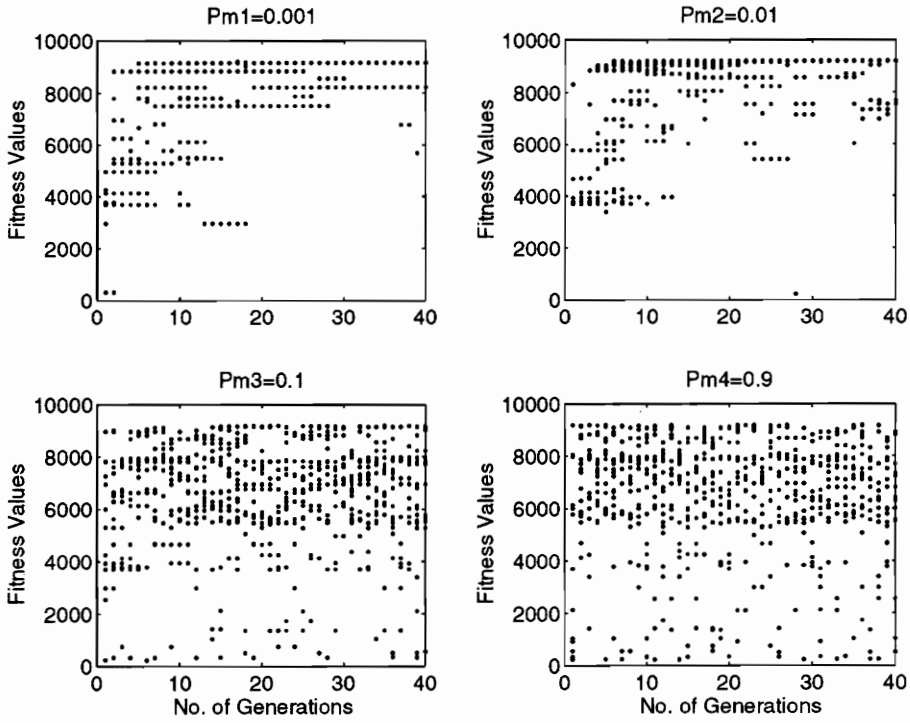

Figure 3.5.6: Population fitness of each generation at different mutation rates for the sixth-order polynomial problem

In Fig. 3.5.4, the result indicates that as the mutation rate increases, noise is added to the fitness function. If the mutation rate is increasing continuously towards one, the algorithm will act like as a random search. Figure 3.5.5 shows that the average fitness tends to wander around at the higher mutation rates. Fig. 3.5.6 indicates the noise effect generated by a high mutation rate.

Population Size

The probability of crossover and mutation are chosen as $P_c = 0.9$ and $P_m = 0.1$ respectively. The population size is varied from 10, 20, 30 and 40 number of strings per population. The algorithm is run for 40 generations with four different population sizes. The results are shown in Fig. 3.5.7-3.5.9.

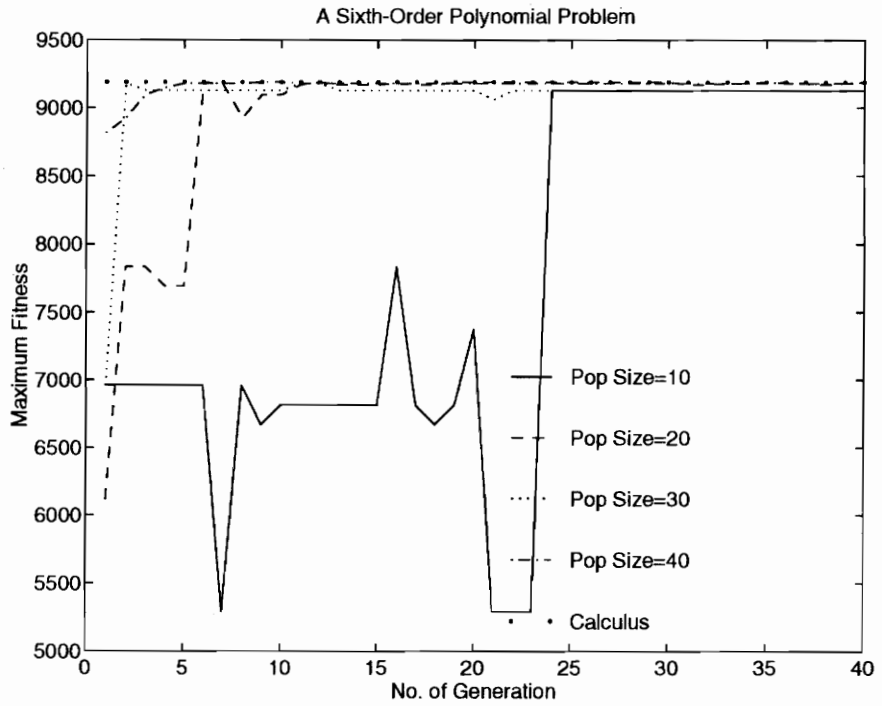

Figure 3.5.7: The maximum fitness at different population sizes for the sixth-order polynomial problem

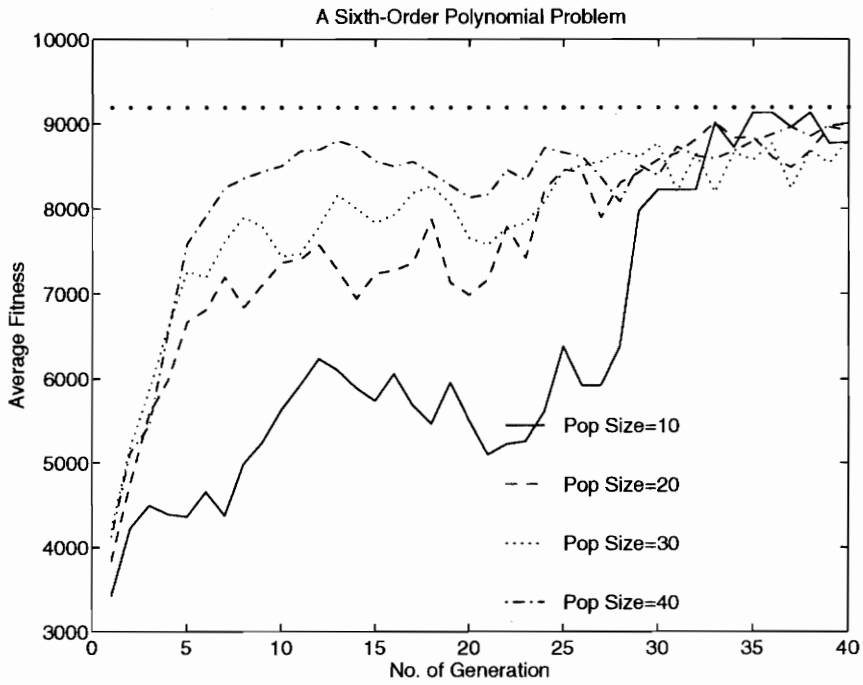

Figure 3.5.8: The average fitness at different population sizes for the sixth-order polynomial problem

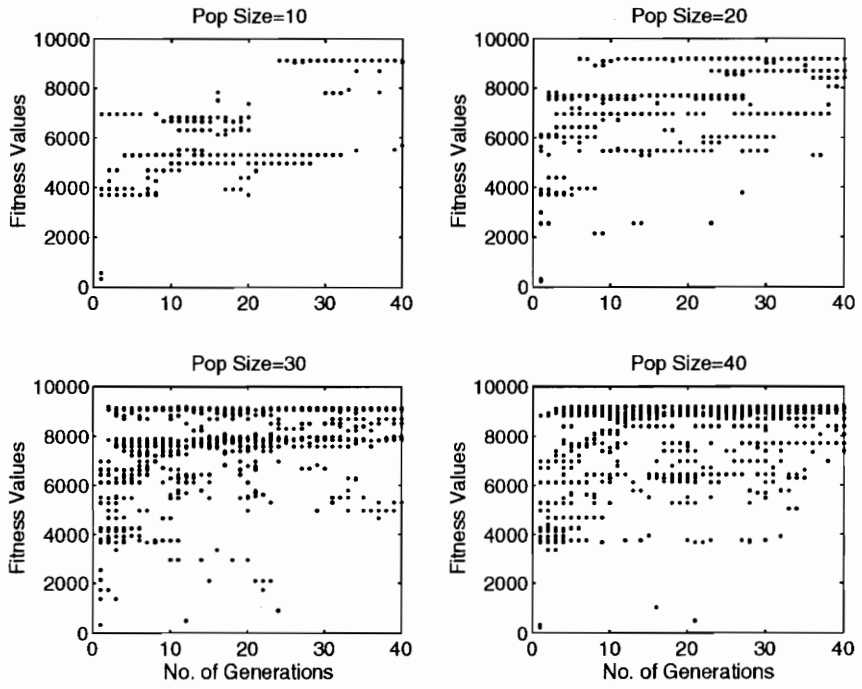

Figure 3.5.9: Population fitness of each generation at different population sizes for the sixth-order polynomial problem

In Fig. 3.5.7, the maximum fitnesses are plotted for four different population sizes. As the population size increases, there are fewer generations required for the population to reach its optimal region. In addition, the average fitness shown in Fig. 3.5.8 tends to reach the optimal region faster as the population size increases. In Fig. 3.5.9, as the population size increases, the entire population fitness are found at the upper optimal region.

In order to optimize the output performance of the algorithm for this problem, we conclude that the parametric size of each genetic operator is chosen, such that population size = 30, crossover rate = 0.85 and mutation rate = 0.01. As a remark, the parametric size of each genetic operator may vary from different problems.

Chapter 4

Numerical Simulation of GA-Based Feedback Control

4.1 Introduction

In conventional control theory, the analysis and design of control systems is carried out using transfer functions together with a variety of graphical techniques such as root-locus plots and Bode diagrams. If necessary, the dynamic behavior of a complex system may be improved by inserting a simple lead or lag compensator. The major drawback of classical control theory however, is that it is only applicable to linear time-invariant single-input-single-output systems. It does not work for multiple-input-multiple-output systems, time-varying systems, and non-linear systems (except simple ones). Therefore, classical control techniques are not commonly used for designing optimal and adaptive control systems.

Control systems designed by the classical technique are based on trial-and-error procedures that, in general, may not yield the optimal design. On the other hand, control systems designed by modern control theory via state-space methods allow one to design systems with desired closed-loop poles or to optimize the systems with respect to given performance indices. The Pole Placement technique and the Linear Quadratic Regulator (LQR) design method are the most common techniques used to solve control problems via state-space representation.

This section presents two feedback control system optimization problems using genetic algorithms: an initial condition free response spring-mass-damper problem and a loading bridge problem. The first one is modeled in a continuous-time system, whereas the second is in a discrete-time system. An GA-based feedback controller will be employed to solve the problems. The quadratic performance index will be chosen as the objective function to implement the genetic algorithm. An improved selection scheme and advanced genetic operators will be introduced.

4.2 Control Design Approach

In the Pole Placement technique, all state variables are assumed to be measurable and are available for feedback. If the system considered is completely state controllable, poles of the closed-loop system may be placed at any desired location by means of state feedback through an appropriate state feedback gain matrix. For a multiple-input-multiple-output system, the Pole Placement technique does not completely specify the control parameters. For example, a jth-order plant with k inputs and its entire state vector available for feedback would have $j.k$ controller parameters to be determined, but only j possible closed-loop pole locations. Hence, k times as many parameters as poles must be set. However, there are infinite ways by which the same closed-loop poles can be found. Which way is the best? What kind of method can be used to determine the appropriate feedback gains?

In addition, most of the time, the designer may not really know the desirable closed-loop pole locations. Choosing the right pole locations and picking the proper gain sets may be a challenging task for someone lacking extensive experience with control design. The optimal control theory was developed to overcome this problem and the LQR design method is chosen as the objective function to implement the genetic algorithms.

4.2.1 LQR Design Methodology

The LQR design calculates an optimal feedback gain set **K** such that the control law

$$u(k) = -\mathbf{K}\mathbf{x}(k) \tag{4.1}$$

minimizes the cost function or the quadratic performance index, J,

$$J = \sum_{k=0}^{N}\left[\mathbf{x}^{T}(k)\mathbf{Q}\mathbf{x}(k) + \mathbf{u}^{T}(k)\mathbf{R}\mathbf{u}(k)\right] \tag{4.2}$$

where **Q** and **R** are the weighting matrices. They are often called the state weighting matrix and control weighting matrix, respectively. In the performance index, the quadratic form $\mathbf{x}^{T}\mathbf{Q}\mathbf{x}$ represents a penalty on the deviation of the state **x** from the origin; whereas the term $\mathbf{u}^{T}\mathbf{R}\mathbf{u}$ represents the "cost of control." **Q** and **R** are square matrices with the size of **Q** being equal to the number of states and the size of **R** equal to the number of control inputs.

Note that the selection of the weighting matrices Q and R is, in a sense, arbitrary. Both Q and R must be positive-definite Hermitian or real symmetric matrices, and usually all the off diagonal elements are set to zero. The relative magnitude between Q and R is

an important factor when determining the LQR gain set. As Q gets higher, a higher control input u is required which will drive the states to zero faster; so the Q matrix penalizes states, and the controller has to work harder to reach the desired system response. R penalizes the control. As R increases, the control input u decreases and makes the states get larger, taking them longer to die out. Although minimizing an "arbitrary" quadratic performance index may not seem to have much significance, the advantage of the quadratic optimal control approach is that the resulting system is a stable system.

4.3 A SDOF Spring-Mass-Damper Model

A single-degree-of-freedom spring-mass-damper model is given with an initial condition $x(0) = 1$ and $\dot{x}(0) = 0$. There is no forcing function on the system so that the system will vibrate freely and die out with time. An adaptive GA-based feedback controller is designed to control the system with the given initial condition. The system model schematic is shown in Fig. 4.3.1.

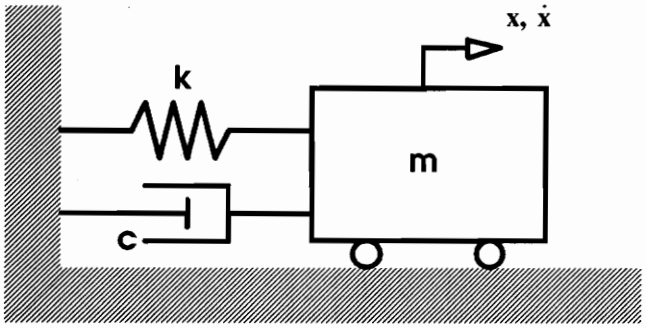

Figure 4.3.1: A Single-Degree-Of-Freedom Spring-Mass-Damper Model

We apply Newton's second law to derive the differential equations of motion of the system:

$$\sum F = ma = -kx - c\dot{x} = 0$$

which yields

$$m\ddot{x} + c\dot{x} + kx = 0 \tag{4.3}$$

We can convert equation 4.3 into standard state-space form

$$\begin{aligned} \dot{\mathbf{x}} &= [\mathbf{A}]\mathbf{x} + [\mathbf{B}]\mathbf{u} \\ \mathbf{y} &= [\mathbf{C}]\mathbf{x} + [\mathbf{D}]\mathbf{u} \end{aligned} \tag{4.4}$$

or for our problem

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -k/m & -c/m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u \\ y &= [1 \quad 0]\, x \end{aligned} \tag{4.5}$$

where m=1, c=1, and k=1.

Equation 4.5 is the dynamic characteristic of the system model. Combining the system dynamics and the initial condition, MATLAB version 4.0 is used to simulate the initial condition response of the system. The result is shown in Fig. 4.3.2.

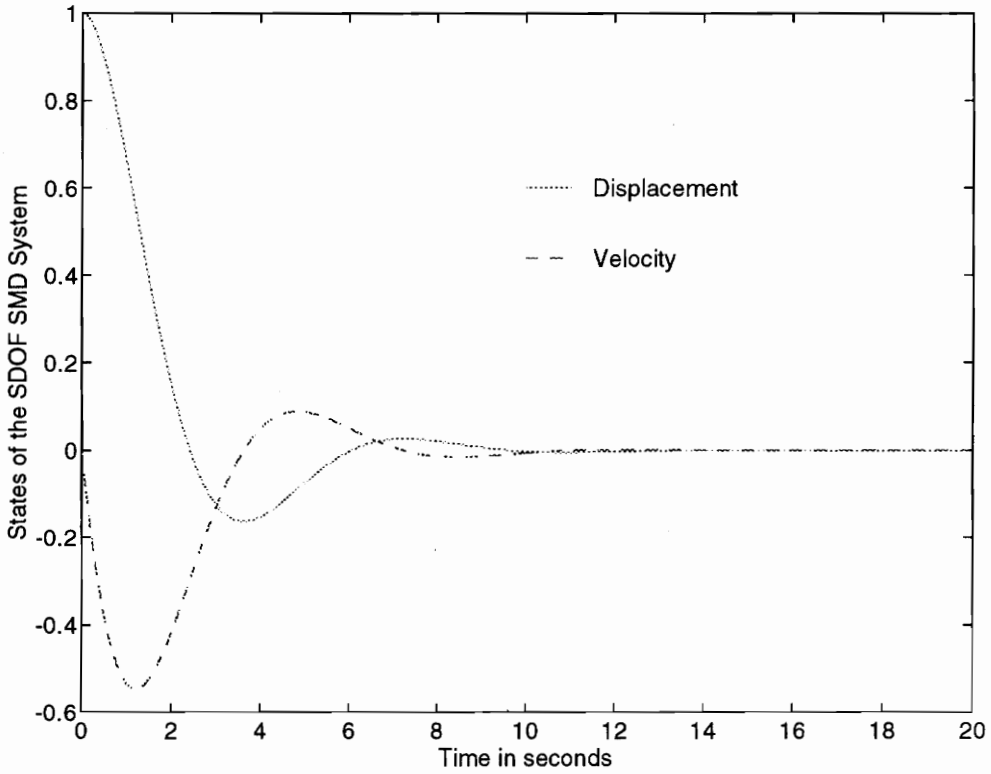

Figure 4.3.2: Free Response of a SDOF SMD System with an initial condition

There is no control applied to the system shown as above.

4.3.1 Improved GA Design

An adaptive GA-based controller is designed to perform feedback control on the system. Two 9-bit binary strings are used to represent the two feedback control gains: K1 and K2. The strings are concatenated to produce an 18-bit string, each string representing one feedback design. The gain values are set in the range of $-2 \leq K1 \leq 2$, $-2 \leq K2 \leq 2$. The

range of the gains is chosen based on the LQR gain set obtained from MATLAB. The two state feedback gain set is designed as:

$$u = -[K1 \quad K2]\begin{bmatrix} x \\ \dot{x} \end{bmatrix} \tag{4.6}$$

where K1 and K2 are the linear feedback gain values for the state of displacement and velocity, respectively. For example, an 18-bit string, divided into 9-bit K1 and K2 for clarity, is shown below:

110110011 000110010

K1 K2

This string represents one possible solution out of 2^{18} solutions. In the previous representation, 000000000 represents a predetermined minimum value for the gains and 111111111 represents a predetermined maximum value for the gains. Note that the mapping of the gain set onto the 9-bit string can be distinct from different gains depending on the desired range and the resolution of the gain values. For a 9-bit representation, the resolution is calculated to be $2/(2^9-1) = 0.00391$. For the above example, the two 9-bit substrings would be decoded to the following values:

$$K1 = (435/512)*(2-(-2))+(-2) = 1.3984$$

$$K2 = (50/512)*(2-(-2))+(-2) = -1.6094$$

The quadratic performance index from equation 4.2 is used as the objective function for implementing the genetic algorithm. The weighting matrices are chosen as:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad R = [1]$$

The genetic algorithm used here requires a mapping of the objective function value to a fitness value. The fitness function is given by:

$$\text{fitness} = \frac{1}{J} \tag{4.7}$$

The population size used in the simulation runs is 30, the crossover operator uses a probability of 0.85, and the mutation rate is 0.01. The two basic genetic operators, crossover and mutation, are still provided to run the algorithm. Instead of using the reproduction operator or the roulette wheel selection method however, an improved selection technique is introduced which is called the stochastic remainder selection without replacement.

Stochastic Remainder Selection Without Replacement

In this method, described by Booker [16], the choice of selecting the next generation of strings is based on two elements: the integer part of the expected number and the fractional portion of the remainder. The expected number of offspring for each individual, expect_j is computed as:

$$\text{expect}_j = \frac{f_j}{\bar{f}} \tag{4.8}$$

where f_j is the fitness value for a particular element of the population;

$\bar{f}$ is the mean value of the whole population fitness.

Once the expected number of each string is calculated, the integer part of each expected number will first be evaluated to select for the next generation. The remains of the pool are filled under weighted probabilites in proportion to the value of each fractional part of the expected numbers.

To illustrate this phenomenon, suppose a population fitness set is defined as: a = [15; 5; 10; 20; 25] and the total sum of fitness, suma = 75. The mean value of the population fitness, $\bar{f}$ = 75/5 =15. The equation 4.8 is applied to compute the expected number for each individual. The result is shown as:

$$a_{expect} = \begin{bmatrix} 1.00 \\ 0.33 \\ 0.67 \\ 1.33 \\ 1.67 \end{bmatrix}$$

The integer part of each individual will first be evaluated. Obviously, the first, the fourth, and the fifth elements will be selected because they contain the value of one in their integer portion. The two remaining choices will then be picked under the weighted probabilities (roulette wheel selection) of the fractional part for each individual. After the random iteration, the third and the fourth elements will be picked. The selection of the next generation is rearranged and set as:

$$a_{new} = \begin{bmatrix} 15 \\ 20 \\ 25 \\ 10 \\ 20 \end{bmatrix}$$

The operational procedure of conducting GA is outlined next.

1) Initialize the 0th generation of population strings randomly (popsize=30)
2) Decode each individual string to determine each state feedback gain set
3) Perform a time simulation of the system response to the given model and evaluate the fitness function
4) Repeat steps 2-3 for all 30 strings in the population
5) Apply the stochastic remainder without replacement selection method and the genetic operators to generate a new set of population of 30 strings
6) Return to step 2

This operational procedure will stop after a fixed number of generations which is given by the user. In this case, the author has picked 60 generations for the algorithm to run. Meanwhile SGA is also run for the same number of generations as a comparison to the improved GA (IGA).

4.3.2 Improved GA Results

Figure 4.3.3 shows the maximum fitness curve for each of the 60 generations. For the SDOF SMD system, both SGA and IGA reach a maximum fitness value of maxfit = 0.06957. The LQR optimal solution obtained from MATLAB is comparable at 0.06956. Iteration result, Fig. 4.3.3, shows that the IGA reaches the maximum value about 10

generations earlier than SGA. This indicates the improved robustness of IGA over SGA. The weakness of using SGA will be clear when optimization of a more complicated system model is addressed. Figure 4.3.4 shows the average fitness curve for each of the 60 generations. At the beginning of the run the average fitness of the population by both IGA and SGA is fairly low. As the number of generations increases, the average fitnesses approach the maximum fitness value. Due to the probabilistic nature of the GA, a certain amount of "genetic noise" is expected due to the crossover and mutation operators. The genetic noise increases as the number of generations pass.

Figure 4.3.5 shows the two state feedback gain values of K1 and K2 corresponding to the maximum string fitness of each generation which is run by the IGA. The IGA feedback gain values approach the LQR gain values after the 10th generation. Figure 4.3.6 shows a 3-D mesh plot of the number of fitness versus the fitness values from each generation. Finally, figure 4.3.7 and figure 4.3.8 show the feedback time response curve of the SDOF SMD model. Clearly, the IGA design obtains an optimal feedback controller which can drive the system states to zero faster.

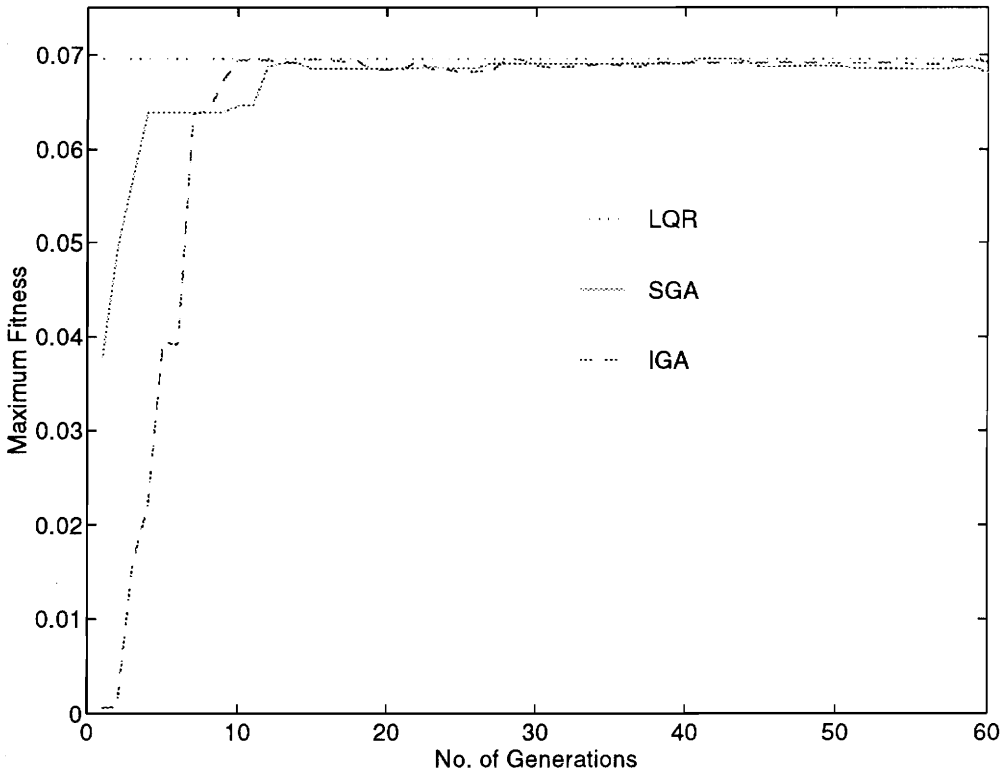

Figure 4.3.3: Maximum Fitness Curve for the SDOF SMD System

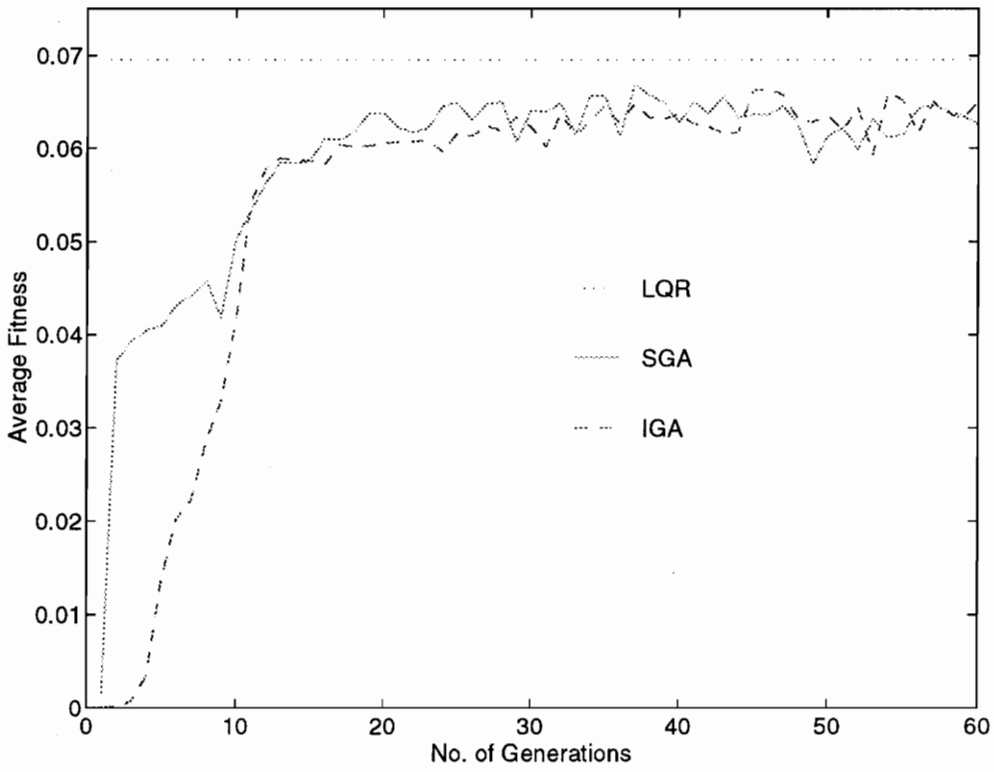

Figure 4.3.4: Average Fitness Curve for the SDOF SMD System

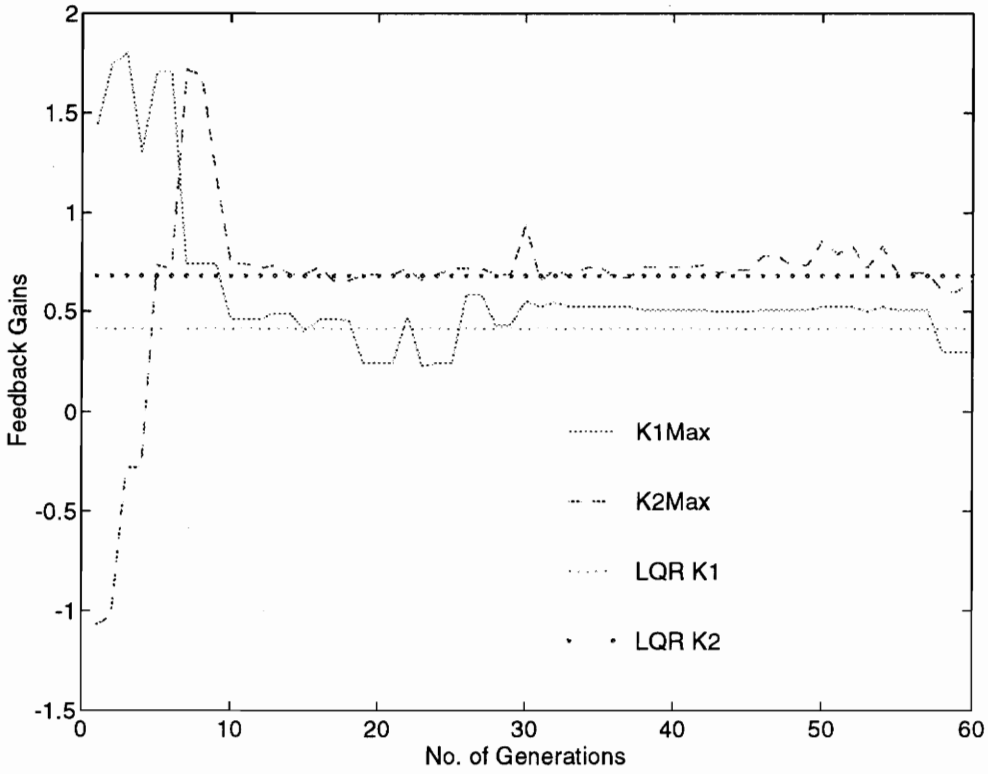

Figure 4.3.5: Full-State Feedback Gains for the SDOF SMD System

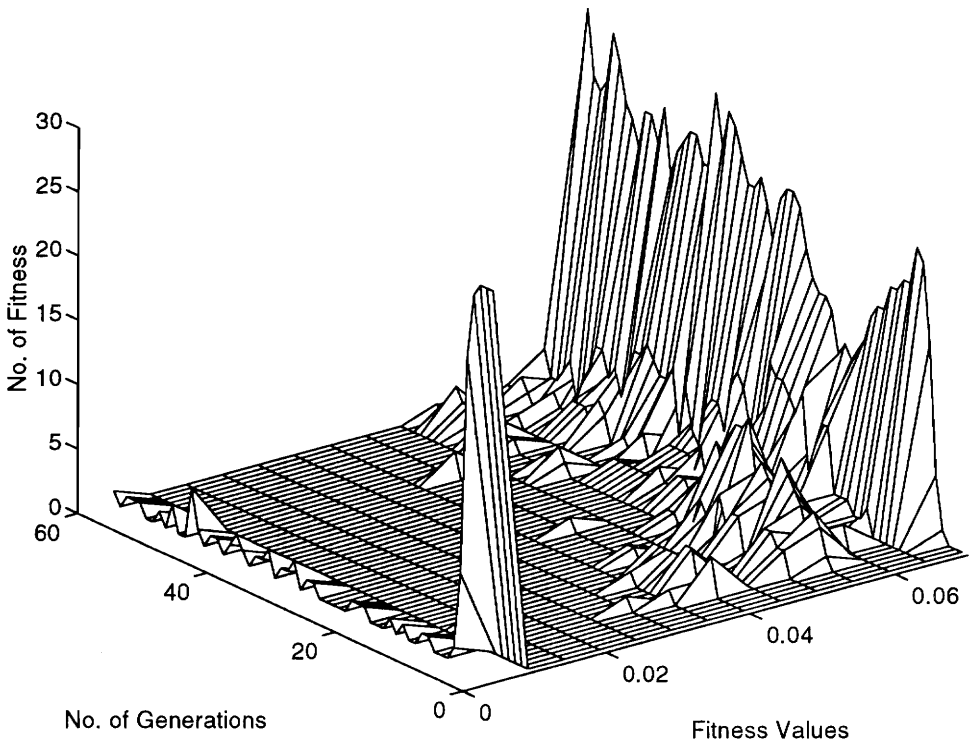

Figure 4.3.6: 3-D Mesh Plot of the Population Fitness vs. No. of Generations for the SDOF SMD System

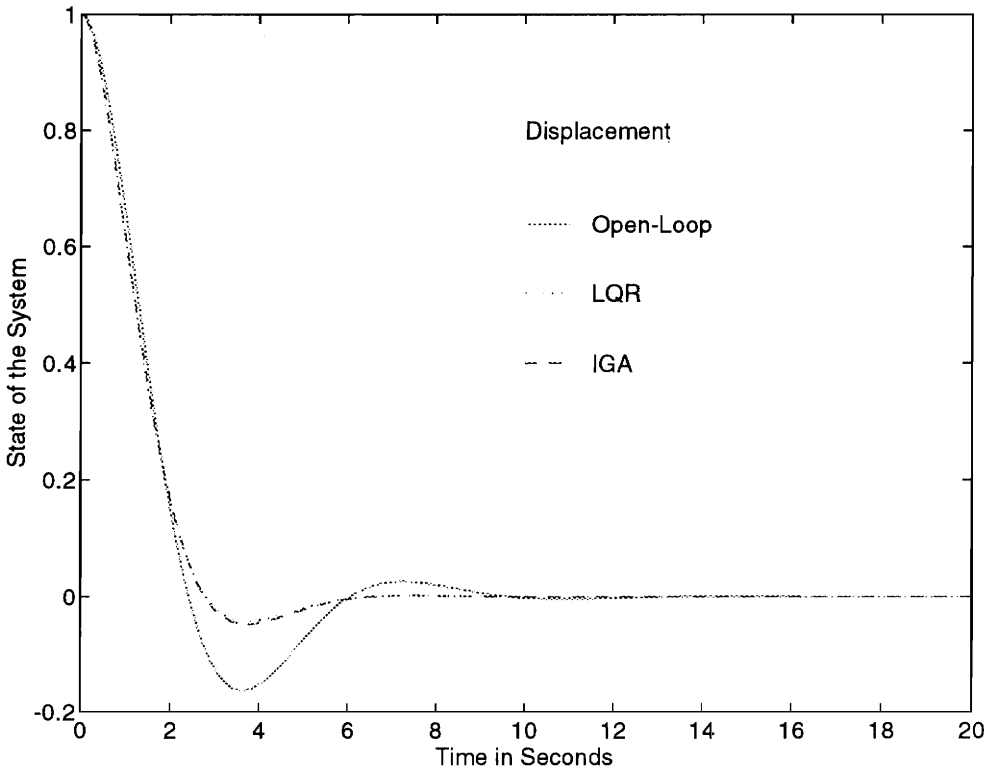

Figure 4.3.7: Two-State Feedback Response of a SODF SMD System

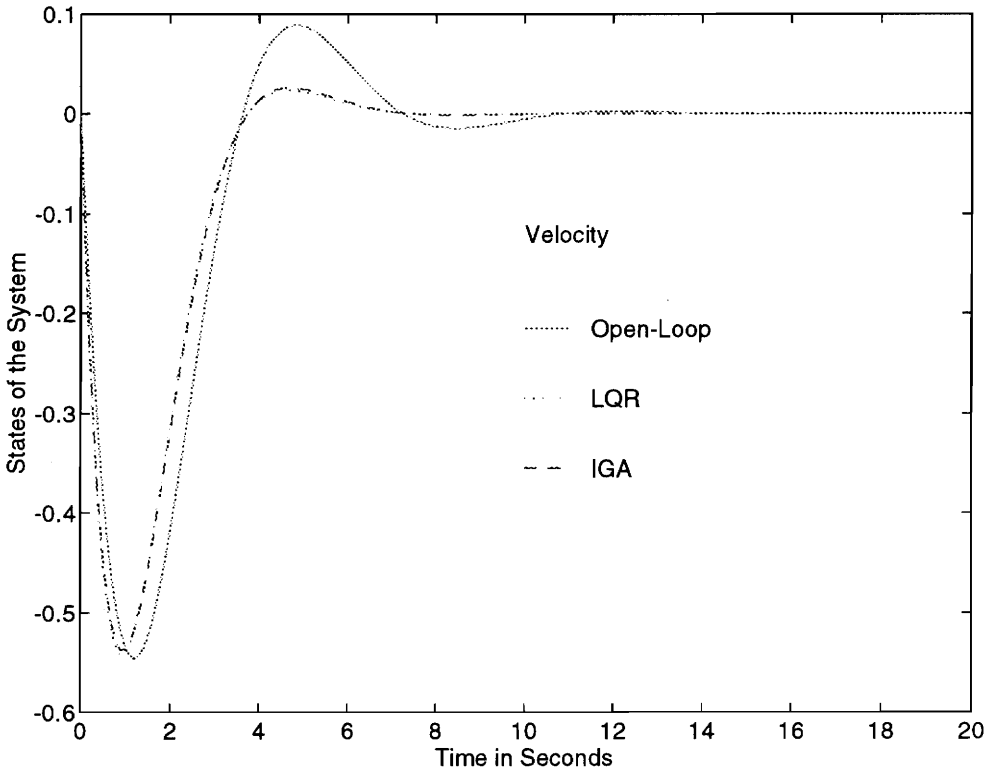

Figure 4.3.8: Two-State Feedback Response of a SDOF SMD System

4.4 A Loading Bridge

In this section, a regulator control problem is presented using advanced genetic algorithms. A loading bridge is used as the dynamic control model. The task is to move the wheeled cart, with a rigid pole plus a load hinged at the bottom of the pole, along a bounded straight track by allowing the cart position to be specified from a given reference input voltage. An advanced GA-based controller is designed to accomplish two major tasks. The first is to accelerate and decelerate the total mass of the system to achieve the required displacement. The second is to reduce any residual vibrating effect caused by the motion of the system. A full state feedback gain set is designed to maintain positioning and load angle with vertical of the system. The quadratic performance index is used again as the GA objective function. The inversion operator and the two-point crossover operator will be introduced in this section.

4.4.1 Mathematical Modeling

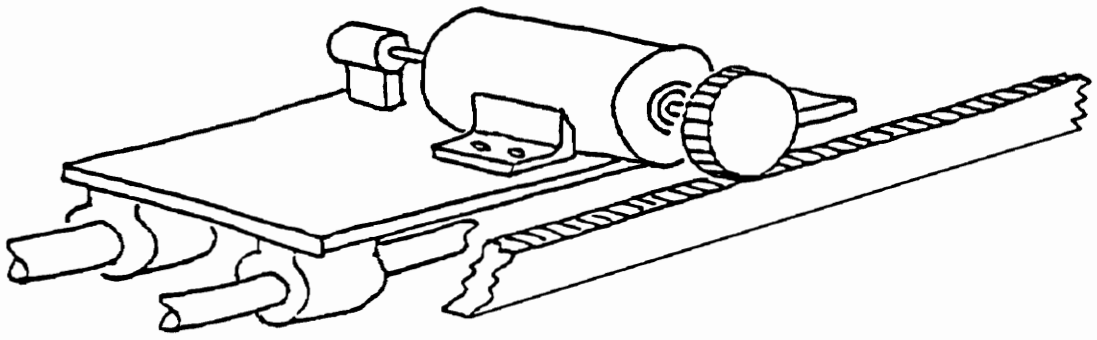

Figure 4.4.1: A drawing of a loading bridge without an attached load

The entire system consists of a D.C. permanent magnet motor, a cart, and a rack and pinion. A potentiometer on the motor shaft is used to measure the angular movement and the tachometer is used to measure the angular velocity of the system. A schematic of a dynamic control model for the loading bridge hinged with a load is shown in Fig. 4.4.2

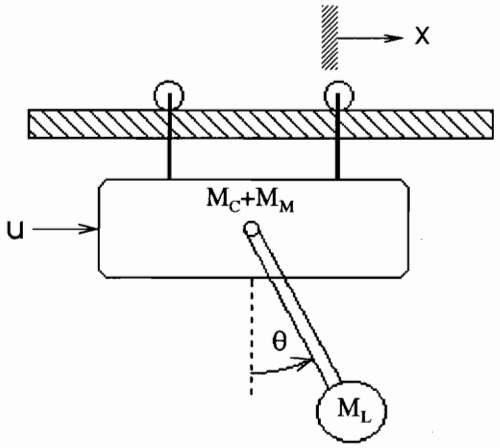

Figure 4.4.2: A dynamic control model for the loading bridge hinged with a load

(Remark: The rigid pole is assumed massless in this problem.)

The state of the loading bridge system at any time t is specified by four variables:

x = position of the cart on the track
$\dot{x}$ = velocity of the cart
θ = angle of the loading pole with the vertical
$\dot{\theta}$ = angular velocity of the loading pole

where $M_C + M_M$ is the total mass of the cart and the motor, M_L is the mass of the load, and u is the control applied to the system.

The system has only two degrees of freedom and its dynamics can be expressed in terms of generalized coordinates by using the Lagrange's equation:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = Q_i \qquad i = 1,2,\ldots, n \tag{4.9}$$

where q_i = generalized coordinates (linear or angular displacements)

T = total kinetic energy of a system

U = change in the potential energy of a system with respect to its potential energy in the static-equilibrium position

Q_i = generalized nonpotential forces or moments (nonconservative forces or moments) that are external to the system

The kinetic energy, the potential energy, and the generalized force acting on the cart are calculated to be:

$$\text{K. E. : } T = \frac{1}{2}M_C\dot{x}^2 + \frac{1}{2}\frac{J_o}{r^2}\dot{x}^2 + \frac{1}{2}M_L\left[\dot{x}^2 + 2\dot{x}\ell\dot{\theta}\cos\theta + \ell^2\dot{\theta}^2\right]$$

$$\text{P. E. : } V = M_L g[\ell - \ell\cos\theta] \tag{4.10}$$

$$Q_x \text{ : } F = \frac{K_T}{r}\frac{1}{R_a}\left[V_a - \frac{K_b\dot{x}}{r}\right]$$

where

M_C = .828 oz-sec^2/in = mass of the cart and the motor

M_L = .478 oz-sec^2/in = mass of the load

J_o = .037 oz-sec^2-in = rotary inertia of the motor

ℓ = 32 in = length from the centroid of the cart to the centriod of the load

K_b = (2.2)(.309) volts/(rad/sec) = motor back emf constant

K_T = 43.8 oz-in/amp = motor torque constant

r = 1.0 in = radius of the motor

R_a = 1.5 ohms = motor armature resistance

g = 386.4 in/sec^2

Using the Lagrangian function, L = T-V, and applying Lagrange's equation from equation 4.9 yields the nonlinear differential equations of motion as:

$$\left[M_C + M_L + \frac{J_o}{r^2}\right]\ddot{x} + M_L\ell\cos\theta\ddot{\theta} - M_L\ell\dot{\theta}^2\sin\theta = \frac{K_T}{r}\frac{1}{R_a}\left[V_a - \frac{K_b\dot{x}}{r}\right]$$
$$M_L\ell\cos\theta\ddot{x} + M_L\ell^2\ddot{\theta} + M_Lg\ell\sin\theta = 0 \qquad (4.11)$$

The equations are nonlinear due to the presence of the trigonometric terms $\sin\theta$ and $\cos\theta$ and the quadratic term $\dot{\theta}^2$. The equations are linearized by assuming a small angle of motion: $\sin\theta \approx \theta$, $\cos\theta \approx 1$, and $\dot{\theta}^2 \approx 0$. Under these approximations, we obtain the linearized dynamic model:

$$\left[M_C + M_L + \frac{J_o}{r^2}\right]\ddot{x} + M_L\ell\ddot{\theta} = \frac{K_T}{R_a r}V_a - \left[\frac{K_T K_b}{R_a r^2}\right]\dot{x}$$
$$M_L\ell\ddot{x} + M_L\ell^2\ddot{\theta} + M_Lg\ell\theta = 0 \qquad (4.12)$$

The linearized equations of motion for the loading bridge can be represented in matrix form as:

$$\begin{bmatrix} M_C + M_L + \frac{J_o}{r^2} & M_L\ell \\ M_L\ell & M_L\ell^2 \end{bmatrix}\begin{Bmatrix} \ddot{x} \\ \ddot{\theta} \end{Bmatrix} + \begin{bmatrix} \frac{K_T K_b}{R_a r^2} & 0 \\ 0 & 0 \end{bmatrix}\begin{Bmatrix} \dot{x} \\ \dot{\theta} \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & M_L g\ell \end{bmatrix}\begin{Bmatrix} x \\ \theta \end{Bmatrix} = \begin{bmatrix} \frac{K_T}{R_a r} \\ 0 \end{bmatrix} V_a$$

or (4.13)

$$[M] \quad \{\ddot{x}\} + \quad [C] \quad \{\dot{x}\} + \quad [K] \quad \{x\} = \begin{bmatrix} \frac{K_T}{R_a r} \\ 0 \end{bmatrix}$$

A state-space representation corresponding to equation 4.13 is obtained as:

$$\begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ [-M^{-1}K] & & [-M^{-1}C] & \end{bmatrix}\begin{Bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \\ M^{-1}\begin{bmatrix} \frac{K_T}{R_a r} \\ 0 \end{bmatrix} \end{bmatrix} V_a \qquad (4.14)$$

Equation 4.14 is in the same form as equation 4.4.

The numerical values of the system matrices are calculated as:

$$M = \begin{bmatrix} 1.343 & 15.296 \\ 15.296 & 489.472 \end{bmatrix};\ C = \begin{bmatrix} 19.85016 & 0 \\ 0 & 0 \end{bmatrix};\ K = \begin{bmatrix} 0 & 0 \\ 0 & 5910.3744 \end{bmatrix};\ F = \begin{bmatrix} 29.2 \\ 0 \end{bmatrix}$$

The dynamic characteristic of the loading bridge model under the continuous system is defined:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ [-M^{-1}K] & & [-M^{-1}C] & \end{bmatrix};\ B = \begin{bmatrix} 0 \\ 0 \\ [M^{-1}F] \end{bmatrix};\ C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix};\ D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

The control model is then discretized because digital control is to be implemented. This is done by assuming a zero-order hold on the system outputs and a sampling time of T, using

$$\Phi = e^{AT} \tag{4.15}$$

$$\Gamma = \int_0^T e^{A\eta}\, d\eta\, B \tag{4.16}$$

which gives the difference equations in the standard form:

$$x(k+1) = \Phi x(k) + \Gamma u(k) \tag{4.17}$$

$$y(k) = Hx(k) \tag{4.18}$$

where Φ, Γ and $\mathbf{H}$ are the system matrices of the discretized control system; $\mathbf{x}(k)$ refers to the state matrix operating at each time step; and $\mathbf{u}(k)$ is the control signal at each sampling interval.

The sampling time is given as ts = 1/40.016006 and the given reference input voltage, ref_{volt} = [3 0 0 0]. The equation of the reference input for the full-state feedback design is given as:

$$\mathbf{u}(k+1) = -[\mathbf{K}]\left[\mathbf{x}(k) - \mathbf{x}_r\right] \tag{4.19}$$

X_r is the state reference and X(k) is the state matrix of the system generating at each time step. K and u are the feedback gain set and the control input, respectively.

An open-loop time response of the loading bridge model is shown in Fig. 4.4.3. An adaptive advanced GA-based feedback controller is designed to control the loading bridge with the given reference input voltage.

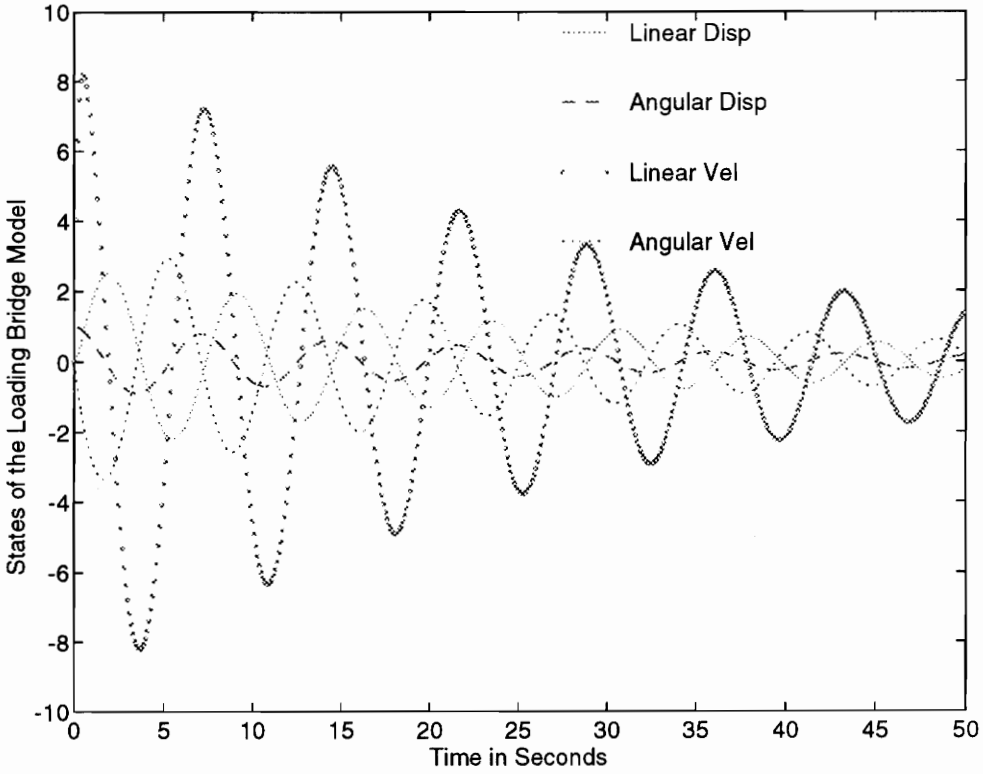

Figure 4.4.3: Open-Loop Response of the Loading Bridge

4.4.2 Advanced GA Design

An adaptive full-state feedback, GA-based controller is designed to control the system. The full-state feedback system is defined as:

$$u = -\begin{bmatrix} K1 & K2 & K3 & K4 \end{bmatrix} \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} \tag{4.20}$$

Each gain in the gain set (Kj) is represented by a 12-bit string. The strings are concatenated to produce a 48-bit string and each represents one feedback design. For example, a 48-bit string, divided into 12-bit K1, K2, K3 and K4 for clarity , is shown:

001111110010 111101110011 101010011010 001011010110

K1 K2 K3 K4

This string represents one possible solution out of 2^{48} solutions. In the previous representation, 000000000000 represents a predetermined minimum value for the gains, and 111111111111 represents a predetermined maximum value for the gains, with this 12-bit binary string mapped to the range of each gain. In this problem, the minimum and the maximum values of K3 and K4 are set at -1 and +3, respectively; whereas $K2_{min}$ and $K2_{max}$ are set at -25 and -15, respectively and the K1 value is set at the range from -2 to +2. For a 12-bit representation, the resolution is calculated to be $2/(2^{12}-1) = 0.000488$. For the above example, the four 12-bit substrings would be decoded to the following values:

K1 = (1010/4096)*(2-(-2))+(-2) = -1.0137

K2 = (3955/4096)*((-22)-(-30))+(-30) = -22.2754

K3 = (2714/4096)*(3-(-1))+(-1) = 1.6504

K4= (726/4096)*(3-(-1))+(-1) = -0.2910

The quadratic performance index from equation 4.2 is used again as the objective function to implement the genetic algorithm. The weighting matrices are chosen as:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2000 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad ; \qquad R = [1]$$

The selection of the weighting matrices was discussed in section 4.2.1. In this problem, we have purposely chosen the second diagonal element of the Q matrix as 2000 because a very large control input is required to drive that particular state to zero. As the problem stated, the control model is designed to accelerate and decelerate the entire system to achieve the required displacement and at the same time, to reduce any residual vibrating effect caused by the motion of the system. Therefore, controling the angle of the loading pole with the vertical to be as small as possible is an important factor in this problem. Hence, the second feedback control gain is designed under a range set from -22 to -30.

The genetic algorithm used here requires a mapping of the objective function value to a fitness value. The same fitness function comes from equation 4.7:

$$\text{fitness} = \frac{1}{J}$$

An advanced genetic algorithm is implemented for this problem. A two-point crossover operator and an inversion operator are introduced to solve the problem. The mutation operator is still used as an assurance against the loss of valuable information through the iterating process. The stochastic remainder selection without replacement technique is used for selecting the pool of strings for each generation.

Two-Point Crossover

In the two-point crossover operation, two strings are paired together at random. Then, two integer positions place along each pair of strings are selected randomly. Based on the probability of crossover, P_C, the paired strings submit to crossover in between the interval of the two selected integer positions. This results in new pairs of strings that are created by exchanging all the characters within the interval. As an example, consider two strings C and D, each of length 8 mated at random from the mating pool of the new generation:

```
C = 1 1 1 0 1 0 1 0
        ^       ^
D = 0 0 0 0 1 1 1 1
        ^       ^
```

Assume the random number generator comes out with the two cross sites at 2 and 6, respectively, the bits are interchanged between string C and string D at the positions marked with the symbol "^". The resulting two-point crossover yields two new strings, C* and D* as follows:

```
C* = 1 0 0 0 1 1 1 0
        ^       ^
D* = 0 1 1 0 1 0 1 1
        ^       ^
```

Inversion

Inversion is a process in which the ordering of strings is rearranged so that GA can search for a good string arrangement and good allele sets for employing other operators. According to the inversion probability, inversion randomly operates to choose an individual string from the mating pool. Two points are then randomly chosen along the length of the string and the binary values of the string are alternated within the two selected bit positions. For example, assume C* and D* from above are selected for inversion operation at the same cross sites, 2 and 6.

```
C* = 1 0 0 0 1 1 1 0
       ^       ^
D* = 0 1 1 0 1 0 1 1
       ^       ^
```

The resulting strings become:

```
C** = 1 1 1 0 0 0 1 0
        ^       ^
D** = 0 0 1 0 1 1 1 1
        ^       ^
```

The inversion probability is normally designed to be not too large in order to maintain the built-up quality of the population.

The operational procedure of using advanced genetic algorithms is as follows:

1. Initialize the 0th generation of population of 30 strings randomly
2. Decode each individual string to determine the four state feedback control gains
3. Perform a time simulation of the system response to the given model and evaluate the fitness function
4. Repeat steps 2-3 for all 30 strings in the population
5. Select strings for the next generation by using the stochastic remainder selection without replacement scheme
6. Crossover pairs of strings in the new generation by using the two-point crossover operator and meanwhile manipulate the inversion and the mutation operators with the new population according to their rate of probabilities.
7. Return to step 2

The entire procedure is run for 200 generations. SGA and IGA are run simultaneously with the advanced GA (AGA) in order to compare results.

4.4.3 Advanced GA Results

Table 4.1: Comparison of the four different optimal methods

	MAX. FITNESS (x10E-005)	# OF GENERAT-IONS TAKEN	# OF FUNCTION EVALUATIONS
Simple GA	1.425625	200	6000
Improved GA	1.441278	165	4950
Advanced GA	1.441278	103	3090
LQR	1.441016	Not Applicable	Not Applicable

Figure 4.4.4 shows the maximum fitness curve for each of the 200 generations. For a more complicated problem, contrary to the previous two problems, the three different GA schemes take more generations to reach the maximum fitness value. The maximum fitness is found to be 1.441278e-005 and the LQR solution obtained from MATLAB is calculated as 1.441016e-005. Figure 4.4.4 indicates that the robustness of advanced GA is superior to both improved GA and simple GA. The AGA reaches the optimal region slightly after the 100th generation, while the IGA does not reach the same region until the 165th generation. The SGA result approached the LQR optimal solution, but did not reach it by the 200th generation. The summary of the maximum fitness simulated by each technique is shown in Table 4.1.

Figure 4.4.5 shows the average fitness curve for each of the 200 generations. At the beginning of the run the average fitness of the population generated by the three techniques is fairly low. As the number of generations increases, there is an upward trend found among the fitnesses towards the maximum fitness value. Once again, genetic noises are found as the number of generations increases which are due to the probabilistic nature

of the algorithm. Clearly, throughout the 200 generations, the AGA obtains the highest average fitness among the three GA techniques. Conversely, the average fitness run by SGA drops rapidly after the 170th generation. This is an inconsistency found in the SGA.

Figure 4.4.6 and figure 4.47 show the four state feedback gain values of K1, K2, K3 and K4 corresponding to the maximum string fitness of each generation run by AGA. The states representing linear motion obtain feedback gain values close to the LQR values. On the other hand, the angular motion states generate a very noisy effect through the progression. The reason for this is due to the nonlinear characteristic of the physical system model.

Figure 4.4.8 gives a 3-D mesh plot of the number of fitness versus the fitness values for each generation. Figures 4.49 - 4.4.12 show the full-state feedback response of the loading bridge model simulated by the AGA design. Finally, figures 4.13 - 4.16 show the full-state feedback response of the loading bridge with the given reference input voltage.

The computer running rate of each algorithm for each generation is stated as: 9.20 sec/gen for SGA, 9.28 sec/gen for IGA, and 9.65 sec/gen for AGA.

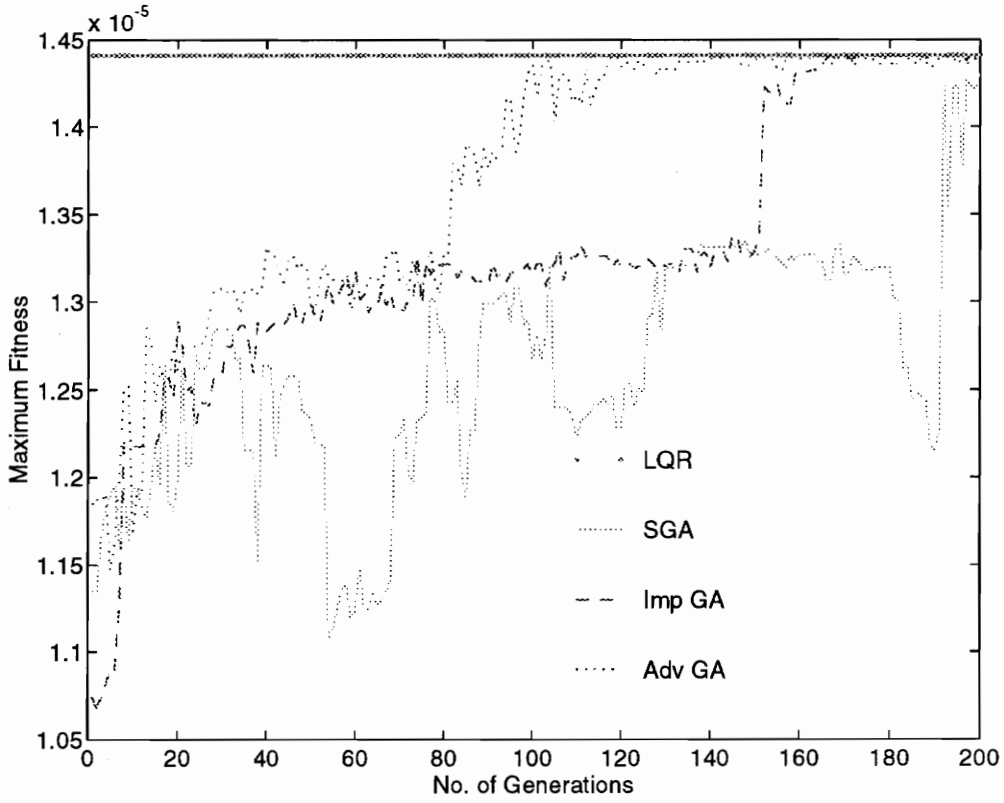

Figure 4.4.4: Maximum Fitness Curve for the Loading Bridge

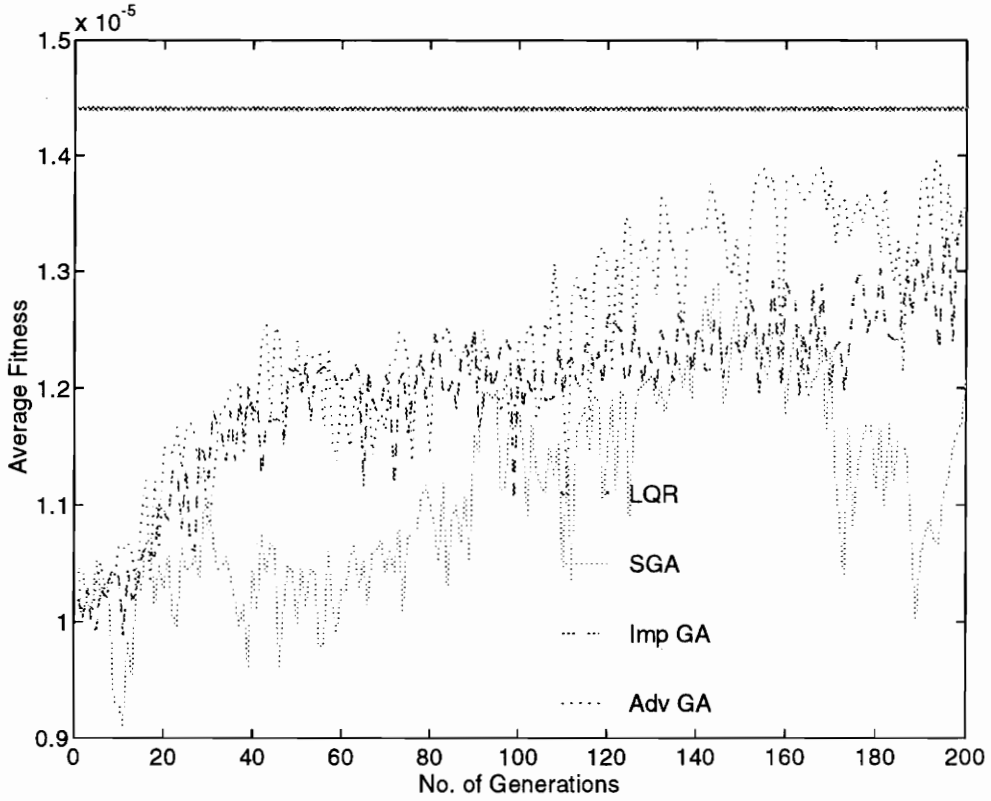

Figure 4.4.5: Average Fitness Curve for the Loading Bridge

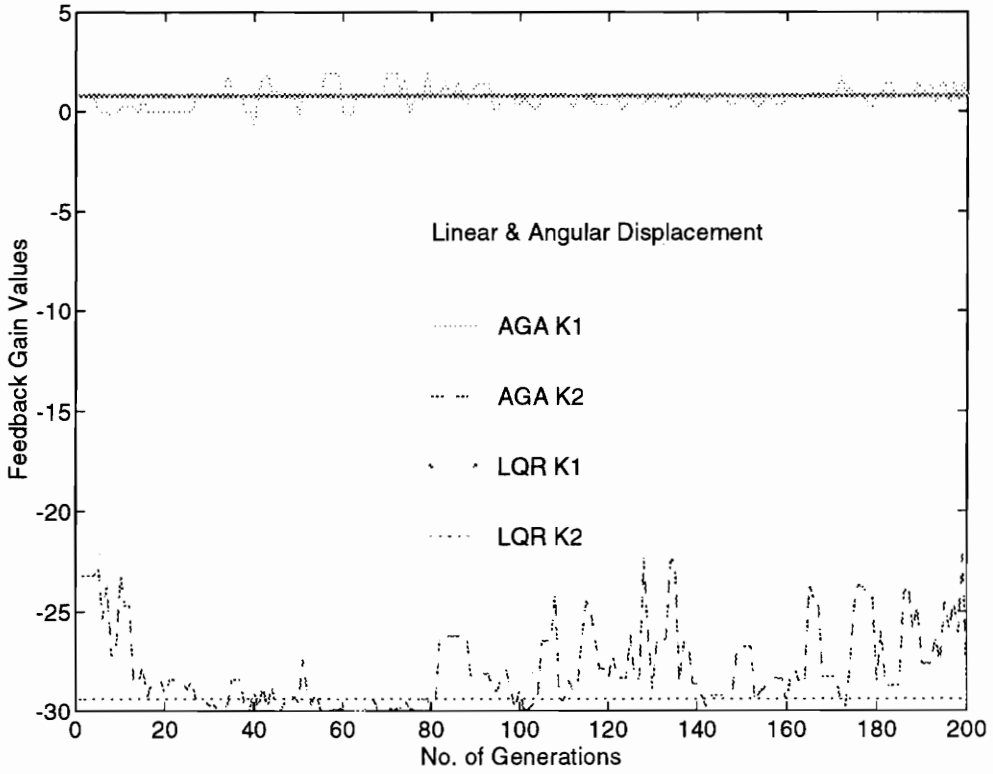

Figure 4.4.6: Two-State Feedback Gains for the Loading Bridge

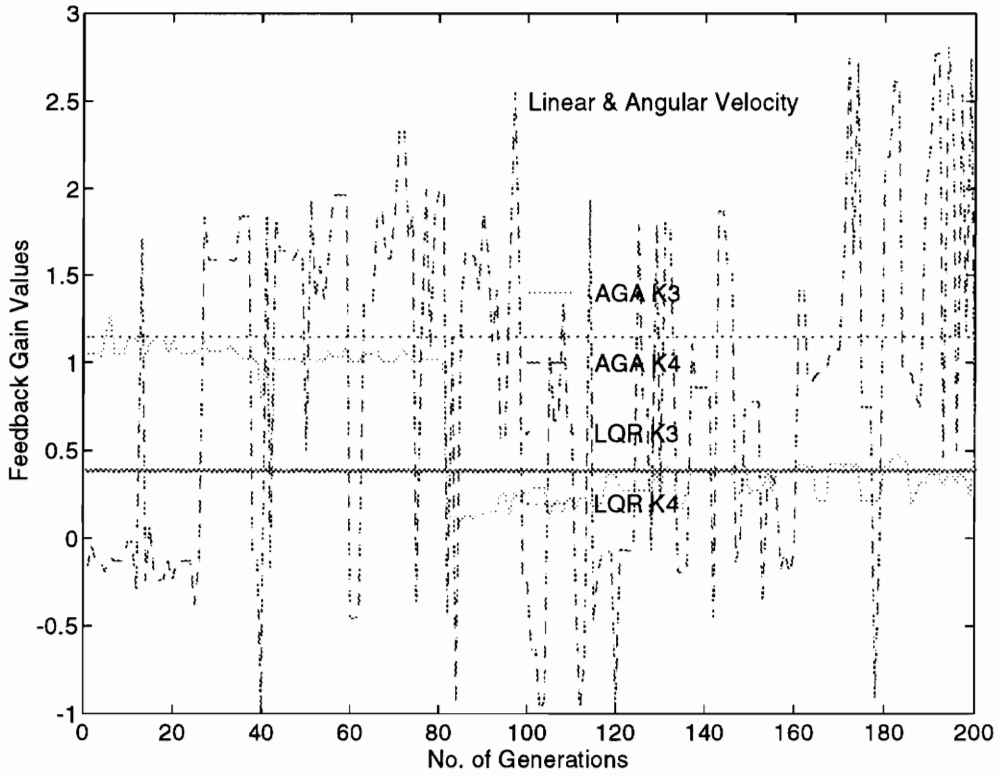

Figure 4.4.7: Two-State Feedback Gains for the Loading Bridge

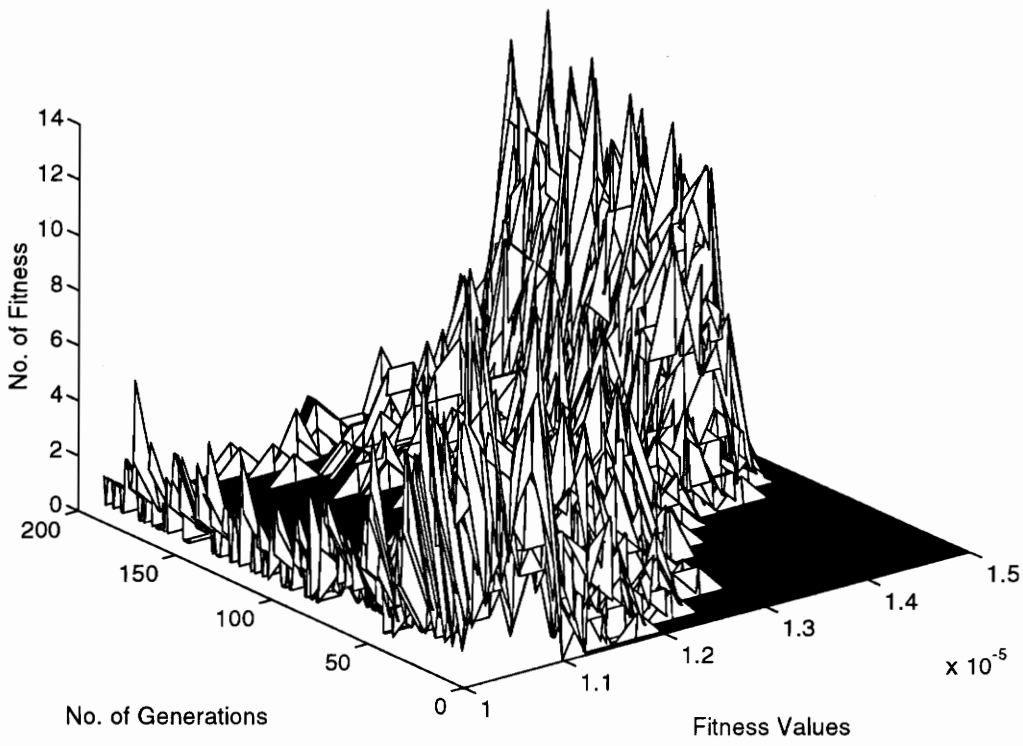

Figure 4.4.8: 3-D Mesh Plot of Population Fitness vs. No. of Generations for the Loading Bridge

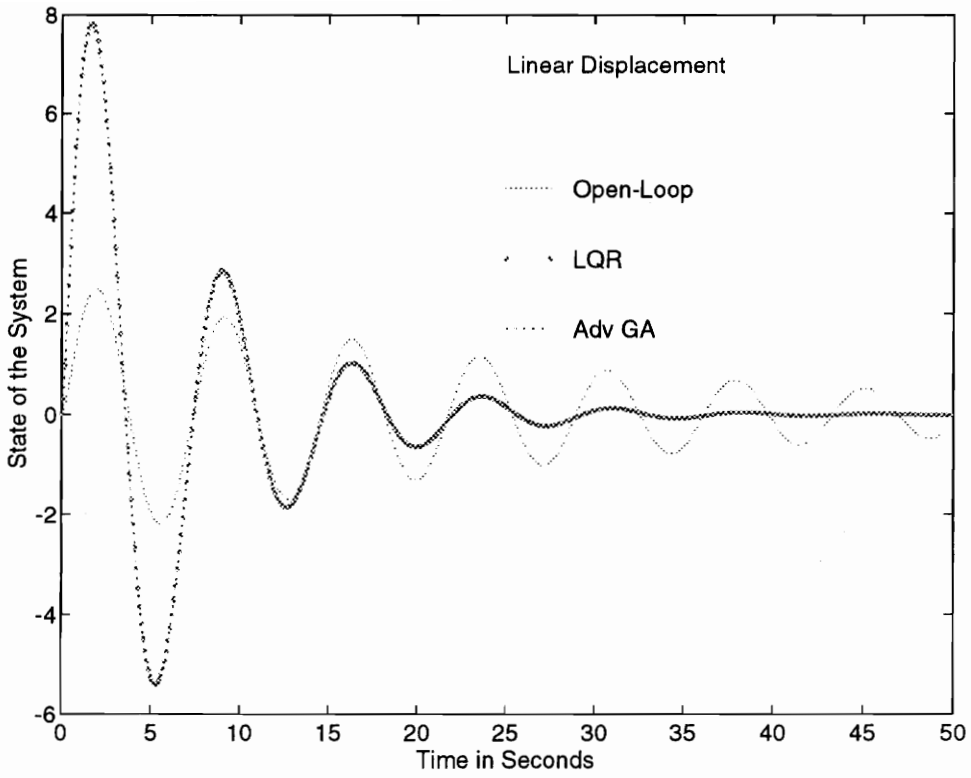

Figure 4.4.9: Full-State Feedback Response of the Loading Bridge

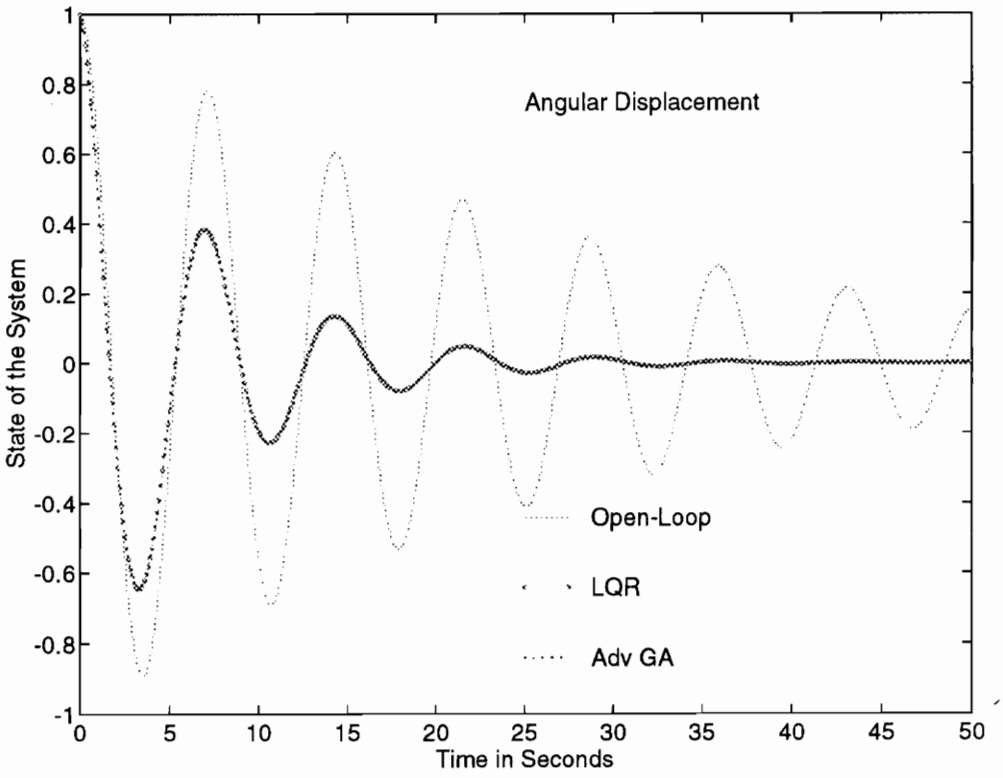

Figure 4.4.10: Full-State Feedback Response of the Loading Bridge

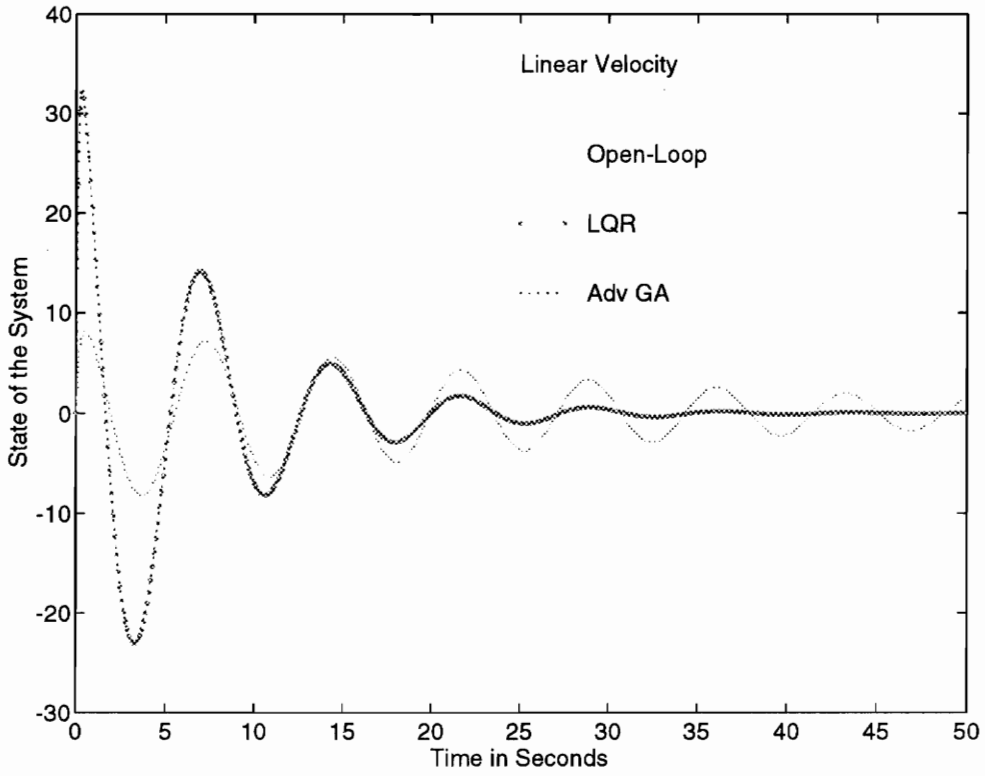

Figure 4.4.11: Full-State Feedback Response of the Loading Bridge

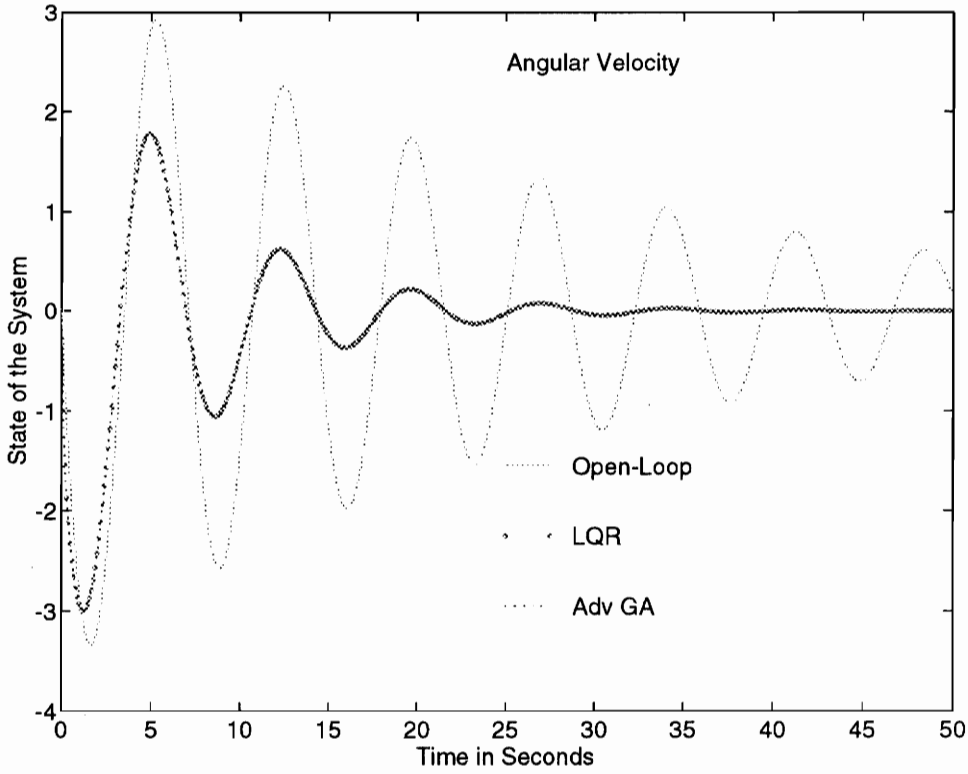

Figure 4.4.12: Full-State Feedback Response of the Loading Bridge

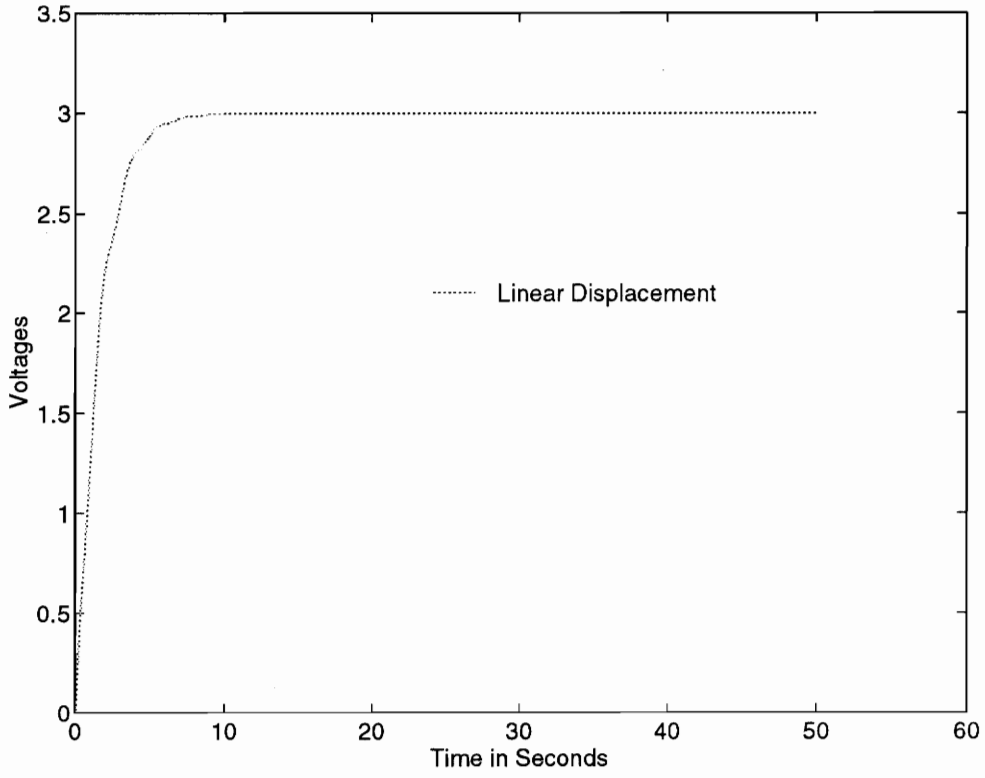

Figure 4.4.13: Feedback Response with a Reference Input of the Loading Bridge

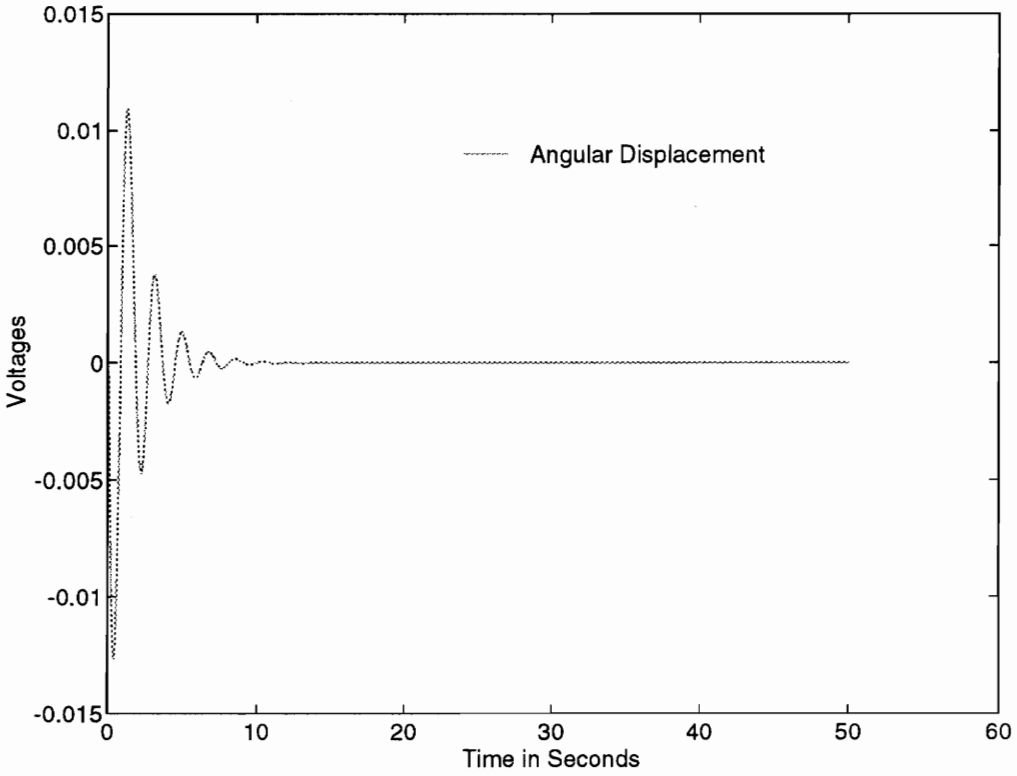

Figure 4.4.14: Feedback Response with a Reference Input of the Loading Bridge

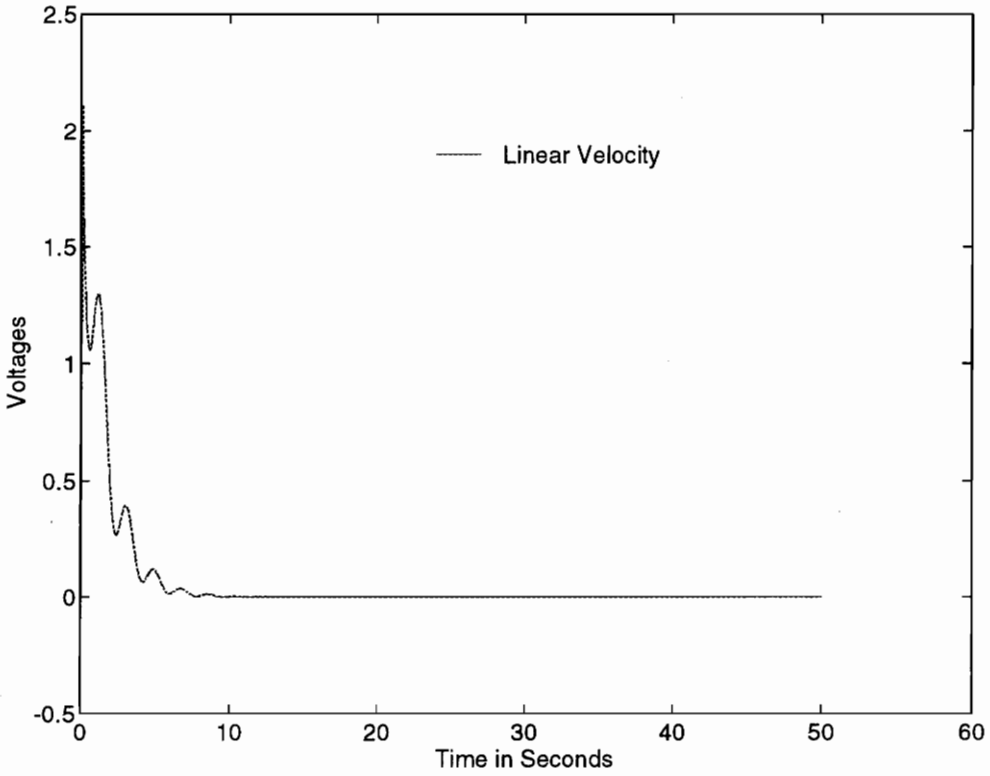

Figure 4.4.15: Feedback Response with a Reference Input of the Loading Bridge

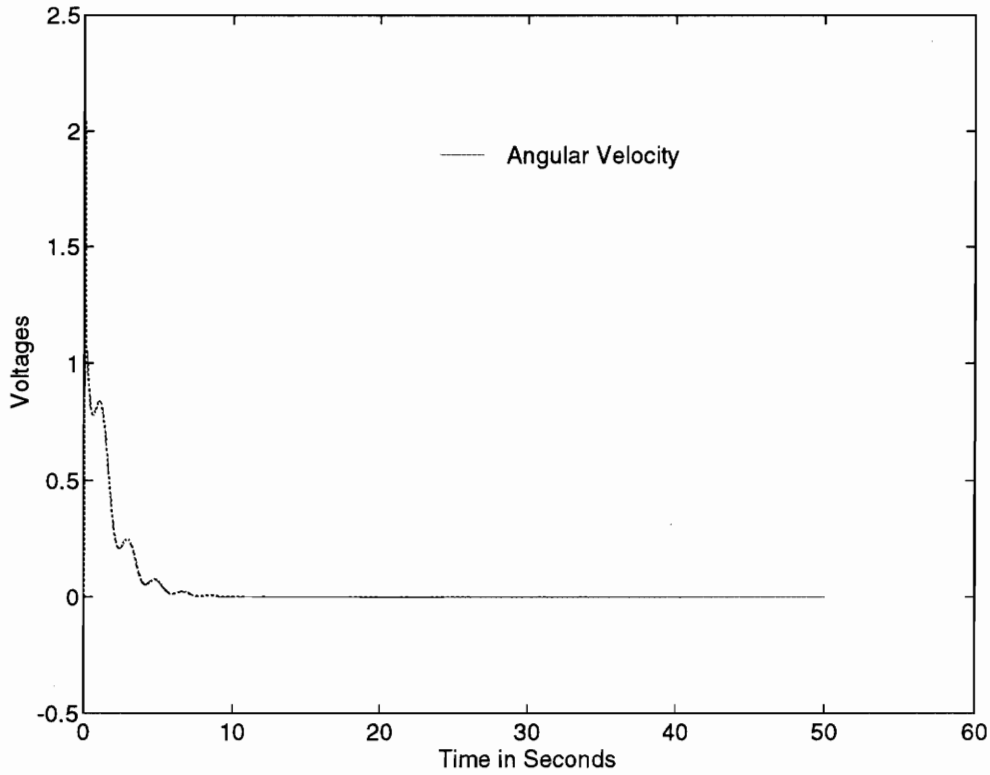

Figure 4.4.16: Feedback Response with a Reference Input of the Loading Bridge

Chapter 5

Conclusions and Recommendations

5.1 Conclusions

This work has demonstrated the robustness and the efficiency of using the genetic algorithm (GA) as a global search optimization technique. The genetic approach based on the mechanics of natural selection and population genetics is simple, robust and efficient and can be adapted to a variety of problem domains. The design, implementation and testing of the algorithm were discussed in detail. Three design optimization problems were solved using genetic algorithms. The results were compared with the traditional numerical techniques and the conventional optimal control method.

In the sixth-order polynomial optimization problem, the global maximum of the function is solved by using three different techniques. Each simulated result was compared to the true solution obtained by the analytical calculus method. Of the three techniques, the Newton-Rapshon method requires a proper initial point to be picked in order to search for the global peak value. Similarly, the golden section method requires a proper set range of parameters to determine the optimal value. The simple genetic algorithm (SGA) only requires the objective (fitness) function with accurate results found in up to 99.9967 % of tried cases. There were 150 function evaluations taken. Hence, the

SGA is concluded to be more robust than the traditional numerical techniques in solving multi-modal functions.

The selection of different sizes of the genetic operators was investigated. Reproduction, crossover, and mutation operators were used. While one of the operators is being tested at different sizes, the size of the other two operators will be kept constant. The results show that lower crossover rates tend to deteriorate the search, whereas higher crossover rates make the search tend towards the optimal solution. However, if the crossover rate is chosen too high while the mutation rate is too low, the maximum fitness might stagnate in the wrong region causing the problem of premature convergence. As the mutation rate increases, noise is added to the population fitnesses. As the mutation rate is pushed increasingly towards one, the algorithm acts more and more like a random search. Finally, as the population sizes increase, there are fewer generations required for the population to reach its optimal region. In order to optimize the output performance of the algorithm for this problem, the parametric size of each genetic operator was chosen such that population size = 30, crossover rate = 0.85, and mutation rate = 0.01.

In the second design problem, genetic algorithms were used to design a two-state feedback optimal gain set for a SDOF SMD model with a given initial condition. There is no forcing function on the system, so the system will vibrate freely and die out with time. An adaptive GA-based feedback controller was designed to control the system with the given initial condition. An improved selection scheme called the stochastic remainder selection without replacement was introduced to replace the roulette wheel selection

method. The LQR optimal solution agreed with the found maximum fitness. Result showed that the Improved GA (IGA) reaches the maximum value about 10 generations earlier than SGA. This indicates the improved robustness of IGA over SGA.

Lastly, a regulator control problem is presented using advanced genetic algorithms (AGA). A loading bridge is chosen as the dynamic control model. Two advanced genetic operators were introduced. They are the two-point crossover and the inversion operators. An AGA-based full-state feedback controller was designed to control the loading bridge with the given reference input voltage. Results showed the robustness of AGA to be superior to both IGA and SGA. The AGA reached the optimal region at the 103th generation, while the IGA did not come to the same optimal region until the 165th generation. The performance of AGA is 60.19 % more efficient than IGA. The SGA run did not even achieve the maximum fitness value. The performance of AGA is about 94.17 % more efficient than SGA. Among the three techniques discussed, the advanced genetic algorithm has been demonstrated as the most robust one for the design of feedback controllers.

5.2 Further Recommendations

The two control optimization problems using genetic algorithms were utilized in an off-line control mode. The function evaluations were simulated and the best result was saved after a number of generations given by the user. The control model was given accurately for computing the optimal control gain sets. However, when the accurate model is not

available or the plant model is changing, can GA still be sufficiently robust to have the learning-control ability? There is much more work that could be done to investigate the ability of the genetic algorithms to adapt to a real-time control process.

A suggested system layout for performing an on-line GA-based control process is shown in Fig. 5.1 [45]. The following procedures are taken in this design process.

1. Identification and estimation of the process model
2. Design of a reference model based on the controller's specifications
3. Optimization of the control parameters using the genetic algorithm

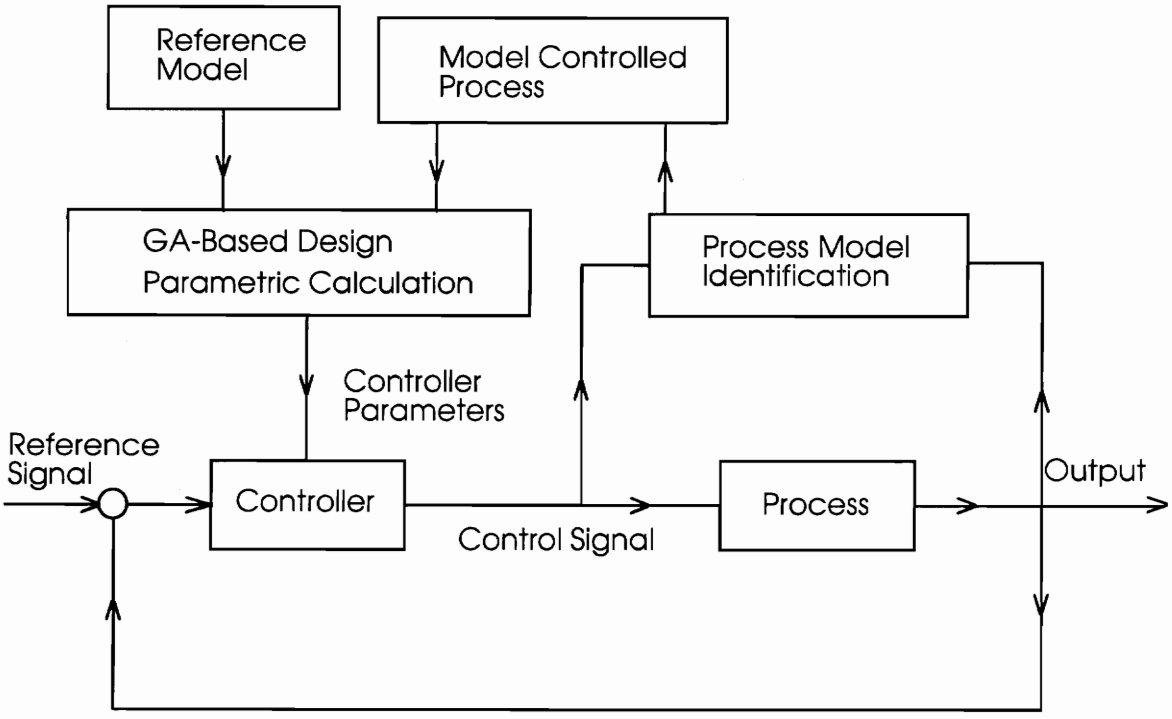

Figure 5.1: Basic Structure of a GA-Based Adaptive Controller

1. Identification and Estimation: Estimation of the process parameters is an essential part in the design of control systems. The system model can be identified and estimated by using the recursive least squares method to find its corresponding system coefficients.

2. Designing a Reference Model: An adequate reference model is designed based on the constraints placed on the controller (e.g. minimum settling time, maximum overshoot and peak time of the system response).

3. Control Optimization: The control parameters are optimized and determined by using the genetic algorithm. This part of the work has been discussed in detail in the previous sections.

Lastly, researchers recently started to focus on a hybrid scheme which is based on a combination of the genetic algorithm with other existing control techniques. Fuzzy logic and neural networks can be combined with genetic algorithms to develop hybrid adaptive controllers. These kinds of controllers would produce more efficient and robust optimal solutions than implementing only one of the existing control techniques.

References

[1] Anderson, B. D. O. and Moore, J. B., "Optimal Control: Linear Quadratic Methods," Prentice-Hall, Inc, 1990.

[2] Åström, K. J. and Wittenmark, B., "Adaptive Control," Addison-Wesley Publishing Company, May, 1989.

[3] Chalam, V. V., "Adaptive Control Systems: Techniques and Applications," Marcel Dekker, Inc., 1987.

[4] Curtis, A. R. D., "An application of genetic algorithms to active vibration control," *Journal of Intelligent Material Systems and Structures*, 1991.

[5] Davis, L., "Handbook of Genetic Algorithms," Van Nostrand Reinhold, 1991.

[6] DeCarlo, R. A., "Linear Systems: A State Variable Approach With Numerical Implementation," Prentice-Hall, Inc, 1989.

[7] De Jong, K. A., "Analysis of the behavior of a class of genetic adaptive systems," *Doctoral dissertation*, University of Michigan Press, Ann Arbor, 1975.

[8] Dimarogonas, A. D. and Haddad, S., "Vibration for Engineers," Prentice-Hall, Inc, 1992.

[9] Eaton, M., "Learning control of an inverted pendulum using a genetic algorithm," *AI and Cognitive Science* '92.

[10] Filipic, B. and Juricic, D., "An interactive genetic algorithm for controller parameter optimization," *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Innsbruck, Austria*, 1993. p. 458-62

[11] Fogarty, T. C., Vavak, F., and Cheng, P., "System identification for load balancing with the genetic algorithm," *International Conference on Control '94,* Conventry, UK p. 750-3 vol.1

[12] Franklin, G. F., Powell, J. D. and Workman, M. L., "Digital Control of Dynamic Systems," Addison-Wesley Publishing Company, Inc, 1990.

[13] Friedland, B., "Control System Design: An Introduction to State-Space Methods," McGraw-Hill, Inc, 1987.

[14] Goldberg, D. E., "Computer-Aided Pipeline Operation Using Genetic Algorithms and Rule Learning. PART I: Genetic Algorithms in Pipeline Optimization," Engineering with Computers, 1987.

[15] Goldberg, D. E., "Computer-Aided Pipeline Operation Using Genetic Algorithms and Rule Learning. PART II: Rule Learning Control of a Pipeline Under Normal and Abnormal Conditions," Engineering with Computers, 1987.

[16] Goldberg, D. E., "Genetic Algorithms in Search, Optimization and Machine Learning," Addison-Wesley., first edition, 1989.

[17] Grenfenstette, J. J., "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, 1986.

[18] Hang, C. C., Lee, T. H. and Ho, W. K., "Adaptive Control," Instrument Society of America, 1993.

[19] Holland, John H., "Adaptation in Natural and Artificial Systems," University of Michigan Press, Ann Arbor, 1975.

[20] Homaifar, A. and McCormick, E., "Full design of fuzzy controllers using genetic algorithms," *SPIE in Neural and Stochastic Methods in Image and Signal Processing*, 1992 vol.1766

[21] Huang, Y. and Chan, S. -P., "Optimizing the Performance of Genetic Algorithms for Finding the Optimal Value of a Given Function," *Proceedings of the 34th Midwest Symposium on Circuits and Systems*, 1992.

[22] Huang, R. and Fogarty, T. C., "Adaptive Classification and Control-Rule Optimisation via a Learning Algorithm for Controlling a Dynamic System," *Proceedings of the 30th IEEE Conference on Decision and Control*, 1991.

[23] Isermann, R., Lachmann, K. -H. and Matko, D., "Adaptive Control Systems," Prentice Hall International (UK) Ltd, 1992.

[24] James, M. L., Smith, G. M., Wolford, J. C. and Whaley, P. W., "Vibration of Mechanical and Structural Systems: with Microcomputer Applications," Harper & Row, Publishers, Inc, 1989.

[25] Janikow, C. Z. and Michalewicz, Z., "A Specialized Genetic Algorithm for Numerical Optimization Problems," *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, 1990.

[26] Karr, C. L., "An adaptive system for process control using genetic algorithms," *Artificial Intelligence in Real-Time Control 1992.* p.329-34

[27] Krishnakumar, K. and Goldberg, D. E., "Control System Optimization using Genetic Algorithms," *Journal of Guidance, Control, and Dynamics*, 1992.

[28] Kristinsson, K. and Dumont, G. A., "System Identification and Control using Genetic Algorithms," *IEEE Transactions on Systems, Man and Cybernetics*, 1992.

[29] Kristinsson, K. and Dumont, G. A., "Genetic Algorithms in System Identification," *Proceedings IEEE International Symposium on Intelligent Control* 1988, 1989.

[30] Lansberry, J. E., Wozniak, L. and Goldberg, D. E., "Optimal Hydrogenerator Governor Tuning with a Genetic Algorithm," *IEEE Transactions on Energy Conversion*, 1992.

[31] Linkens, D. A. and Nyongesa, H. O., "Real-time acquisition of fuzzy rules using genetic algorithms," *Artificial Intelligence in Real-Time Control 1992.* p.335-9

[32] Maclay, D. and Dorey, R., "Application of Genetic Search Techniques to Drivetrain Modelling," *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 1992.

[33] McGregor, D. R., Odetayo, M. O. and Dasgupta, D., "Adaptive Control of a Dynamic System using Genetic-Based Methods," *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 1992.

[34] Meleshchuk, S. B. and Shcherbakov, N. V., "Experimental off-line test stand for adjusting real-time adaptive control systems," *Journal of Computer and Systems Sciences International*, March-April, 1993.

[35] Michalewicz, Z., Krawczyk, J. B., Kazemi, M. and Janikow, C. Z., "Genetic algorithms and optimal control problems," *Proceedings of the 29th IEEE Conference on Decision and Control*, 1990.

[36] Ogata, K., "Modern Control Engineering," Prentice-Hall, Inc, 1990.

[37] Oliveira, P., Sequeira, J. and Sentieiro, J., "Selection of Controller Parameters using Genetic Algorithms," *Engineering Systems with Intelligence. Concepts, Tools and Application*, 1991.

[38] Parker, J. K., Tan, C., and Deb, K., "Determining PID Control Gains by Genetic Algorithms," *Twenty-Second Annual Pittsburgh Conference on Modeling and Simulation*, 1991.

[39] Rao, S. S., Pan, T. -S. and Venkayya, V. B., "Optimal placement of actuators in actively controlled structures using genetic algorithms," *AIAA Journal*, 1991.

[40] Renders, J. M., Nordvik, J. P. and Bersini, H., "Genetic algorithms for process control: a survey," *Artificial Intelligence in Real-Time Control 1992.* p.323-8

[41] Sastry, S. and Bodson, M., "Adaptive Control: Stability, Convergence, and Robustness," Prentice-Hall Advanced Reference Series, 1989.

[42] Schaffer, J. D., Caruana, R. A., Eshelman, L. J. and Das, R., "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, 1989.

[43] Schraudolph, N. N. and Belew, R. K., "Dynamic Parameter Encoding for Genetic Algorithms," *Machine Learning*, 1992.

[44] Solano, J. and Jones, D. I., "Parameter determination for a genetic algorithm applied to robot control," *International Conference on Control '94*, Conventry, UK p.765-70 vol.1

[45] Srivastava, R. P., "Use of genetic algorithms for optimization in digital control of dynamic systems," *1992 ACM Computer Science Conference.* Communications Proceedings p.219-24

[46] Swokowski, E. W., "Calculus with Analytic Geometry," PWS-KENT Publishing Company, 1988.

[47] Szarkowicz, D. S., "A Genetic Algorithm for Minimum-Time Trajectories," *Proceedings of the 1992 Summer Computer Simulation Conference.* Twenty-Fourth Annual Computer Simulation Conference, 1992.

[48] Thierens, D. and Vercauteren, L., "A Topology Exploiting Genetic Algorithm to Control Dynamic Systems," Parallel Problem Solving from Nature. *1st Workshop, PPSN 1 Proceedings*, 1991.

[49] Thomson, W. T., "Theory of Vibration with Applications," Prentice-Hall, Inc, 1988.

[50] Ueyama, T., Fukuda, T. and Arai, F., "Coordinate Planning using Genetic Algorithm - Structure Configuration of Cellular Robotic System," *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 1992.

[51] Urbancic, T., Juricic, D., Filipic, B. and Bratko, I., "Automated synthesis of control for nonlinear dynamic systems," Artificial Intelligence in Real-Time Control 1992. p.341-346

[52] Vanderplaats, G. N., "Numerical Optimization Techniques for Engineering Design: With Applications," McGraw-Hill, Inc, 1984.

[53] Varsek, A., Urbancic, T. and Filipic, B., "Genetic algorithms in controller design and tuning," *IEEE Transactions on Systems, Man and Cybernetics*, 1993. vol.23 p.1330-9

[54] Wang, P., and Kwok, D. P., "Auto-Tuning of Classical PID Controllers using an Advanced Genetic Algorithm," *Proceedings of the 1992 International Conference on*

Industrial Electronics, Control, Instrumentation, and Automation. Power Electronics and Motion Control.

[55] Wang, P. and Kwok, D. P., "Optimal fuzzy PID control based on genetic algorithm," *Prceedings of the 1992 International Conference on Industrial Electronics, Control, Instrumentation, and Automation. Power Electronics and Motion Control.*

[56] Widrow, B. and Stearns, S. D., "Adaptive Signal Processing," Prentice-Hall, Inc, 1985.

Vita

Nga Hin Benjamin Fong was born in Hong Kong on July 25, 1970. He grew up in Hong Kong, where he completed 9th grade at St. Joseph's College. With the support of his parents, he left home in September, 1985 to continue his high school education at Rishworth School, England. In order to pursue a higher education, he came to U.S.A. to begin his college program in the fall of 1988 at the University of Texas at Arlington. After receiving his B.S.M.E. degree in May of 1992, he decided to pursue a Master's degree in Mechanical Engineering at Virginia Tech. Two years and one hundred days later, he successfully defended his thesis. His future plans are uncertain; he is currently seeking employment in the automotive industry.