

**REAL-TIME TRAJECTORY OPTIMIZATION  
USING A CONSTRAINED GENETIC ALGORITHM**

by

**Paul G. van Deventer**

B.Eng Universiteit van Stellenbosch (1990)


Submitted to the Department of Aeronautics and Astronautics  
in Partial Fulfillment of the Requirements for the  
Degree of

**Master of Science**  
in Aeronautics and Astronautics  
at the

**Massachusetts Institute of Technology**  
June 1993

© Massachusetts Institute of Technology 1993  
All rights reserved


Signature of Author: \_\_\_\_\_

  
Department of Aeronautics and Astronautics  
April 13, 1993

Certified by: \_\_\_\_\_

Professor Wallace E. Vander Velde  
Thesis Supervisor  
Department of Aeronautics and Astronautics

Accepted by: \_\_\_\_\_

  
Professor Harold Y. Wachman  
Chairman, Department Graduate Committee

**Aero**

1

MASSACHUSETTS INSTITUTE  
OF TECHNOLOGY

**JUN 08 1993**

LIBRARIES

# REAL-TIME TRAJECTORY OPTIMIZATION USING A CONSTRAINED GENETIC ALGORITHM

by

Paul G. van Deventer

Submitted to the Department of Aeronautics and Astronautics  
in partial fulfillment of the requirements for the degree of  
Master of Science in Aeronautics and Astronautics at the  
Massachusetts Institute of Technology

## Abstract

The efficiency of the optimization algorithm used in a mission planning system must be interpreted in terms of the real-time nature of the system. Continuity of commands requires that a solution be available before the aircraft reaches the next waypoint. Furthermore, the flight control commands and the flight path are respectively constrained by the aircraft dynamics and trajectory objectives. It is therefore essential to have an algorithmic structure that will rapidly produce a solution within these constraints and also allow for further improvement if time permits. This thesis investigates genetic algorithms as an alternative optimization procedure in the flight trajectory planner to enhance the real-time algorithmic efficiency. The effect of the control variables on the performance of the genetic algorithm is investigated. A comparison is also done between the real-time performance of the optimal genetic algorithm and the Broyden-Fletcher-Goldfarb-Shannon minimization routines used previously.

Thesis leader: Prof. Wallace E. Vander Velde  
Title: Professor of Aeronautics and Astronautics  
Massachusetts Institute of Technology

# Acknowledgments

I would like to express my gratitude to Prof. Vander Velde for the opportunity to work on this project. It was a pleasure to work under his guidance.

Many thanks to Mauritz for his valuable help with MATLAB, but mostly for his support and friendship.

To my friends; Kyle, James, Matt and Eric, thanks for making life at MIT enjoyable.

I am also indebted to my family, especially my grandparents, for their moral and financial support.

Lastly a special thanks to Cobus and Daan for showing me the meaning of true friendship.

---

# Contents

<b>Abstract</b>	<b>2</b>
<b>Acknowledgments</b>	<b>3</b>
<b>Glossary</b>	<b>9</b>
<b>1 Introduction</b>	<b>11</b>
<b>2 Problem Formulation</b>	<b>13</b>
2.1 Mission Planning System . . . . .	13
2.2 Flight Control System and Vehicle Dynamics . . . . .	17
2.3 Threat Function and Trajectory Risk . . . . .	21
<b>3 The Genetic Algorithm</b>	<b>23</b>
3.1 Introduction . . . . .	23
3.2 Basic Genetic Algorithm . . . . .	25
3.3 Mathematical Foundation of Genetic Algorithm . . . . .	27
3.3.1 Schemata - An Introduction . . . . .	27
3.3.2 The Fundamental Theorem . . . . .	30
3.4 The Parallel Genetic Algorithm . . . . .	32
<b>4 Flight Trajectory Planning using Genetic Algorithms</b>	<b>33</b>

4.1	Flight Control Commands Modeling . . . . .	33
4.1.1	Chebyshev Polynomials . . . . .	34
4.1.2	Filtered Piecewise Constant . . . . .	36
4.1.3	Gray Code vs. Binary Code . . . . .	38
4.1.4	Initialization of Population . . . . .	40
4.2	Reproduction . . . . .	43
4.2.1	Fitness Function . . . . .	43
4.2.2	Selection . . . . .	44
4.3	Constrained Optimization . . . . .	47
4.3.1	Flight Control Commands . . . . .	47
4.3.2	Search Space . . . . .	48
4.4	Crossover, Mutation and Population Size . . . . .	49
<b>5</b>	<b>Results</b>	<b>56</b>
<b>6</b>	<b>Conclusions</b>	<b>66</b>
6.1	Summary . . . . .	66
6.2	Recommendations . . . . .	68

# List of Figures

2.1	Structure of the MPS . . . . .	15
2.2	Two-waypoint trajectory extension and concatenation . . . . .	16
2.3	Aircraft body axis system and inertial axis system . . . . .	19
2.4	Bound on roll angle . . . . .	21
3.1	The effectiveness of different search algorithms over different problem types. . . . .	25
3.2	The basic genetic algorithm cycle. . . . .	27
3.3	The crossover operator . . . . .	28
3.4	The mutation operator . . . . .	29
4.1	Coding of the Chebyshev coefficients as a genotype (chromosome length $l = 4$ ). . . . .	35
4.2	Modeling the flight control commands using Chebyshev polynomials. . . . .	37
4.3	Coding of filtered piecewise constant command as a genotype (chromosome length $l = 4$ ). . . . .	38
4.4	Modeling the flight control commands using filtered piecewise constant values. . . . .	39
4.5	Typical member of initial population obtained using the random filtered piecewise constant method. . . . .	41
4.6	Typical member of initial population obtained using the Chebyshev polynomial and sampling method. . . . .	42

4.7	Linear fitness scaling without adjustment for negative fitness values ( $C_{\text{mult}} = 2$ ). . . . .	45
4.8	Linear mapping of roll angle between $\Phi_{c_{\text{min}}}$ and $\Phi_{c_{\text{max}}}$ . . . . .	47
4.9	The two point crossover operator . . . . .	50
4.10	Average off-line performance of the genetic algorithm as a function of crossover probability. . . . .	52
4.11	Average off-line performance of the genetic algorithm as a function of mutation probability. . . . .	53
4.12	Average off-line performance of the genetic algorithm as a function of population size. . . . .	54
4.13	Average off-line performance of the genetic algorithm as a function of total number of trajectory evaluations. . . . .	55
5.1	Result of genetic algorithm optimization of test segment 1. . . . .	58
5.2	Result of genetic algorithm optimization of test segment 2. . . . .	59
5.3	Result of genetic algorithm optimization of test segment 3. . . . .	60
5.4	Result of genetic algorithm optimization of test segment 4. . . . .	61
5.5	Flight control commands for different values of maximum normal ac- celeration. . . . .	62
5.6	Flight trajectories for different values of maximum normal acceleration. . . . .	63
5.7	Evolution of the best constrained solutions available to the FCS for test segment 1. . . . .	64
	Evolution of the best constrained solutions available to the FCS for test segment 3. . . . .	65

# List of Tables

3.1	Comparison of natural and artificial genetic algorithm terminology. .	26
4.1	Comparison between Gray coded and binary coded integers. . . . .	40
4.2	Off-line performance of the genetic algorithm shown as the average maximum fitness of 240 experiments using 4 different problems. . . .	51



# Glossary

$a_{nc}$	normal acceleration command
$a_n$	normal acceleration
$\Phi_c$	roll angle command
$\Phi$	roll angle
$\Theta$	pitch angle
$\Psi$	yaw angle
$\mathbf{V}$	aircraft velocity vector
$V_{nom}$	nominal aircraft velocity
$\alpha$	angle of attack
$\beta$	sideslip angle
$\mathbf{X}_E, \mathbf{Y}_E, \mathbf{Z}_E$	earth reference frame axes
$\mathbf{X}_A, \mathbf{Y}_A, \mathbf{Z}_A$	aircraft reference frame axes
$x, y, z$	aircraft position in earth reference frame
$v_x, v_y, v_z$	aircraft velocity in earth reference frame
$a_x, a_y, a_z$	aircraft acceleration in earth reference frame
$z_{nom}$	nominal aircraft altitude
$\tau_{a_n}$	time constant of normal acceleration channel of FCS
$\tau_{\Phi}$	time constant of roll angle channel of FCS
$g$	acceleration due to gravity

$T(x, y)$	threat function
$J_i$	trajectory risk
$J_{\text{ceil}}$	constant
$\Gamma_i$	threat intensity
$s$	distance along trajectory
$t$	time
$t_i$	initial time of segment
$t_f$	final time of segment
$k$	bit position
$l$	string length
$n$	population size
$H$	schema
$m(H, t)$	number of instances of a schema
$o(H)$	schema order
$\delta(H)$	schema length
$\mathbf{A}$	population
$A_i$	string
$T_i$	Chebyshev polynomial
$\lambda$	normalized time
$\mathbf{c}$	vector of Chebyshev coefficients
$c_i$	Chebyshev coefficient
$F_i$	raw string fitness
$F_i^*$	scaled string fitness
$C_{\text{mult}}$	fitness scaling multiplier
$E_i$	expected value of a genotype
$p_c$	probability of crossover
$p_m$	probability of mutation
$p_s$	probability of crossover survival
$\Delta t$	time interval
$w_s$	sampling frequency

# Chapter 1

## Introduction

As the sophistication and performance of modern flight vehicles increases, so does the demand to lighten pilot workload by autonomous control systems. It is possible that artificial intelligence and modern control methods will soon regulate all the flight operations and allow pilots to concentrate all their efforts on mission related tasks.

This research has its foundation in a mission planning system (MPS) which is under development at the Charles Stark Draper Laboratory. The purpose of the MPS is to alleviate the workload of the pilot by generating flight control commands which integrate to an optimal flight path, subject to certain objectives. These objectives are modeled by a database which includes geographic waypoints, natural and man-made threats, and constraints imposed on both the flight control commands and the flight path.

Because of the dynamic nature of most of these objective variables it is necessary for the database to be time-variant. This requires the flight control commands to be generated in real-time. In the research done by Walker [9] a method was presented by which the flight control commands were optimized using a gradient-based, constrained, nonlinear optimization technique.

Although the optimization did converge it was a computational-intensive procedure. Another disadvantage was that the best set of control commands at an arbitrary time during the optimization, such as the time when the solution is needed in

real-time operation, were not necessarily within the specified constraints. This was a result of the iterative method used to enforce the constraints.

The purpose of this research is to investigate how a genetic algorithm can be implemented in the MPS as a search procedure to achieve better performance and make it more realizable. Genetic algorithms are search procedures which are modeled after natural selection. The underlying premise of these algorithms is that the optimal solution to a search problem can be evolved from a population of potential solutions. Genetic algorithms are well suited to the control commands optimization problem because of the following:

- Genetic algorithms do not rely on derivatives or gradients. It is therefore a highly robust search procedure which is well suited to the discontinuous search space of the control commands optimization problem. Because it is not necessary to calculate derivatives, it also leads to less computations.
- The constraints on the control commands and associated trajectories can be enforced such that the best set of control commands available at any arbitrary time during the optimization are within bounds.
- Genetic algorithms are highly parallelizable. This allows the computation time of the optimization to be greatly reduced while still achieving the same level of convergence.

This thesis addresses the implementation of a genetic algorithm in the MPS to achieve robust performance characteristics and computational efficiency. Different methods of modeling the flight control commands and different variants of the genetic algorithm operators are investigated. A statistical analysis of the stochastic genetic algorithm control parameters is also done to realize optimal performance.

## Chapter 2

# Problem Formulation

### 2.1 Mission Planning System

The MPS is a three-part flight vehicle mission planning system which formulates optimal strategies in order to accomplish specific mission-imposed goals using a detailed database of objective and threat information which is continuously updated. At the highest level of the MPS is the goalpoint planner (GP). It's function is to generate a sequence of intermediate goals and geographic locations along with associated time and energy constraints to accomplish given mission objectives. At the next level of the MPS is the high-level trajectory planner (HLTP). Using knowledge of operational facilities, weather systems, and major threat concentrations, which can be man-made or natural, the HLTP identifies a maximum survivability flight path such that the intermediate goals and associated time and energy constraints are met while avoiding major threat concentrations. The flight path is characterized by waypoints of a nominal separation determined by the vehicle operating mode characteristics, such as speed, altitude, and turning radius. Associated with each set of waypoints are:

- A time constraint which is consistent with the high-level constraints on time and energy generated by the GP.
- A set of nonzero capture radii. The values of the capture radii are measures of how close the final trajectory should pass to the respective waypoints.

- A detailed database of low-level threats to be used in the next level of the MPS.

The lower level of the MPS consists of the command planner (CP). The purpose of the CP is to generate flight control commands such that the trajectory will minimize the risk associated with the low-level threat database while satisfying the waypoint restrictions listed above. Since the nonlinear dynamics of the flight vehicle and its control system are embedded in the evaluation of the risk function (given in Equation 2.22) it provides a platform on which the flight vehicle limitations are recognized from the outset at the expense of having to optimize complex, constrained, nonlinear equations. A diagram of the MPS structure is given in Figure 2.1.

Because both the mission objectives and threat information available to the HLTP are time-varying, the solution must be updated as rapidly as possible. Each time an update becomes available a new solution is generated and replaces the old one. It is assumed that the environment does not change appreciably in the time it takes the vehicle to cover the distance between two waypoints. It will therefore be sufficient to compute a new solution each time the vehicle passes a waypoint. A look-ahead of two waypoints is assumed. The commands segment will normally be executed to its point closest to the first waypoint, at which it will be discarded for the new commands segment which spans the next two waypoints. Continuity is ensured by using the projected flight variables of the flight vehicle at its closest point to the first waypoint as initial conditions for the solution of the next two-waypoint commands segment. This method of concatenation of the flight control input commands has the disadvantage of limiting the time available to converge to a suitable solution, but the resulting trajectories are smoother and without unnecessarily sharp turns. The process is illustrated in Figure 2.2.

This approach allows the period it takes the flight vehicle to reach the first waypoint to vary according to the needs of the risk optimization process. As a result the trajectory can avoid threats at the expense of violating short-term time constraints. Although low-level time constraints are not enforced, it is possible to compensate for any errors by updating the HLTP database, forming a feedback loop which will ensure that mission-imposed constraints on time and energy are met.

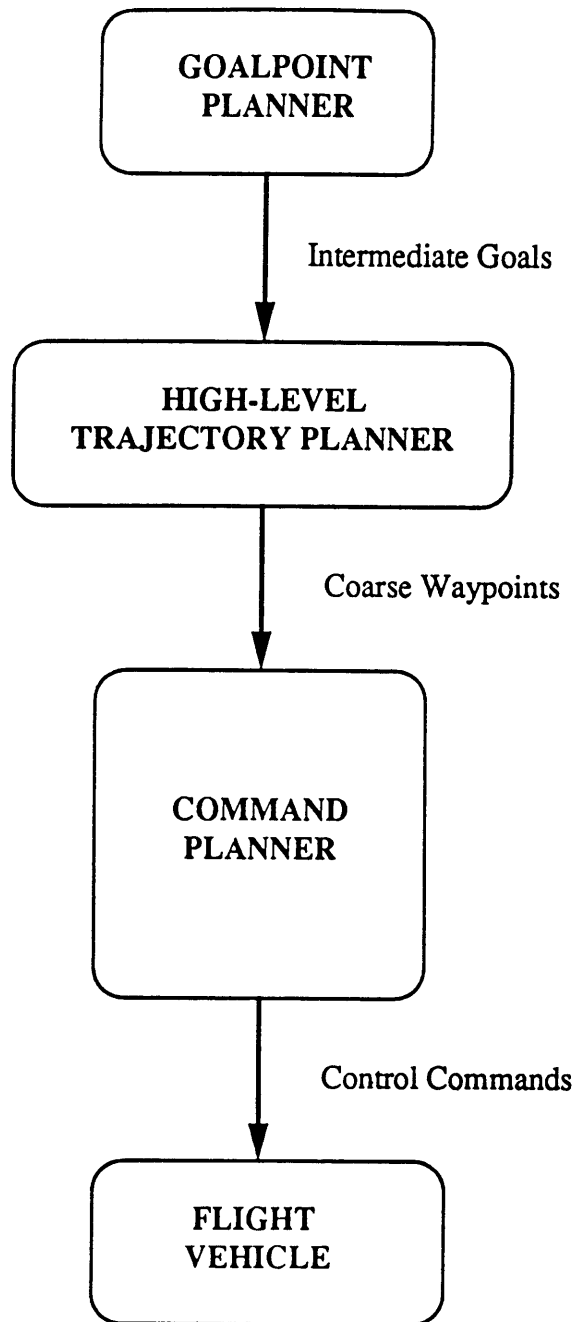


Figure 2.1: Structure of the MPS

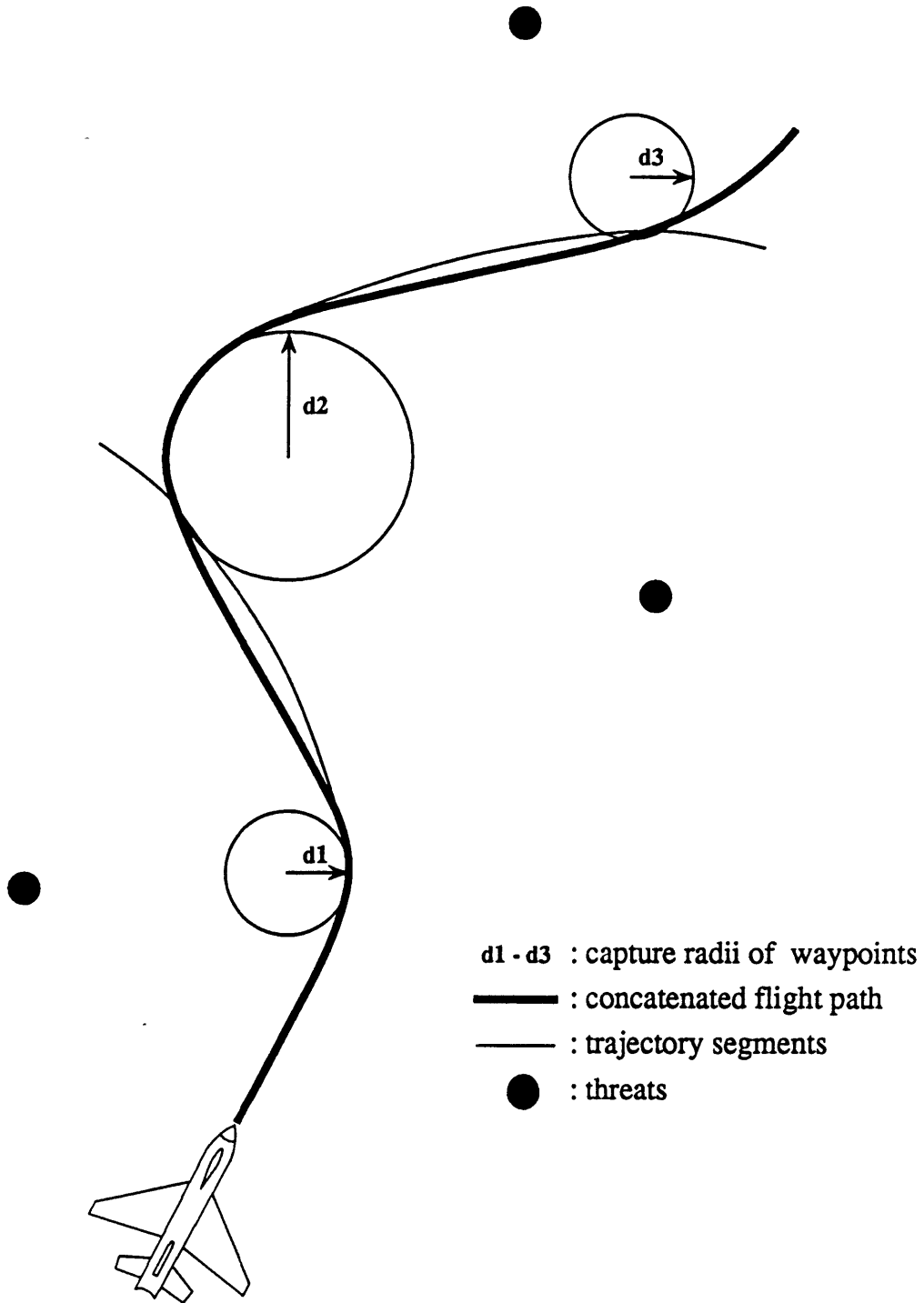


Figure 2.2: Two-waypoint trajectory extension and concatenation



## 2.2 Flight Control System and Vehicle Dynamics

As stated in Section 2.1 the dynamic behavior of the flight vehicle is an integral part of the command planner. It is necessary therefore to choose a specific application for the purpose of the research. To maintain continuity with the work of Walker [9], the flight environment was chosen based on the characteristics of a military aircraft operating at low level in a *Terrain Following/Terrain Avoidance* TF/TA mode. The following assumptions were made:

- nominal speed :  $V_{\text{nom}} = 250\text{m/s}$
- nominal altitude :  $z_{\text{nom}} = 200\text{m}$
- maximum normal acceleration :  $|a_{\text{nc}}|_{\text{max}} = 4g$

With the aircraft characteristics defined as above, the minimum turning radius is approximately 1500 m. The nominal separation between the waypoints supplied by the HLTP is then taken to be 5000 m. When operating in this flight envelope it is normal practice to maintain a constant power setting which would be chosen to meet the time constraints set by the HLTP. Thrust is therefore not considered as an active control, and the flight control inputs are taken to be normal acceleration  $a_{\text{nc}}$  and roll angle  $\Phi_c$ .

The Flight Control System (FCS) is modeled as a first order lag between each of the command variables and the corresponding physical quantities as follows:

$$a_n = \frac{a_{\text{nc}}}{1 + s\tau_{a_n}} \quad (2.1)$$

$$\Phi = \frac{\Phi_c}{1 + s\tau_{\Phi}} \quad (2.2)$$

where  $\tau_{a_n}$  and  $\tau_{\Phi}$  are the time constants associated with the normal acceleration and roll angle control channels of the FCS.

An important consideration in modeling the FCS is the ability of the aircraft to follow the optimal trajectory. To this end a more complex model of the FCS will be preferable. With gradient based optimization methods an increase in complexity adversely affected the convergence of the solution. Although genetic algorithms are not dependent on gradients, a simplified model is still preferred because of the increase in computations involved in evaluating the response of a more complex

model of the aircraft. El Dirani [4] justified the use of the Equations 2.1–2.2 by showing that the errors resulting from these simple models were of the same order as those due to air turbulence and could therefore be ignored in the presence of velocity and position feedback.

The aircraft body axis system (aircraft reference frame) and the inertial axis system (earth reference frame) are shown in Figure 2.3. For the purpose of this research the sideslip angle  $\beta$  and angle of attack  $\alpha$  are neglected so that the velocity vector  $\mathbf{V}$  of the aircraft coincides with the  $\mathbf{X}_A$ -axis in the aircraft body axis system. The  $\mathbf{Z}_E$ -axis of the inertial axis system points vertically downward, while the  $\mathbf{X}_E$ - and  $\mathbf{Y}_E$ -axes lie in the horizontal plane. The acceleration of the aircraft in the inertial axis system referred to the normal acceleration in the aircraft body axis system is given by the following Euler transformation:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = a_n \begin{bmatrix} \cos \Phi \sin \Theta \cos \Psi + \sin \Phi \sin \Psi \\ \cos \Phi \sin \Theta \sin \Psi - \sin \Phi \cos \Psi \\ \cos \Phi \cos \Theta \end{bmatrix} \quad (2.3)$$

Substituting Equations 2.1–2.2 in Equation 2.3 leads to the following state equations for the aircraft and its control system:

$$\dot{x} = v_x \quad (2.4)$$

$$\dot{y} = v_y \quad (2.5)$$

$$\dot{z} = v_z \quad (2.6)$$

$$\dot{v}_x = a_n(\cos \Phi \sin \Theta \cos \Psi + \sin \Phi \sin \Psi) \quad (2.7)$$

$$\dot{v}_y = a_n(\cos \Phi \sin \Theta \sin \Psi - \sin \Phi \cos \Psi) \quad (2.8)$$

$$\dot{v}_z = a_n(\cos \Phi \cos \Theta) + g \quad (2.9)$$

$$\dot{a}_n = \frac{1}{\tau_{a_n}}(a_{nc} - a_n) \quad (2.10)$$

$$\dot{\Phi} = \frac{1}{\tau_{\Phi}}(\Phi_c - \Phi) \quad (2.11)$$

where:

$$\Psi = \tan^{-1} \left( \frac{v_y}{v_x} \right) \quad (2.12)$$

$$\Theta = \sin^{-1} \left( \frac{-v_z}{\sqrt{v_x^2 + v_y^2 + v_z^2}} \right) \quad (2.13)$$

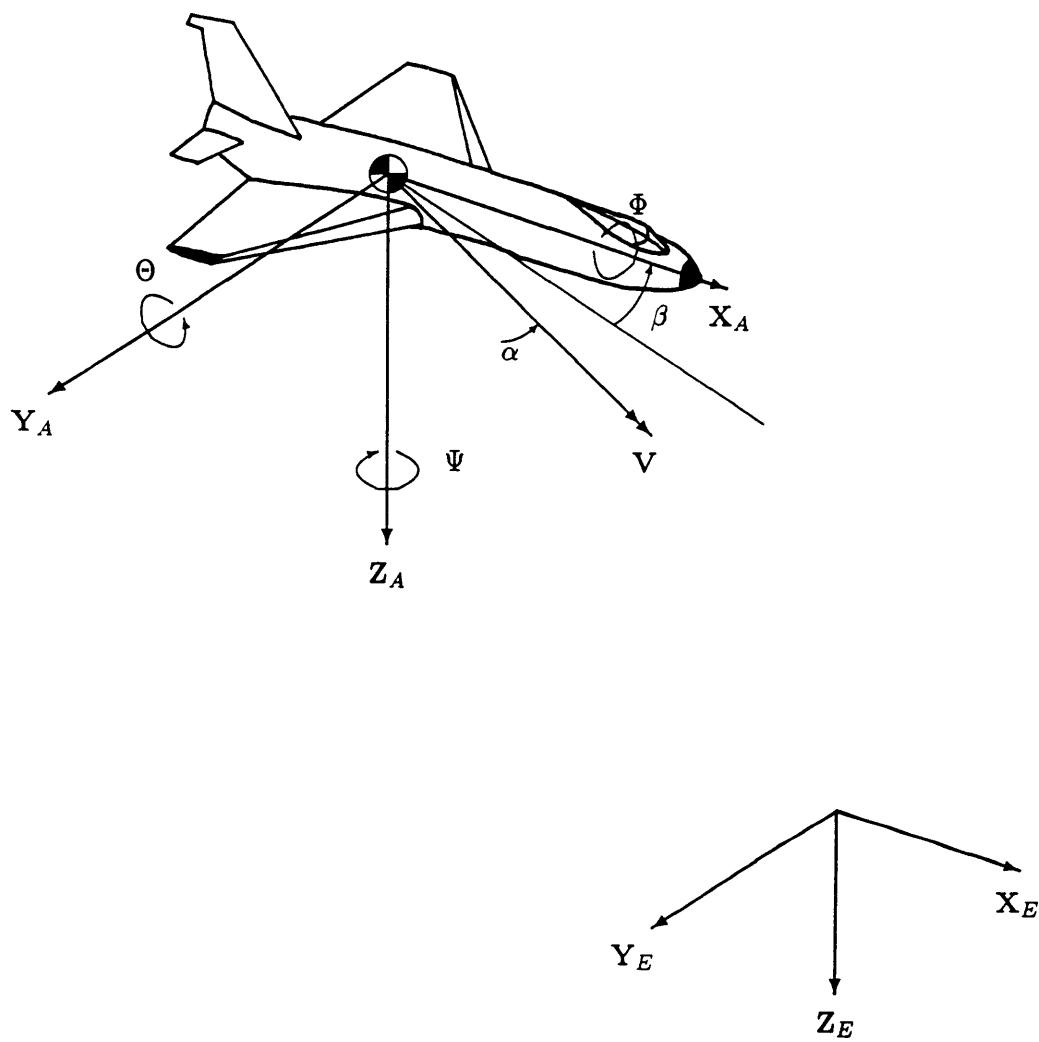


Figure 2.3: Aircraft body axis system and inertial axis system

In the previous research both the normal acceleration and the roll angle were optimized as variable inputs to the FCS. The associated flight paths were functions of horizontal position  $x, y$  and altitude  $z$ , where any deviations from a reference altitude were penalized in the cost function of the optimization algorithm. The resulting optimal trajectories deviated from the reference altitude by 50–100 m which is not desirable for the TF/TA flight environment specified where  $z_{\text{ref}} = 200$  m. Level flight at a nominal altitude without deviation can be obtained by reducing the trajectory dimensionality as follows:

$$z(0) = z_{\text{nom}} \quad (2.14)$$

$$v_z = 0 \quad (2.15)$$

$$\dot{v}_z = 0 \quad (2.16)$$

Substituting for  $v_z$  and  $\dot{v}_z$  in Equations 2.13 and 2.9 the normal acceleration becomes a function of the roll angle:

$$a_n = \frac{-g}{\cos \Phi} \quad (2.17)$$

This simplification therefore has the added advantage that it reduces the search space of flight control commands. The disadvantage is that it places constraints on the roll angle. These constraints are the following:

- the normal acceleration is a nonlinear function of the roll angle with an infinite value at  $\Phi = 90^\circ$ ,
- the bound on the normal acceleration.

The bound on the normal acceleration is the more strict of the two constraints on the roll angle. For a maximum normal acceleration of  $4g$  the roll angle is constrained to be  $|\Phi| \leq 75^\circ$  from Equation 2.17. This constraint is shown in Figure 2.4.

This simplification may not be preferable for other flight environments where it may be necessary for the flightpath to be a function of altitude: it is used for the purpose of comparing the performance of genetic algorithms to that of the gradient based method used previously.

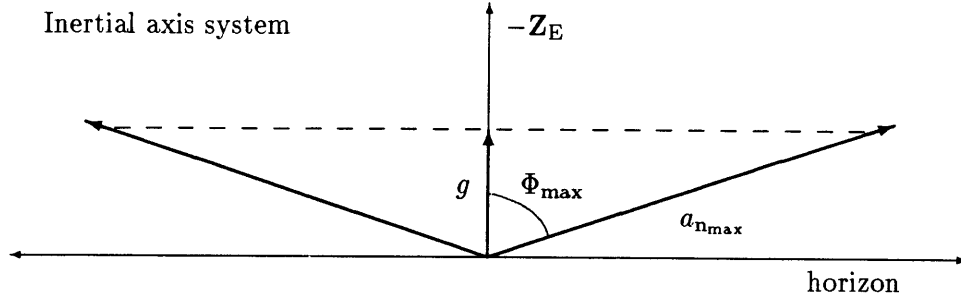


Figure 2.4: Bound on roll angle

### 2.3 Threat Function and Trajectory Risk

The threat function  $T(x, y)$  is a measure of the threat to which the aircraft is exposed when its position is  $(x, y)$ . The nature of the threat function is defined by the actual threat that it represents. A radar installation can be shielded by a mountain and will result in a threat that is a function of the distance as well as direction of the aircraft from the threat. The threat data may also be available as tabulated numerical values in the MPS. The following analytical form of the threat function is adopted for convenience:

$$T(x, y) = \sum_{i=1}^n \frac{\Gamma_i}{\sqrt{(x - x_i)^2 + (y - y_i)^2}} > 0 \quad (2.18)$$

where  $(x_i, y_i)$  and  $\Gamma_i$  is the position and intensity of the  $i$ th threat respectively.

The risk  $J$  associated with an arbitrary trajectory  $\zeta$  is defined as the integral of the threat function with respect to distance along the trajectory:

$$J = \int_{\zeta} T(x, y) ds \quad (2.19)$$

The optimal trajectory has the property that it minimizes the associated risk subject to the constraints defined in Section 2.2.

For a particular trajectory segment over the interval  $[t_i, t_f]$  the risk can be written as:

$$J = \int_{t_i}^{t_f} T[x(t), y(t)] \frac{ds}{dt} dt \quad (2.20)$$

where:

$$\frac{ds}{dt} = \left[ \left( \frac{dx}{dt} \right)^2 + \left( \frac{dy}{dt} \right)^2 \right]^{\frac{1}{2}} \quad (2.21)$$

In Section 2.2 it was assumed that the aircraft is flying at a nominal speed which is constant. Equation 2.20 can thus be simplified to:

$$J = V_{\text{nom}} \int_{t_i}^{t_f} T[x(t), y(t)] dt \quad (2.22)$$

Some threat distributions may require the trajectory to reach the second (or final) waypoint in less time than is allowed. Because the modeling of the control commands is done over a fixed interval  $[t_i, t_i + \Delta]$  there will be a segment of trajectory that goes beyond the final waypoint. This segment is superfluous and is ignored by defining the terminal time  $t_f$  to be the time when the trajectory reaches its point closest to the final waypoint.

The risk is used in the genetic algorithm to evaluate and compare the fitness of different population members. Because the optimal trajectory has a minimum risk, Equation 2.19 can not be used directly as a fitness function in the genetic algorithm which is a maximization algorithm. The necessary modifications that are made to the risk function to accommodate the maximization are discussed in Section 4.2.1.

## Chapter 3

# The Genetic Algorithm

### 3.1 Introduction

Genetic algorithms are random search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured, yet stochastic, information exchange to form a search procedure that has been theoretically and empirically proven to provide robust search in complex spaces. To achieve this robustness, genetic algorithms differ from normal optimization and search procedures in the following ways:

- genetic algorithms work with a coding of the parameter set and not the parameters themselves,
- genetic algorithms search from a population of points and not a single point,
- genetic algorithms use objective information and not derivatives or other auxiliary information,
- genetic algorithms use stochastic operators and not deterministic rules.

Genetic algorithms require that the natural parameter set be coded into a finite-length string over a finite alphabet. By exploiting the underlying similarities of the coding, genetic algorithms are largely unconstrained by the limitations (such as continuity, existence of derivatives etc.) of more traditional optimization procedures.

The robustness of genetic algorithms is largely obtained from the existence of a population of possible solutions. Instead of searching point to point, genetic algorithms work from a large database of points, which allows it to perform a number of searches in parallel and thus reducing the possibility of converging on a local optimum.

Most of the traditional search techniques are gradient based and as such require derivatives to be able to climb peaks in the search space. Genetic algorithms only make use of objective functions to determine the fitness of a particular solution. This characteristic permits genetic algorithms to search effectively in more complex spaces where the gradient based search would have broken down. Constraints can also be incorporated as an integral part of the optimization procedure because genetic algorithms are not limited by the discontinuities in the search space.

It is important to note that although the transition rules of genetic algorithms are stochastic, a distinction exists between the randomized operators of genetic algorithms and other methods that constitute random search. Genetic algorithms use random choice to guide a directed search. A comparison of the effectiveness of robust genetic algorithm search and other search methods over different problem types is shown in Figure 3.1.

Because genetic algorithms are rooted in both natural genetics and computer science, the genetic algorithm terminology is a mixture of artificial and biological terms. A short summary of the corresponding terms is given in Table 3.1. This thesis uses both sets of terminology, giving preference to the phrase that is more descriptive in the context that it is used.

Section 3.2 describes the basic cycle of a genetic algorithm and the different genetic operators which are used in each iteration. The derivation of *the fundamental theorem of genetic algorithms*, which is the mathematical foundation of the convergence properties of genetic algorithms is done in Section 3.3.



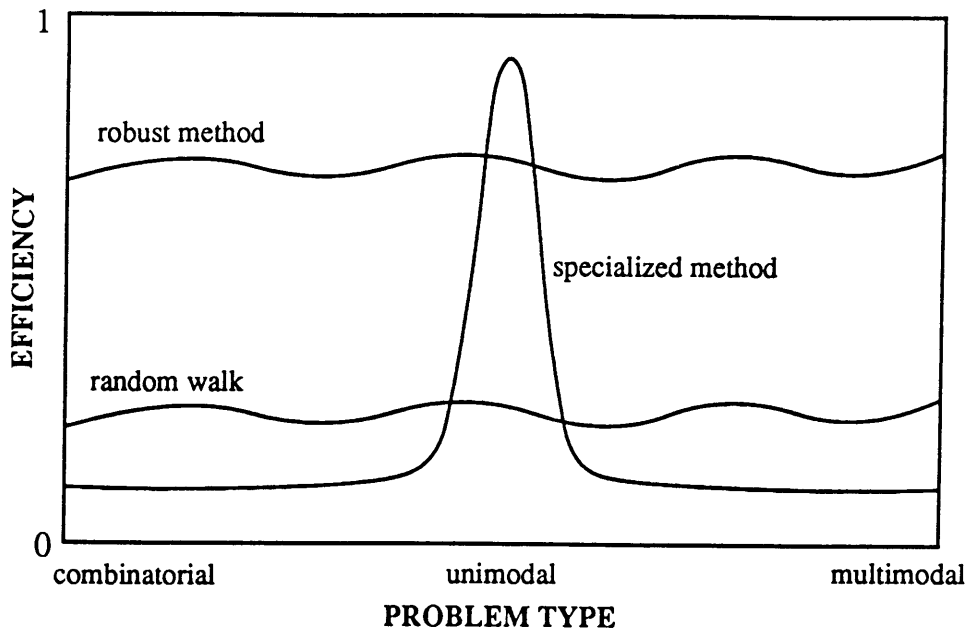


Figure 3.1: The effectiveness of different search algorithms over different problem types.

### 3.2 Basic Genetic Algorithm

The mechanics of genetic algorithms are very simple and consist of nothing more than copying strings and swapping partial strings. Each string is a coding of the parameter set of a possible solution in the defined search space. The initial population of strings is chosen at random in order to have the maximum genetic diversity. From this parent population successive generations have to be evolved that improve over time. This is achieved by using the following three basic operators:

- reproduction
- crossover
- mutation

Natural	Artificial
chromosome	substring
gene	feature or character
allele	feature value
locus	string position
genotype	string
phenotype	parameter set, decoded structure
epistasis	nonlinearity

Table 3.1: Comparison of natural and artificial genetic algorithm terminology.

Reproduction is a process in which individual genotypes are copied to a mating pool according to their objective function values. The objective function value (or fitness) of a genotype is a measure of the characteristic or collection of characteristics that has to be maximized. The larger the fitness of a particular genotype, the larger the probability that it will be selected to reproduce in the mating pool. Genotypes with a very large fitness relative to the average fitness may be selected more than once. Selection of genotypes is done until the mating pool has  $n$  members (or parents). The mating pool of high-fitness genotypes serves as a transitional population for further genetic operator actions to create the next generation. A detailed description of the different phases of reproduction is given in Section 4.2.

Crossover is the artificial manifestation of the natural swapping of genetic code. A pair of strings is selected at random from the mating pool to act as parent strings. These parent strings then interchange characters between positions  $k + 1$  and  $l$  inclusively, where  $l$  is the string length and  $k$  is chosen at random such that  $1 \leq k < l$ . The resulting two offspring strings are part of a new generation. Figure 3.3 shows the crossover operator acting on a pair of parent strings.

In applying reproduction and crossover it occasionally occurs that a useful piece of genetic material (a piece of code at a specific locus) is lost. Genetic algorithms guard against this irrecoverable loss by employing the mutation operator. Mutation is the change of value of a random string position of each of the new offspring strings. Although regarded by many as a background operator, recent research [8] suggests that mutation has a much stronger role. Figure 3.4 shows two strings before and

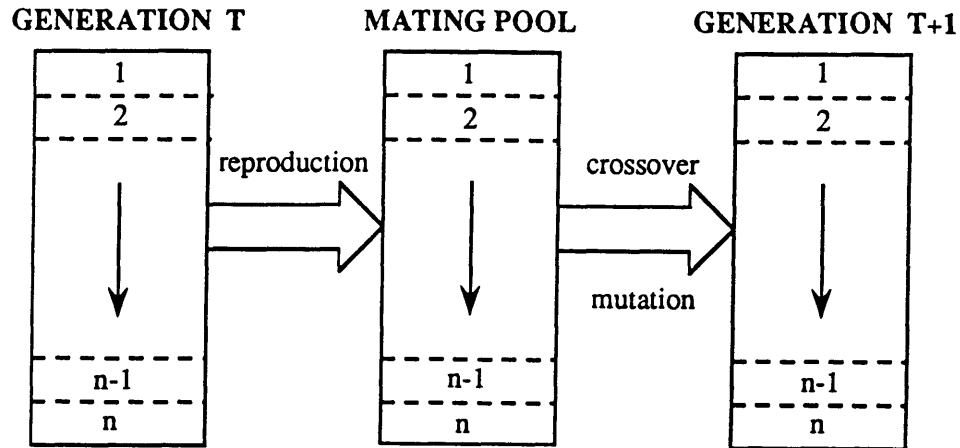


Figure 3.2: The basic genetic algorithm cycle.

after mutation occurring with probability  $p_m$ .

The study of the biological example has given rise to other genetic operators and productive schemes. However, reproduction, crossover and mutation have proven to be computationally simple and effective in a number of optimization problems.

### 3.3 Mathematical Foundation of Genetic Algorithm

#### 3.3.1 Schemata - An Introduction

Although genetic algorithms are based on natural genetics and therefore have an intuitive appeal, it is necessary to provide a mathematical background to support its mechanics. The effect of reproduction, crossover and mutation on the information contained in a population is investigated in a more rigorous way using schemata which exploit the underlying similarities between the strings of the population.

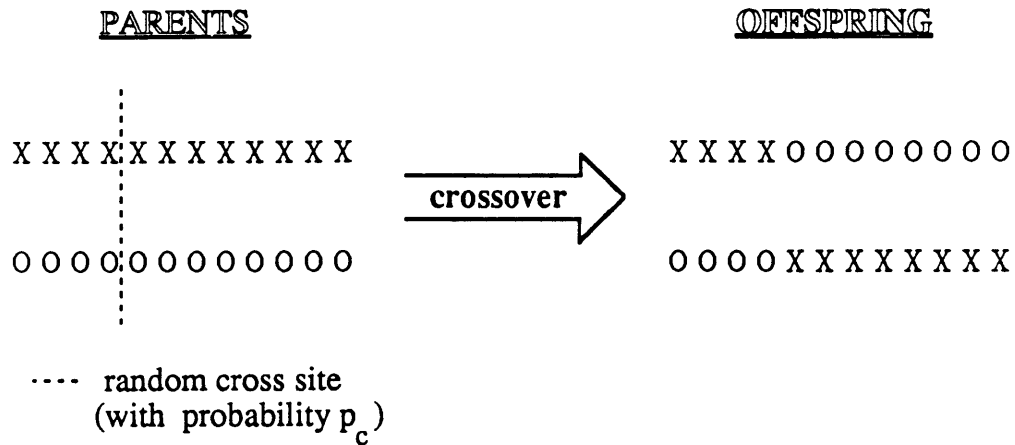


Figure 3.3: The crossover operator

A schema is a similarity template describing subsets of strings with similarities at specific loci. It can also be thought of as a hyper-plane in the search space. Using the binary alphabet  $\{0,1\}$ , define a new element  $*$  which functions as a wildcard. Schemata are then created over the extended alphabet  $\{0,1,*\}$  as a pattern matching mechanism where  $*$  can be used to match either 1 or 0. A schema matches a particular string if at every location 1 in the schema fits 1 in the string, 0 matches 0, or  $*$  matches either. An example of schemata and sets of matching strings is shown below.

schema	100*	1*0*	10**
sets of	1000	1000	1000
matching	1001	1001	1001
strings		1100	1010
		1101	1011

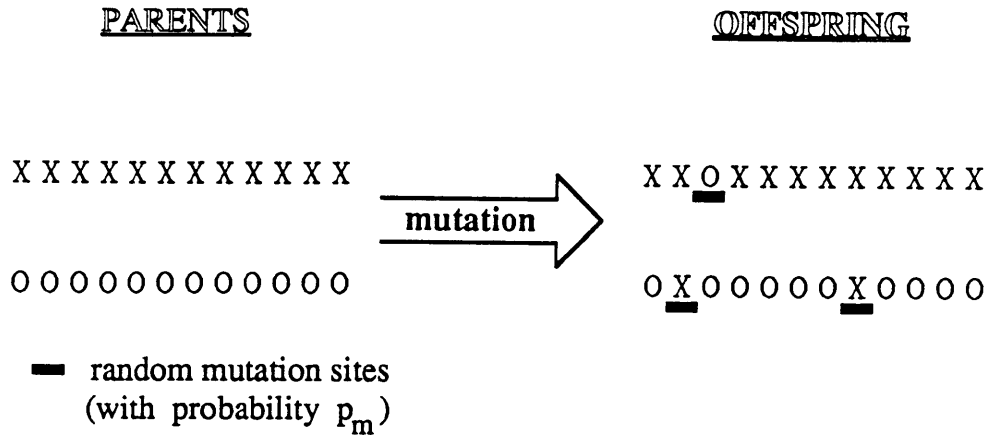


Figure 3.4: The mutation operator

For alphabets of cardinality  $k$  and string length  $l$  there are  $(k + 1)^l$  schemata. Because each string position can take its own value or the wildcard symbol, each string is a match to  $k^l$  schemata. A population of  $n$  strings from the binary alphabet will therefore display between  $2^l$  and  $2^l n$  schemata, depending on the diversity of the members of the population. For a population of  $n$  members an estimated  $O(n^3)$  schemata (or hyper planes) are processed. Thus, for each generation requiring  $n$  fitness evaluations,  $n^3$  schemata are processed in parallel without memory other than the population itself. This important feature of genetic algorithms is known as *implicit parallelism*.

Two definitive properties which distinguish between schemata are the schema order and schema defining length. The order  $o(H)$  of a schema  $H$  is the number of specific bits (1's and 0's) in the template. For example, if  $H$  is given by  $*1**10*$ , then  $o(H) = 3$ . The defining length  $\delta(H)$  of a schema  $H$  is the distance between

the first and last specific bit positions. For the same  $H$  as above  $\delta(H) = 4$ . In the case where  $o(H) = 1$  the defining length is by definition  $\delta(H) = 0$ .

### 3.3.2 The Fundamental Theorem

Schemata provide the means to analyze the effects of the genetic operators on the information contained in a population. The effect of reproduction on the expected number of schemata is determined by treating each schema as a random variable with mean estimated by the average fitness of its occurrences in the population. During reproduction a string  $A_i$  is copied according to its fitness  $F_i$ , by being selected to the mating pool with probability  $p_i = F_i / \sum F_j$ . The expected number of copies of the string  $A_i$  in the mating pool is then given by  $np_i$ . The growth or decay of a particular schema can be described in a similar manner. Define  $m(H, t)$  to be the number of instances of a particular schema  $H$  contained in the population  $\mathbf{A}(t)$  at time step  $t$ . The expected occurrences of  $H$  in  $\mathbf{A}(t+1)$  is then given by the equation:

$$m(H, t + 1) = n \frac{m(H, t) f(H)}{\sum F_j} \quad (3.1)$$

where  $f(H)$  is the average fitness of the strings representing schema  $H$  at time  $t$ . Recognizing that the average fitness of the entire population  $F_{\text{avg}} = \sum F_j / n$ , Equation 3.1 can be reduced to:

$$m(H, t + 1) = m(H, t) \frac{f(H)}{F_{\text{avg}}} \quad (3.2)$$

which is the *reproductive schema growth equation*. From this equation, the number of above-average schemata in the population will grow and below-average schemata will become less. If  $f(H)$  remains consistently above the average fitness of the population, such that  $f(H) = (1 + b)F_{\text{avg}}$ , where  $b$  is a constant, then Equation 3.2 becomes:

$$m(H, t + 1) = (1 + b)m(H, t) \quad (3.3)$$

which is the discrete equivalent of the exponential form. Note that this does not account for the effect of crossover and mutation. Reproduction thus yields an exponentially increasing (decreasing) number of above- (below-) average schemata in progressive populations.

Reproduction is a highly exploitive search of the search space. If reproduction was the only genetic algorithm operator employed, successive populations would consist only of increasing numbers of above average strings. Crossover allows for a structured information exchange between strings, which creates new strings and thus promotes exploration of the search space. Reproduction and crossover are opposing forces in that reproduction tends to increase the number of above-average schemata while crossover destroys schemata. If crossover between two strings occurs at random with probability  $p_c$ , then the lower bound on the probability that a schema survives crossover can be given by:

$$p_s \geq 1 - p_c \frac{\delta(H)}{l-1} \quad (3.4)$$

Combining reproduction and crossover, the expected occurrences of schema  $H$  in the population  $\mathbf{A}(t+1)$  can be given by:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{F_{\text{avg}}} \left[ 1 - p_c \frac{\delta(H)}{l-1} \right] \quad (3.5)$$

where the two operators are assumed to be independent. The significant terms in Equation 3.5 are  $f(H)$  and  $\delta(H)$ . Effectively, schemata that have above-average performance and short defining lengths will increase exponentially in successive populations.

Mutation has the same tendency to destroy schemata as crossover and allows for even more exploration of the search space. For a schema to survive mutation all the specified bits (there are  $o(H)$  of them) have to survive mutation. The probability that a schema survives mutation is then  $(1-p_m)^{o(H)}$  where  $p_m$  is the probability that a bit will mutate. For small values of  $p_m$  this expression can be approximated by  $1 - o(H)p_m$ . Combining reproduction, crossover and mutation the expected number of copies that a particular schema  $H$  will have in the next generation is given by:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{F_{\text{avg}}} \left[ 1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m \right] \quad (3.6)$$

where the cross-products have been ignored. Equation 3.6 proves that above-average schemata of low order with short defining lengths increase exponentially in successive populations. This result is known as the *fundamental theorem of genetic algorithms*.

### 3.4 The Parallel Genetic Algorithm

For genetic algorithms there exists an optimal trade-off between the amount of genetic search that is done and the number of computations required. If the population size is too small, then the genetic algorithm will have an inefficient search because of an insufficient number of schemata in the population. Choosing the population size too large results in an inordinate amount of time required to perform all the evaluations. In the worst case the genetic algorithm can be reduced to random search if the available time is exhausted before any genetic search is performed. To increase the real-time genetic search efficiency the genetic algorithm can be modified to evolve more than one population at the same time. Crossover and mutation can also be implemented in parallel leads to a further reduction in computing time.

A well known parallel implementation of the genetic algorithm is the classic parallel genetic algorithm (PGA). A PGA consists of a group of identical nodal genetic algorithms (NGA's). Each NGA maintains a small population which is a portion of the large population and functions in the same way as a normal sequential genetic algorithm. The difference between a sequential genetic algorithm and a NGA is that the NGA communicates with its neighboring NGA once during every cycle. The communication consists of sending the best individual in the local population to each neighboring population, and receiving the best individual of each neighboring population. The PGA can thus be thought of as a sequential genetic algorithm with a very large population.

The desirable property of the PGA is that an increase in population size by the addition of another NGA increases the execution time of the PGA only slightly. The increase in time is due to increased communication overhead among the larger set of NGA's. Initial studies [6] have indicated that a PGA is a viable means of increasing the population size of a genetic algorithm and allowing more efficient real-time genetic search. The theoretical investigation of the PGA is still continuing.



## Chapter 4

# Flight Trajectory Planning using Genetic Algorithms

### 4.1 Flight Control Commands Modeling

In Chapter 3 the genetic operators were described using the binary alphabet for the chromosome codings. This choice of coding can be justified by the *principal of minimum alphabets*, which is defined by Goldberg [5]:

**The user should select the smallest alphabet that permits a natural expression of the problem.**

Seen mathematically, the binary alphabet offers the maximum number of schemata per bit of information of any coding. In addition, Goldberg also defines the *principal of meaningful building blocks*:

**The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.**

The choice of low-order schemata with short defining length is justified by the *fundamental theorem of genetic algorithms* which is proved in Section 3.3. These two principals are used as guidelines in choosing a method for modeling the flight control

commands. The modeling is done in two steps:

1. map the continuous command inputs to discrete parameters which uniquely define the commands, and
2. code the discrete parameters into finite-length binary strings, as required by the *principal of minimum alphabets*.

This research evaluates two mapping methods; the Chebyshev polynomial modeling which was used in the previous research, and a filtered piecewise constant modeling method. Both standard binary coding and Gray codes are evaluated as coding procedures.

#### 4.1.1 Chebyshev Polynomials

Walker [9] structured the command functions using a weighted sum of five Chebyshev polynomials which are orthogonal on the interval  $[t_i, t_f]$ , where  $t_i$  and  $t_f$  are the initial and final times of a particular trajectory as defined in Section 2.3. The Chebyshev polynomials are given by:

$$T_0(\lambda) = 1 \quad (4.1)$$

$$T_1(\lambda) = \lambda \quad (4.2)$$

$$T_{i+1}(\lambda) = 2\lambda T_i(\lambda) - T_{i-1}(\lambda) \quad \text{for } i \geq 1 \quad (4.3)$$

The basis of orthogonality is shifted to the trajectory interval  $[t_i, t_f]$  by the following change of variable:

$$\lambda(t) = 2 \frac{t - t_i}{t_f - t_i} - 1 \quad (4.4)$$

The roll angle command is then given by:

$$\Phi_c(t) = c_1 + c_2 T_1(t) + c_3 T_2(t) + c_4 T_3(t) + c_5 T_4(t) \quad (4.5)$$

Continuity of the commands between segments requires the boundary condition  $\Phi_c(t_i) = \Phi_0$  where  $\Phi_0$  is the value of the previous trajectory at its point closest to the first waypoint as defined in Section 2.1. From Equations 4.1–4.5 for  $t = t_i$ :

$$\Phi_c(t_i) = c_1 - c_2 + c_3 - c_4 + c_5 \quad (4.6)$$

The boundary condition therefore eliminates one degree of freedom so that the parameter set defining the roll angle command is given by:

$$\mathbf{c} = [c_2 \ c_3 \ c_4 \ c_5]^T \quad (4.7)$$

For the purpose of using genetic algorithms, the elements of the parameter set  $\mathbf{c}$  are viewed as chromosomes. A genotype is formed by appending the substrings representing the chromosomes to form one structure. The genotype is then the coding of a specific point in the search space (a flight trajectory in this case). The coding procedure for the genotype is shown in Figure 4.1.

Chebyshev coefficients	$c_2$	$c_3$	$c_4$	$c_5$
numerical values	14	8	3	7
genotype	1110	1000	0011	0111

Figure 4.1: Coding of the Chebyshev coefficients as a genotype (chromosome length  $l = 4$ ).

This method of coding the genotype has the advantage that each continuous command input can be coded into a relatively short string which means less computing time. There are however distinct disadvantages to this method. The value of each Chebyshev coefficient contributes to the shape of the command over the whole interval  $[t_i, t_f]$ . This would not be a problem if the objective function was a linear function of the command inputs. However, the command inputs are integrated through the nonlinear dynamics of the aircraft to find the trajectory which determines the objective fitness. Perturbations in the value of a Chebyshev coefficient thus result in large spatial changes in the associated trajectory. Because of the boundary conditions imposed to ensure continuity, numerical perturbations in one coefficient must also result in changes in the value of another coefficient which compounds the effect described above. Although nonlinearities in the search space do not present any difficulties to genetic algorithms, the behavior of the objective function as a result of perturbations in the coefficient values described above breaks down the notion of short, low-order, above-average building blocks.

Using a finite-dimensional model also imposes restrictions on the bandwidth of the commands. This was particularly evident when the commands were constrained

and the problem required the commands to have a boundary value for a finite time. Although the commands are constrained easily, it is not possible to maintain a constant value. An example is shown in Figure 4.2 where the problem required the commands to have boundary values for a finite time. A larger bandwidth can be obtained by increasing the order of the Chebyshev polynomials used to model the control commands. However, the gain in modeling flexibility is small compared to the increased computational expense of introducing more complex Chebyshev polynomials.

#### 4.1.2 Filtered Piecewise Constant

This method uses numerical values at discrete points in time to model the input commands. These sampled data points are held constant over an interval  $\Delta t$  and then filtered to form a continuous input command over the interval  $[t_i, t_f]$ . Continuity of commands between waypoint segments is enforced by specifying the command value for the first interval  $\Delta t$ .

For the purpose of using genetic algorithms, each sample point can be viewed as a chromosome. As in the previous method, the chromosomes are coded into finite length substrings and appended to form the genotype. This coding procedure is shown in Figure 4.3.

Comparing Figures 4.1 and 4.3, it is clear that for the same chromosome length the genotype formed by the filtered piecewise constant method is a longer string structure than the genotype formed by the Chebyshev polynomial method. This is the case even if larger chromosome lengths are used in the Chebyshev coefficient codings because of the large number of samples in the filtered piecewise constant method necessary to model the commands over the specified waypoint separation. The filtered piecewise constant method thus requires more bit manipulations, which results in less real-time algorithmic efficiency. However, significant advantages are gained from using the filtered piecewise constant method. Because each chromosome represents a constant command value over a discrete time interval, a change in the bits of the substring will only result in a change of command value over the associated time interval. The command values thus have no time dependency on

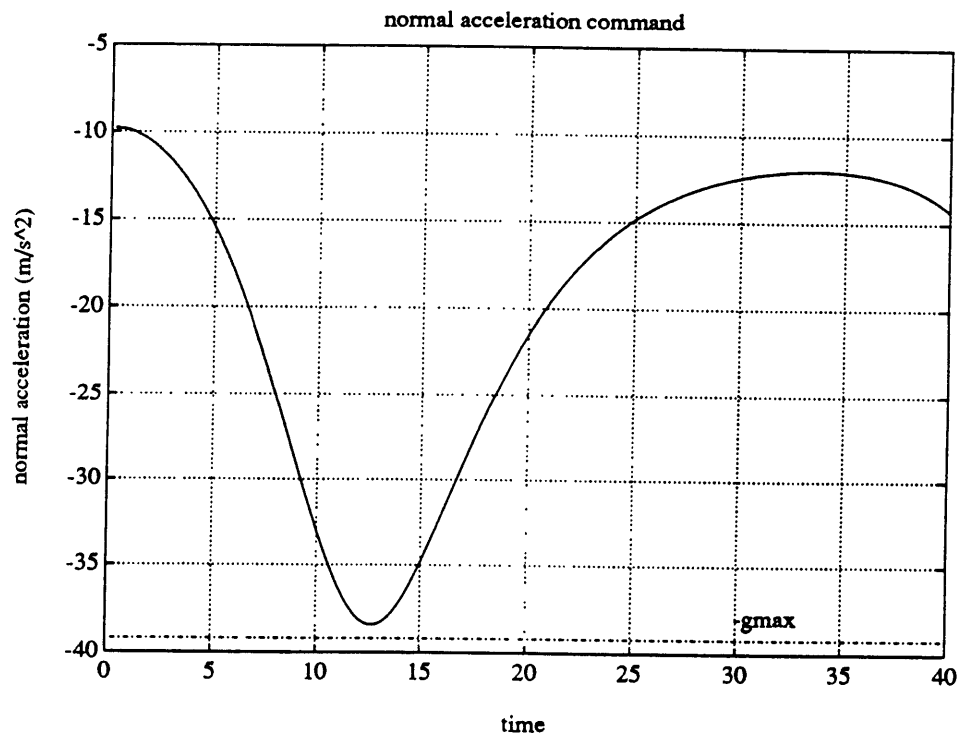
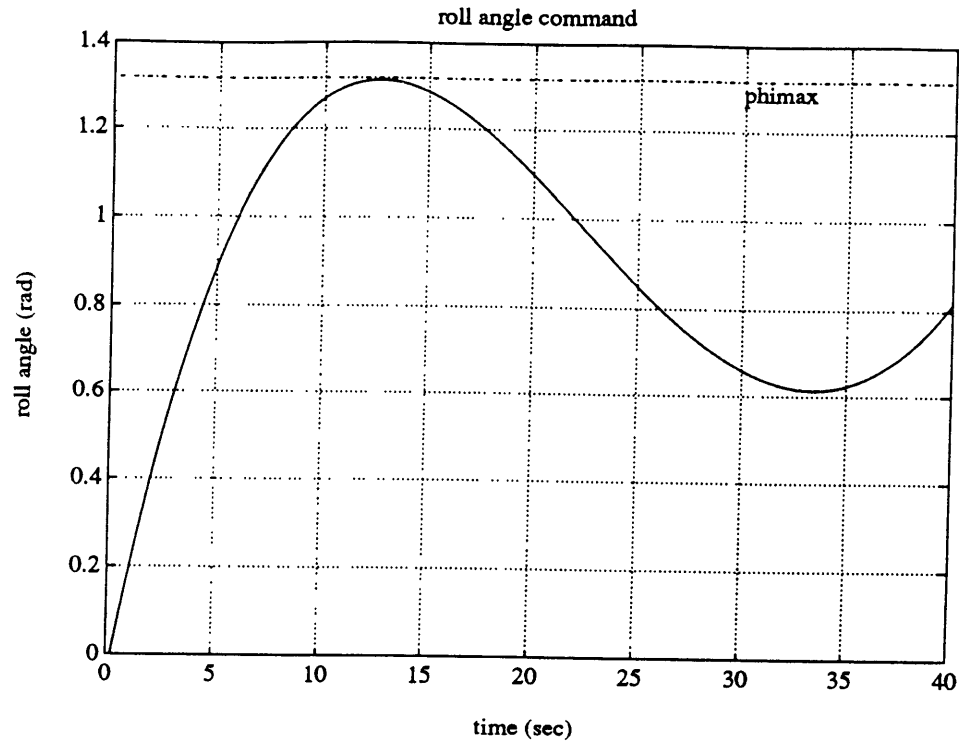


Figure 4.2: Modeling the flight control commands using Chebyshev polynomials.

time interval	$t_i + \Delta t$	$t_i + 2\Delta t$	$\dots$	$t_f - \Delta t$
command sample values	14	10	$\dots$	8
genotype	1110	1110	$\dots$	1000

Figure 4.3: Coding of filtered piecewise constant command as a genotype (chromosome length  $l = 4$ ).

the chromosomes as before. This in effect reduces the magnitude of the nonlinearities in the search space that exists between bit perturbations that were evident in the previous method.

The filtered piecewise constant method also allows the existence of short, low-order building blocks which are required for effective growth of above-average schemata. This was clearly evident from the way that genotypes evolved in a particular solution over a number of cycles. It was found that all the strings present in such a population had the same trend in bit patterns which was much more pronounced than in the case of the Chebyshev polynomial modeling method.

The sample points, which represent command values, are independent of each other, and the highest frequency component is thus only restricted by the sample frequency. More importantly, it allows the commands to hold boundary values over more than one time interval  $\Delta t$ . Figure 4.4 shows an example where it was necessary for the aircraft to hold a tight turn. The boundary values imposed on the control commands are also shown and small overshoots of these boundaries caused by the filtering are clearly evident.

It is clear that the fundamental advantages gained by using the filtered piecewise constant modeling far outweighs the computational increase required. All the subsequent experiments were done using the filtered piecewise constant modeling of the control input commands.

### 4.1.3 Gray Code vs. Binary Code

The literature [5], [3] often refers to Gray coding as achieving better results than standard binary coding. Both codings were therefore implemented to judge whether Gray coded integers would cause a performance improvement in the trajectory plan-

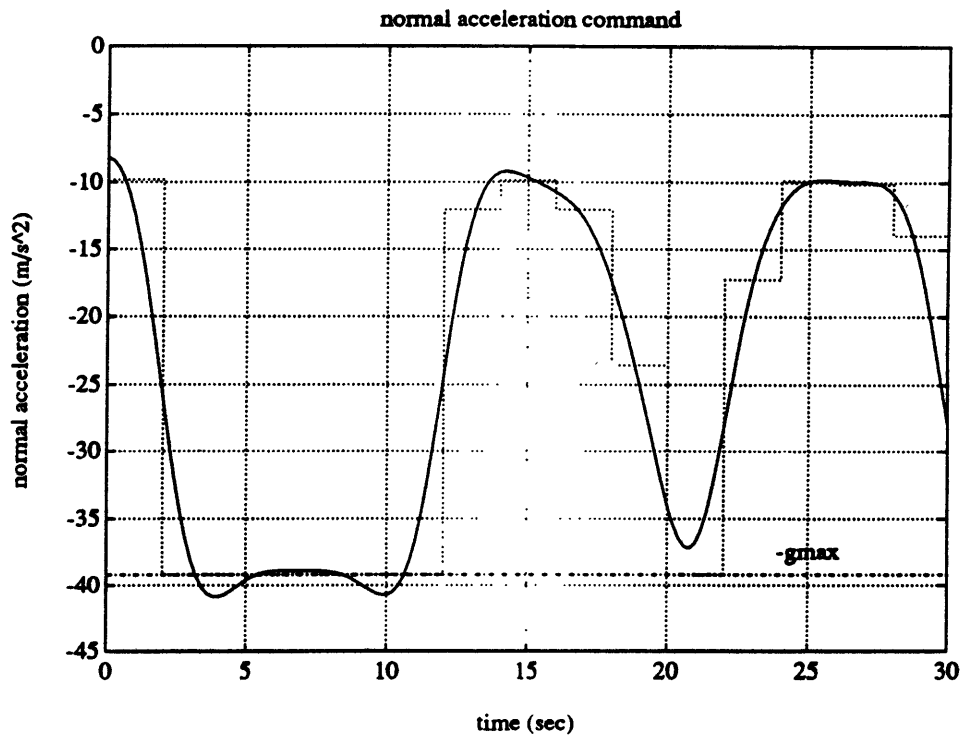
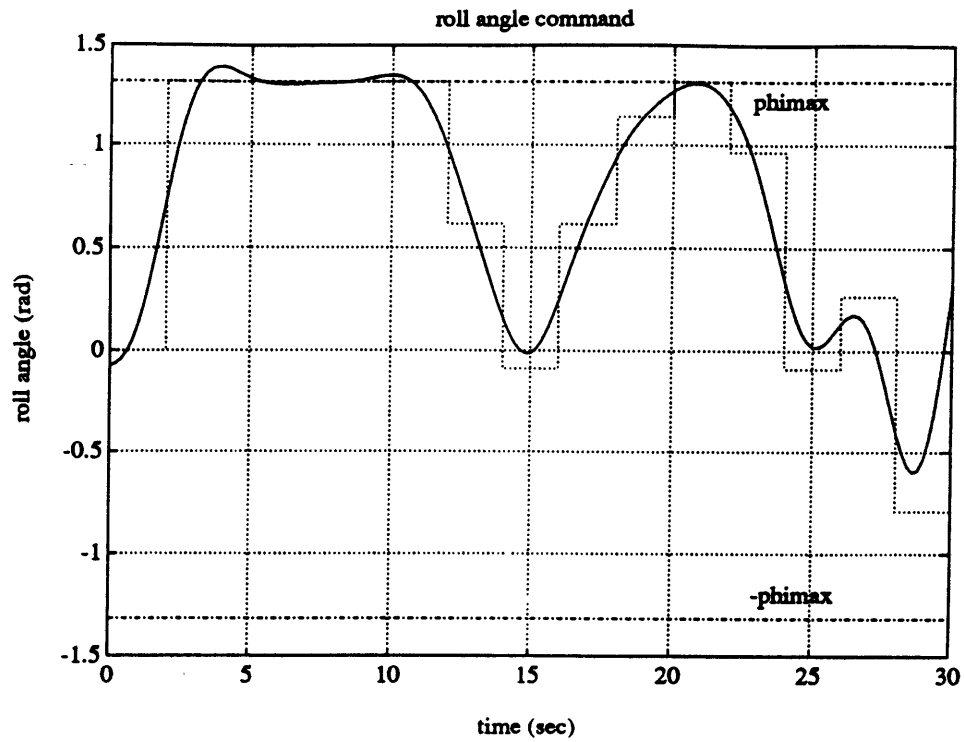


Figure 4.4: Modeling the flight control commands using filtered piecewise constant values.

ner. However, the convergence of the genetic algorithm was found to be significantly poorer when Gray coded integers were used. This is a result of the Hamming distance of 1 between adjacent integers which is characteristic of Gray codes. A table of 3-bit Gray codes is shown in Table 4.1 where the difference of only 1 bit between adjacent integers is clearly visible. This property makes Gray codes preferable in some cases where bit mutations cause small perturbations and a more thorough search. In the case of the trajectory planner however, the small perturbations caused by the bit mutations are integrated through the aircraft dynamics and the effect on the associated trajectory is negligible. Using Gray codes in the case of the trajectory planner thus almost completely neutralizes the mutation operator.

integer	binary code	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Table 4.1: Comparison between Gray coded and binary coded integers.

#### 4.1.4 Initialization of Population

The initial population used in genetic algorithm optimization is chosen at random in order to achieve the maximum diversity of information. The least complicated way of obtaining a random population is by creating a random binary population of strings (genotypes) with bit probabilities  $p\{1\} = p\{0\} = 0.5$ . If the roll angle commands are modeled using the filtered piecewise constant method, then we can assume  $E\{\Phi_c\} = 0$ . In fact, for sufficiently large strings, the roll angle commands will be similar to sampled white noise and have a flat power spectral density over the interval  $[0, w_s/2]$ . This implies that the roll angle commands will have high



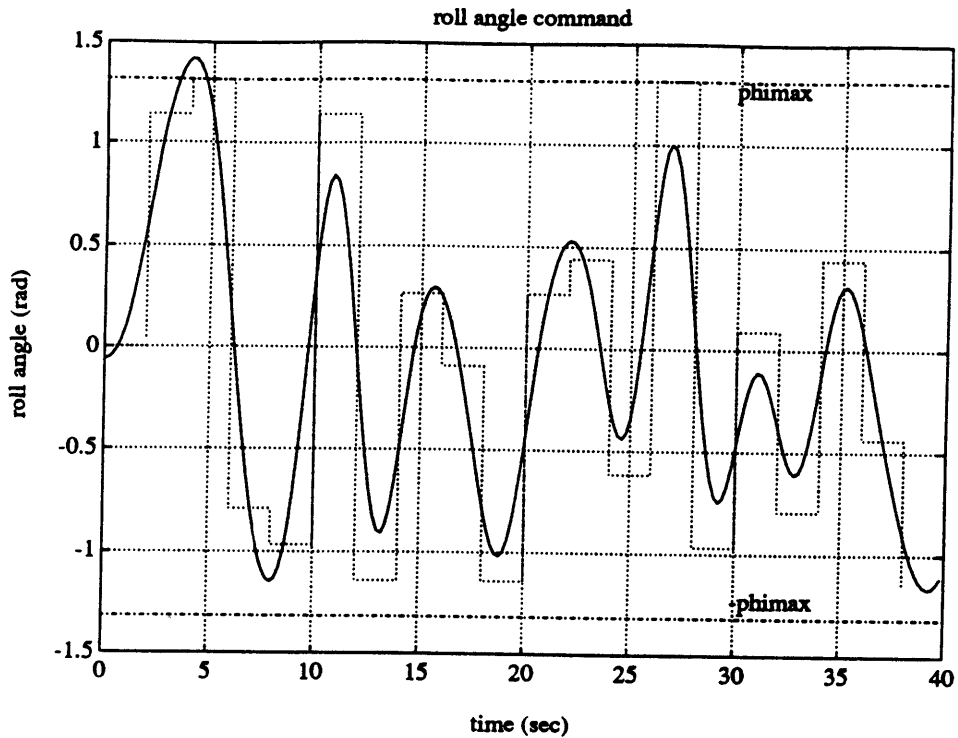


Figure 4.5: Typical member of initial population obtained using the random filtered piecewise constant method.

frequency components. Because of the relation between the roll angle and normal acceleration commands given in Equation 2.17, the high frequencies will also be visible in the normal acceleration commands. Although the aircraft dynamics will low pass filter the control commands to obtain trajectories that might even seem like good solutions, the high frequencies are both undesirable and unrealistic. Furthermore, it will lead to an initial population which is not random in the trajectory search space. This method was implemented and led to very bad performance as predicted. Not only did the random filtered piecewise constant initial population lack the crucial diversity of information required, but the genetic algorithm found it difficult to eliminate the high frequency components in the control commands. The high frequency components are clearly visible in Figure 4.5 which shows a typical member of the initial population of roll angle commands.

To obtain better performance, *a priori* knowledge of the dynamics of the aircraft is used to obtain an initial population that represents desirable control commands

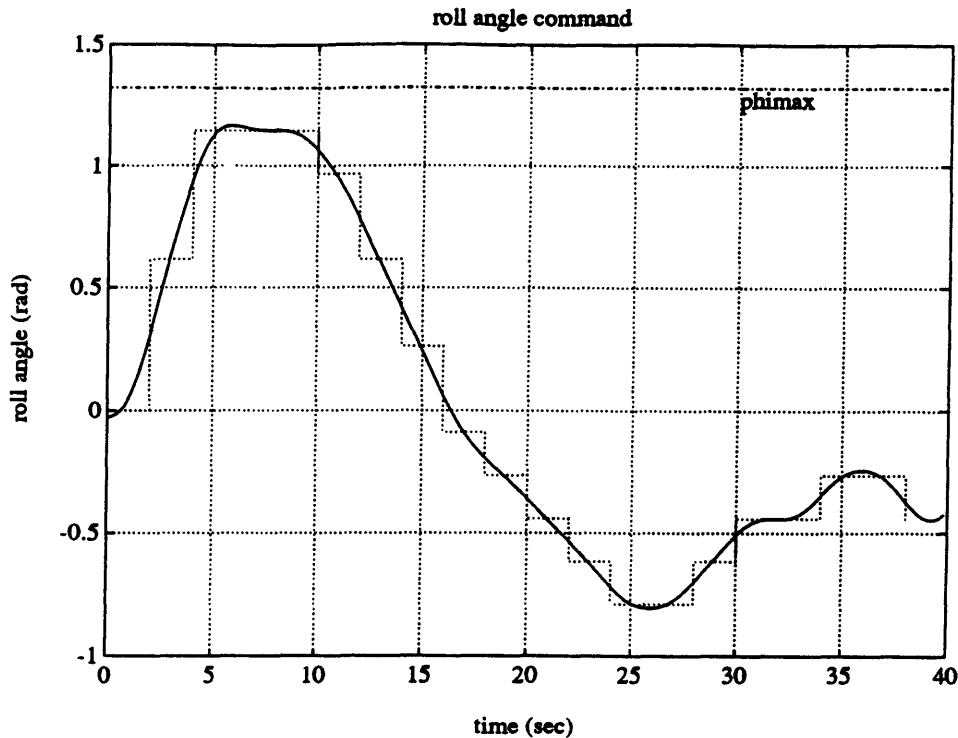


Figure 4.6: Typical member of initial population obtained using the Chebyshev polynomial and sampling method.

which integrate to a diversity of trajectories in the search space. This is achieved by making use of the Chebyshev polynomial mapping and sampling of the resulting initial population. The initial binary population is decoded into a parameter set of random Chebyshev coefficients. These random coefficients represent an initial population of roll angle commands that do not have any of the statistical characteristics described above and depict a more desirable low frequency behavior. The filtered piecewise constant modeling of the initial population is then obtained by sampling the roll angle commands and coding the samples. Figure 4.6 shows a typical roll angle command of the initial population obtained in this manner. The Chebyshev polynomial and sampling initialization method gave far better results than the random filtered piecewise constant method, and was implemented in the flight trajectory planner.

## 4.2 Reproduction

The action of the reproduction operator described in Section 3.2 can be divided into two steps:

1. evaluate and allocate a fitness value for each individual of the generation (fitness allocation), and
2. create a mating pool consistent with the expected value associated with each individual (selection).

Both these steps have been studied extensively [1], [2] and their effect on the performance of genetic algorithms is well known.

### 4.2.1 Fitness Function

For the control commands optimization problem, the flight trajectories are evaluated according to the risk associated with each trajectory where the risk is defined by Equation 2.22. Because genetic algorithms do a maximization of a nonnegative fitness function, it is necessary to map the natural objective function (risk  $J_i$ ) to a dual fitness function form. The following risk-to-fitness transformation is used:

$$F_i = \begin{cases} J_{\text{ceil}} - J_i & \text{when } J_i < J_{\text{ceil}}, \\ 0 & \text{otherwise} \end{cases} \quad (4.8)$$

where  $J_{\text{ceil}}$  is a constant.

At the start of a genetic algorithm optimization it is common to have a population with a few extraordinary members. Because the selection method is based upon the relative fitness of individuals, these highly-fit members will dominate the consecutive populations and lead to premature convergence which is undesirable. The opposite happens after a few cycles have passed, where most of the members have reached a high fitness and the relative fitness values are about the same for both average and more optimal members. Instead of a survival-of-the-fittest improvement in subsequent populations, this results in a random walk among the mediocre individuals. Using linear fitness scaling enhances the competition between individuals

at the beginning and end of a genetic algorithm optimization. The scaled fitness value is defined as follows;

$$F_i^* = aF_i + b \quad (4.9)$$

The constants  $a$  and  $b$  are chosen so that the average of the scaled fitness values equals the average of the raw fitness values. This ensures that the expected value of an average member remains the same after scaling. The second degree of freedom is used to choose the expected number of offspring that the fittest member will contribute to the subsequent generation (or number of occurrences in the mating pool). This is done by choosing

$$F_{\max}^* = C_{\text{mult}} F_{\text{avg}} \quad (4.10)$$

where  $C_{\text{mult}}$  is the expected number of offspring of the fittest member of the population. The scaling procedure is shown in Figure 4.7 for  $C_{\text{mult}} = 2$ . From the figure it can be seen that if there is an occurrence of a member with a relative low raw fitness, then the linear scaling may force the scaled fitness to become negative. To satisfy the nonnegative requirement on fitness values, the value of  $C_{\text{mult}}$  is then adjusted so that  $F_{\min}^* = 0$ .

The two problems described above are magnified in the control commands optimization problem because the trajectory risk is a quadratic function of distance. In the first few cycles of a genetic algorithm run where the population consists of highly diverse members, the raw fitness of each will differ to a great degree and the highly-fit members will dominate the other members. To ensure nonnegative raw fitness values (at least for the larger percentage of the population)  $J_{\text{ceil}}$  must be chosen quite large. After a few cycles, when the members are comparable, the large value of  $J_{\text{ceil}}$  will cause the relative raw fitness to be the same. Fitness scaling is thus of extreme importance for the control commands optimization problem.

#### 4.2.2 Selection

The selection phase of reproduction determines the actual number of offspring each individual will receive based on its relative performance (fitness). The selection phase is composed of two parts:

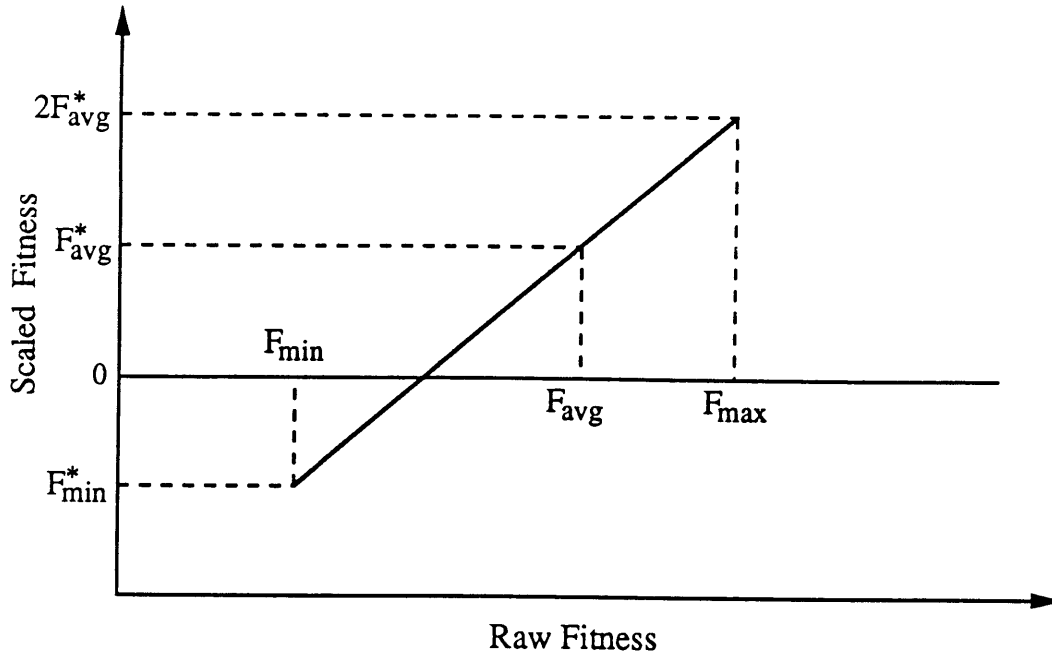


Figure 4.7: Linear fitness scaling without adjustment for negative fitness values ( $C_{\text{mult}} = 2$ ).

1. determination of the individual's expected value, and
2. formation of a mating pool based on the expected values (sampling).

The expected value of an individual is defined as  $E_i = F_i / F_{\text{avg}}$  and is an indication of the number of times that the individual should be reproduced in the next generation. If a member has an expected value of 2.5, then it must be copied in the mating pool an average of  $2\frac{1}{2}$  times.

The sampling algorithms used to convert expected values to integer number of realizations in the mating pool is evaluated according to three measures; bias, spread, and efficiency.

**bias:** Bias is defined to be the absolute difference between the expected value and the actual sampling probability of an individual. The optimal zero bias is achieved when each individual's sampling probability equals its expected value.

**spread:** The spread is defined as the range of actual realizations that an individual can achieve in a mating pool. Whereas the bias indicates the accuracy, the

spread reveals the consistency of the sampling algorithm.

**efficiency:** The efficiency of the sampling algorithm is determined by its effect on the computational complexity of the genetic algorithm.

All the sampling algorithms that are currently available fail to provide zero bias and minimum spread. There thus exists a trade-off between these measures. The remainder stochastic without replacement (RSSwoR) is the most commonly used sampling algorithm because of its efficiency and minimal spread. However, Baker [1] proved that RSSwoR exhibits severe bias which gets progressively worse as more individuals are selected to the mating pool. Remainder stochastic independent sampling (RSIS) is another sampling theorem which also has minimum spread and optimal efficiency. These sampling algorithms are implemented as follows:

1. Each individual contributes to the mating pool a number of offspring according to the integer value of its expected value. For example, an individual with expected value 2.4 will contribute 2 samples (offspring). The integer value is then subtracted from the expected value, which results in a new expected value that is a fraction. For the example, the new expected value will be 0.4. This is done for all individuals of the generation until only fractional expected values remain. The integer phase of RSSwoR and RSIS is exactly the same.
2. RSIS then independently uses each fractional expected value as a probability of further selection. This is accomplished by traversing the population and stochastically determining whether each individual should be selected. Bias occurs if a second traversal of the population is necessary to fully populate the mating pool. However, most of the samples are typically obtained in the zero bias first traversal, which leads to small overall bias. RSSwoR makes use of the error-prone spinning wheel method which sets the expected value of the individual to zero once it has been selected in the fractional phase. This prevents individuals from having multiple selections during the fractional phase and biases the sampling towards smaller fractions.

Empirical evidence by Baker also proved that RSIS exhibits an order of magnitude less bias than RSSwoR. RSIS can furthermore be partially implemented in parallel,

which is desirable for future research to enhance the real-time performance of the trajectory planner. Because of its superior performance and its parallel implementation property, RSIS is used as the sampling algorithm in the trajectory planner.

## 4.3 Constrained Optimization

### 4.3.1 Flight Control Commands

In Section 2.2 a simplification was introduced which reduced the control command set to one input command, the roll angle command. The simplification caused the roll angle command to be constrained to  $|\Phi_c| \leq \Phi_{c_{\max}}$  by the maximum normal acceleration. The constraint can easily be implemented in the filtered piecewise constant modeling method by making use of a linear mapping procedure. The mapping is done by choosing the smallest binary integer as  $\Phi_{c_{\min}}$  and the largest binary integer as  $\Phi_{c_{\max}}$ . The intermediate values are obtained by linearly mapping the remaining binary integers as shown in Figure 4.8. Using longer strings will thus result in a higher resolution.

	linear mapping between		
command value	$\Phi_{c_{\min}}$	→	$\Phi_{c_{\max}}$
4-bit binary integer	0000	...	1111

Figure 4.8: Linear mapping of roll angle between  $\Phi_{c_{\min}}$  and  $\Phi_{c_{\max}}$ .

Because of the filtering of the piecewise constant values the roll angle command can not be guaranteed to be within constraints. Figure 4.4 is a good example of where the filtering allows the roll angle command to overshoot the boundary values. These overshoots are much larger in the normal acceleration command because of the nonlinear relation with the roll angle command. The normal acceleration is therefore obtained as piecewise constant values before the filtering is done independently on each of the commands. For the purpose of this research the roll angle and normal acceleration commands are assumed to be constrained satisfactorily. It is presumed that when the trajectory planner is implemented a more effective filter will be designed that will prevent overshoot of the boundary values.

### 4.3.2 Search Space

Each waypoint has an associated nonzero capture radius which is an indication of how close to the waypoint the flight trajectory must pass. This constraint would seem to pose no problem, as trajectories which pass outside of these capture radii can be assigned a zero fitness and will not be selected to the mating pool. However, trajectories which pass very close outside the capture radii usually contain valuable genetic information that will be lost if this method is employed. To preserve the genetic information in the population, a penalty method is used which degrades the fitness in relation to the degree of constraint violation. The penalty function transforms the constrained problem to an unconstrained problem by associating a penalty with all constraint violations which is included in the objective function (fitness) evaluation. The degree of constraint violation for the  $i$ th waypoint is defined to be:

$$h_i = \frac{d_i}{r_i} \quad (4.11)$$

where  $d_i$  - closest distance between trajectory and  $i$ th waypoint

$r_i$  - capture radius of  $i$ th waypoint

The penalty function is included in the objective function defined in Equation 4.8 as follows:

$$F_i = \begin{cases} J_{\text{ceil}} - \left( J_i + \Omega \sum_{i=1}^2 P(h_i) \right) & \text{when } \left( J_i + \Omega \sum_{i=1}^2 P(h_i) \right) < J_{\text{ceil}}, \\ 0 & \text{otherwise} \end{cases} \quad (4.12)$$

where  $P$  - penalty function

$\Omega$  - penalty coefficient

A number of alternatives exist for the penalty function  $P$ . The most commonly used penalty function is the square of the degree of constraint violation  $h_i$ . The nature of the trajectory planner problem forces the use of a modified penalty function. A trajectory which passes within the circle that is specified by the capture radius around a waypoint is within bounds and should not be penalized. This property of the trajectory planner problem is realized by defining the penalty function  $P$  as



follows:

$$P(h_i) = \begin{cases} h_i^2 & \text{when } h_i > 1, \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

It was found that choosing the penalty coefficient  $\Omega$  too large results in poor performance of the genetic algorithm. This is similar to choosing the fitness of trajectories that are not within bounds to be zero, which is undesirable as mentioned above. For smaller values of  $\Omega$ , it often happens that a trajectory which is just outside the capture circle of a waypoint has a larger fitness value than a trajectory which is within bounds but passes close to a threat. To compensate for this each member of the genetic population is assigned a boolean flag which is set to indicate whether the associated trajectory is within bounds. The optimal solution at any time is therefore the population member that is within bounds and with the highest fitness. This optimal solution is kept in memory and updated each time that a solution within constraints and with a higher fitness is generated. This ensures that there are flight control commands available to the FCS for the next trajectory segment at any time after the first legitimate solution has been generated.

#### 4.4 Crossover, Mutation and Population Size

The genetic algorithm was implemented in the MPS using a simple two point crossover scheme. Two point crossover is similar to basic crossover, except that two crossover points are chosen. The two crossover points  $k_1$  and  $k_2$  are integers chosen at random such that  $1 \leq k_1 < k_2 < l$  where  $l$  is the genotype string length. The parent strings then swap the substrings between  $k_1 + 1$  and  $k_2$  inclusively. The two point crossover operator is shown in Figure 4.9. It desirable to keep the number of crossover points small, as multiple crossover with too many crossover points tends to a random shuffle of substrings between parent strings which destroys schemata. More complex crossover schemes have been developed, such as adaptive crossover, checkerboard crossover and knowledge-augmented crossover. However, these methods require additional overhead which leads to degraded real-time performance. The basic mutation operator was implemented as described in Section 3.2.

An analysis was done on the off-line performance of the flight trajectory planner

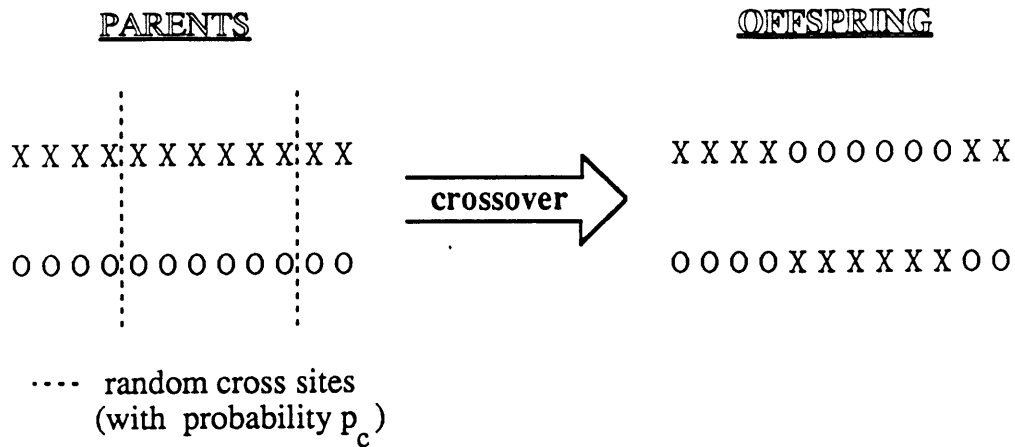


Figure 4.9: The two point crossover operator

to find the numerical values of the control variables that optimize the real-time convergence of the genetic algorithm. The off-line performance is measured using only the best legitimate member of each generation. This is in accord with the flight control commands optimization problem where only the optimal solution available to the FCS at any time is of importance. The control variables that were used in the analysis are mutation probability, crossover probability and the population size.

Due to the massive amount of computations involved it was not possible to do an extensive analysis. A CRAY super-computer was used with a sample size of 240 experiments. A total of 4 different problems was used to obtain a degree of diversity in the search space. The optimizations were run for 80 generations which was enough to allow for convergence to a suitable trajectory solution. The analysis showed a definite pattern in the off-line performance of the genetic algorithm as a function of the three parameters analyzed. The numerical results are shown in Table 4.2.

As expected, smaller crossover probabilities cause a decline in genetic algorithm performance. The small crossover probabilities lead to less information exchange between high-fit members which breaks down the core of the genetic algorithm theory, the evolution process. The off-line performance as a function of crossover

$p_m$	$p_c$	pop. size	generations			
			20	40	60	80
0.001	0.85	80	8.3094	8.3515	8.3696	8.3803
0.002	0.85	80	7.9456	8.1247	8.3015	8.3902
0.005	0.85	80	8.3605	8.3712	8.3774	8.3870
0.010	0.85	80	8.3110	8.3592	8.3763	8.4034
0.020	0.85	80	8.3592	8.3854	8.4049	8.4076
0.050	0.85	80	8.3221	8.3568	8.3617	8.3644
0.005	0.25	80	8.2188	8.2476	8.2672	8.2743
0.005	0.40	80	8.3004	8.3385	8.3581	8.3665
0.005	0.55	80	8.1940	8.3476	8.3671	8.3783
0.005	0.70	80	8.1617	8.3669	8.3822	8.4018
0.005	1.00	80	8.3292	8.3712	8.3908	8.3992
0.005	0.85	40	8.2642	8.3287	8.3451	8.3643
0.005	0.85	60	8.1990	8.3502	8.3773	8.3955
0.005	0.85	100	8.1834	8.3414	8.3863	8.4065
0.005	0.85	150	8.2519	8.2819	8.3789	8.3968

Table 4.2: Off-line performance of the genetic algorithm shown as the average maximum fitness of 240 experiments using 4 different problems.

probability is shown in Figure 4.10.

The mutation probability analysis reveals a boundary value where the performance is at a peak. Larger values of probability lead to a sharp decline in performance which is a result of the collapse of the optimization procedure as it becomes a random walk through the search space. The off-line performance as a function of mutation probability is shown in Figure 4.11.

The trade-off that exists between the speed and extent of convergence as a function of the population size is clearly visible in Figures 4.12 and 4.13. The results show that the relation of faster convergence to smaller population size breaks down once a certain boundary value is crossed. Both 60 and 100 member populations converge slower than the 80 member population.

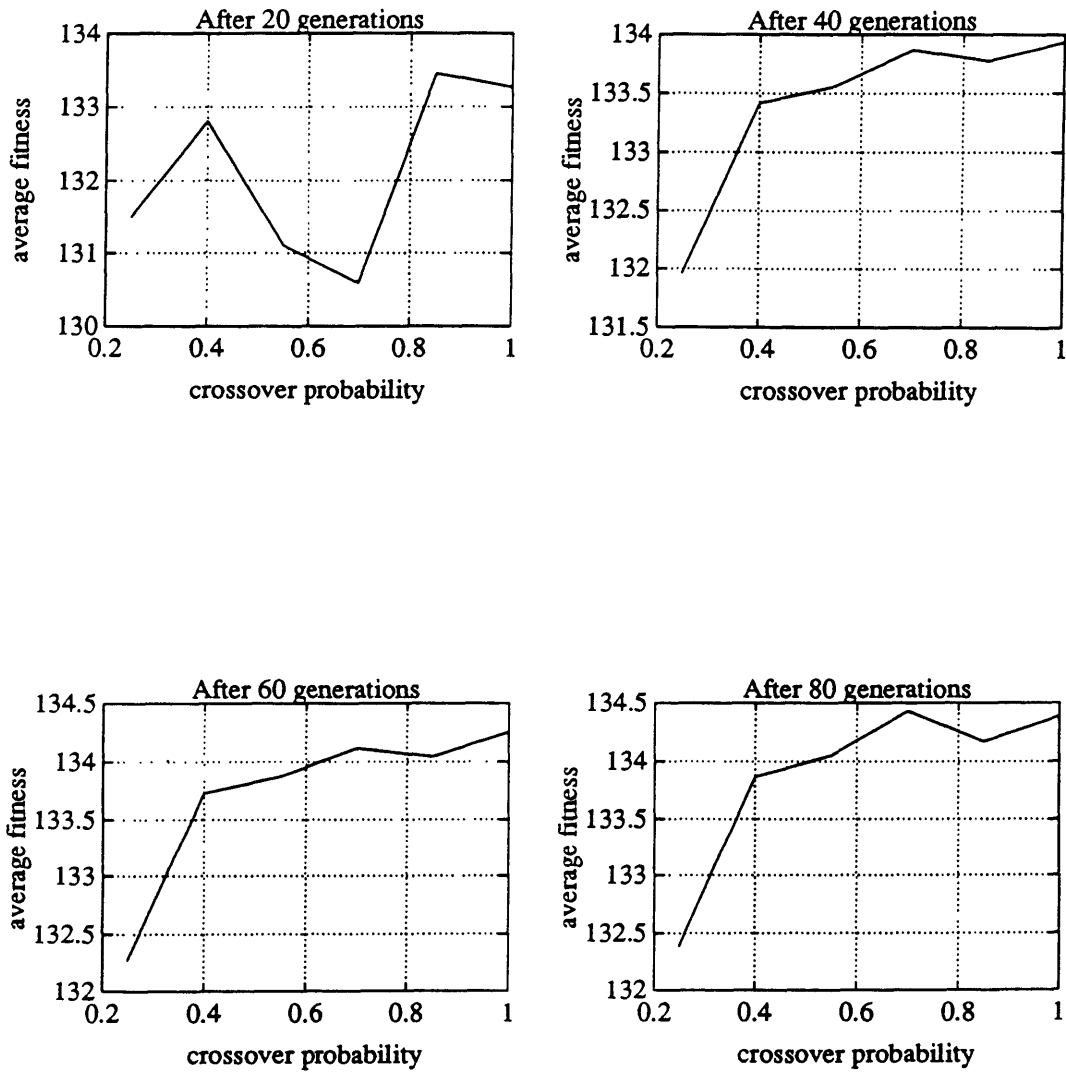


Figure 4.10: Average off-line performance of the genetic algorithm as a function of crossover probability.

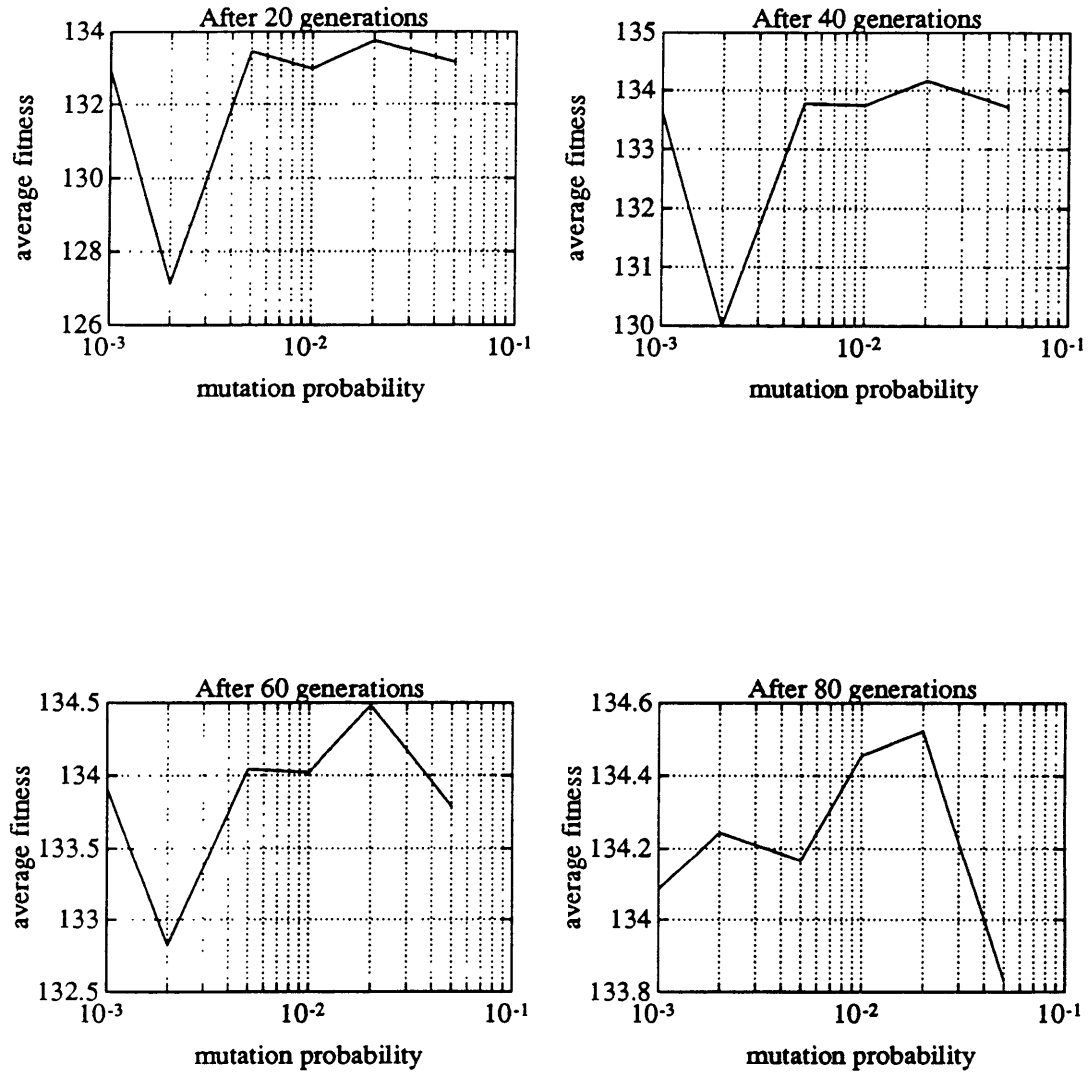


Figure 4.11: Average off-line performance of the genetic algorithm as a function of mutation probability.

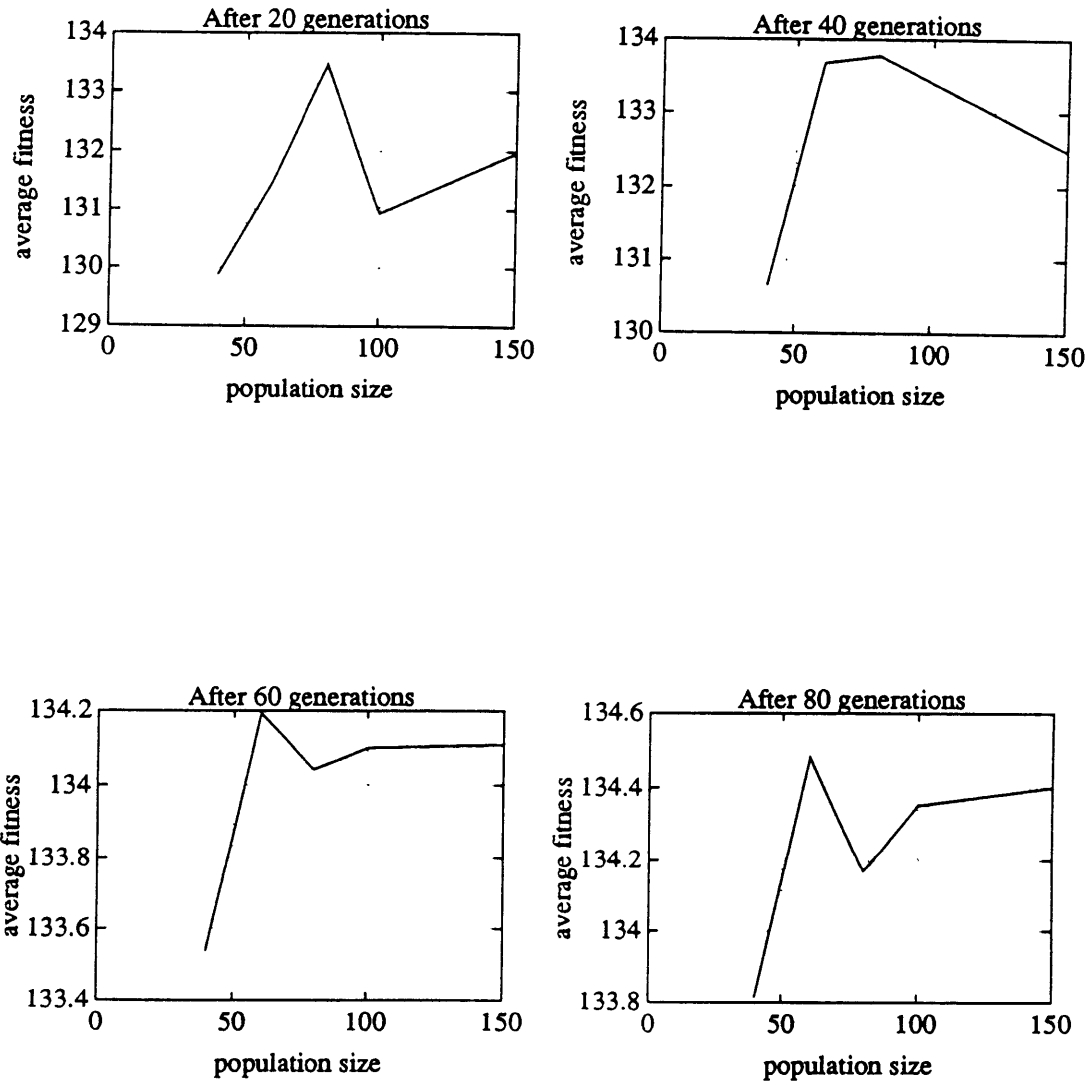


Figure 4.12: Average off-line performance of the genetic algorithm as a function of population size.

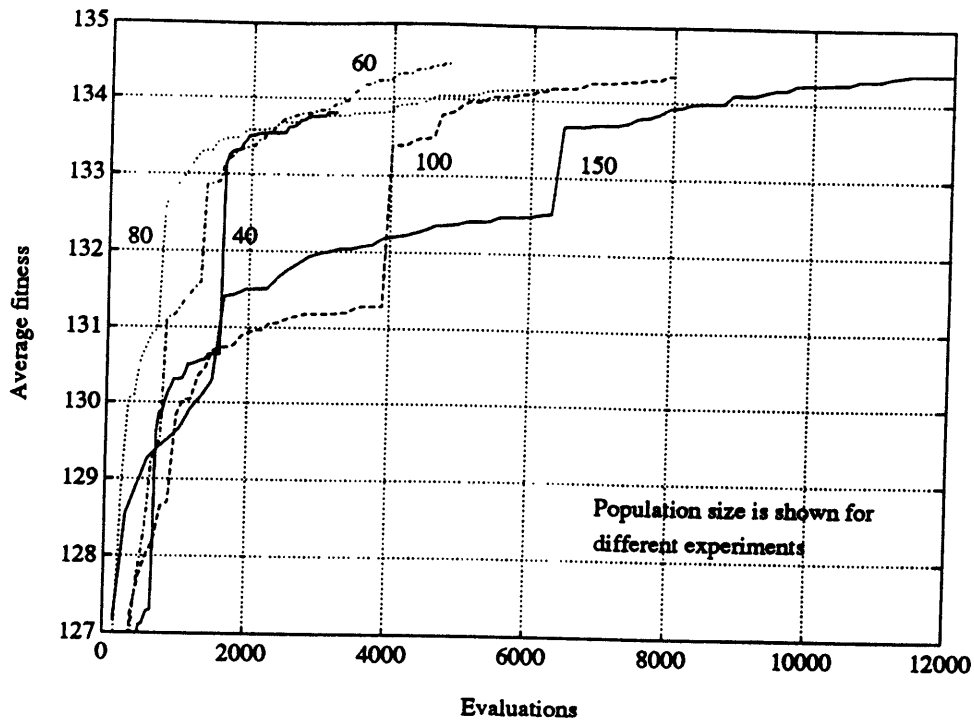


Figure 4.13: Average off-line performance of the genetic algorithm as a function of total number of trajectory evaluations.

Although the analysis did not bring to light any unexpected results, it gave a good indication of the behavior of the flight trajectory planner due to numerical changes in the different control variables. From the results of the analysis, the genetic algorithm control variables for optimal real-time performance of the flight trajectory planner were chosen as:

- crossover probability :  $p_c = 0.70$
- mutation probability :  $p_m = 0.02$
- population size :  $n = 80$

## Chapter 5

# Results

A genetic algorithm was implemented as the optimization procedure in the flight trajectory planner as described in Chapter 4. A variety of waypoint and threat distributions was chosen to represent a set of diverse realistic scenarios. The Broyden-Fletcher-Goldfarb-Shannon (BFGS) minimization routines used in the previous research converged to a constrained optimal trajectory after approximately 6000 evaluations. For the purpose of comparing the performance of the genetic algorithm to that of the BFGS routines, the genetic algorithm cycle was run for 75 generations which constitute 6000 trajectory evaluations for a population size of 80 members. The resulting flight trajectories and flight control commands are shown in Figures 5.1–5.4. The normal acceleration values are shown as negative. This results from the convention which has the positive  $Z_E$ -axis pointed downward.

Because of the simplification introduced in Section 2.2 the altitude of the aircraft in all of these cases is constant at the initial value (the flight environment was chosen for an altitude of  $200m$ ). It is important to note that the flight control commands are pushed to their limits in all the test segments. Figures 5.5 and 5.6 compares the flight trajectories and control commands for varying levels of constraint on the maximum normal acceleration. This is an indication that the flight trajectories are avoiding the threats to the full extent that the aircraft dynamics allow. In other words, the flight trajectory planner is operating correctly.

Figures 5.7 and 5.8 show how the best constrained solution evolves with passing



generations. For the 4 test segments used, the first constrained solution was available after an average of 2 genetic algorithm cycles (160 evaluations). If it was necessary, the FCS could therefore have access to flight control commands which would not be optimal in the sense of avoiding threats, but it would be within bounds, so that the aircraft would reach both waypoints without leaving its constrained dynamic range.

The optimization routines and command integrations were all programmed in MATLAB. Using a 486 IBM compatible personal computer running at a clockspeed of 33Mhz, each genetic algorithm cycle took approximately 24 seconds. On average, a constrained solution was therefore available after 48 seconds. To run for 2000 evaluations took about 9 minutes, whereas it will take the aircraft just more than 20 seconds to cover the distance to the first waypoint. Because of the binary nature of genetic algorithms it is possible to implement the population of strings as physical bits in the computer using a low-level programming language. This will result in significant improvement in the computing time required for convergence of the algorithm. With the technological advancement in computer hardware it thus is possible that the MPS will soon be able to converge to a realistic solution in the required time without considering the potential for parallel processing to speed up the execution time.

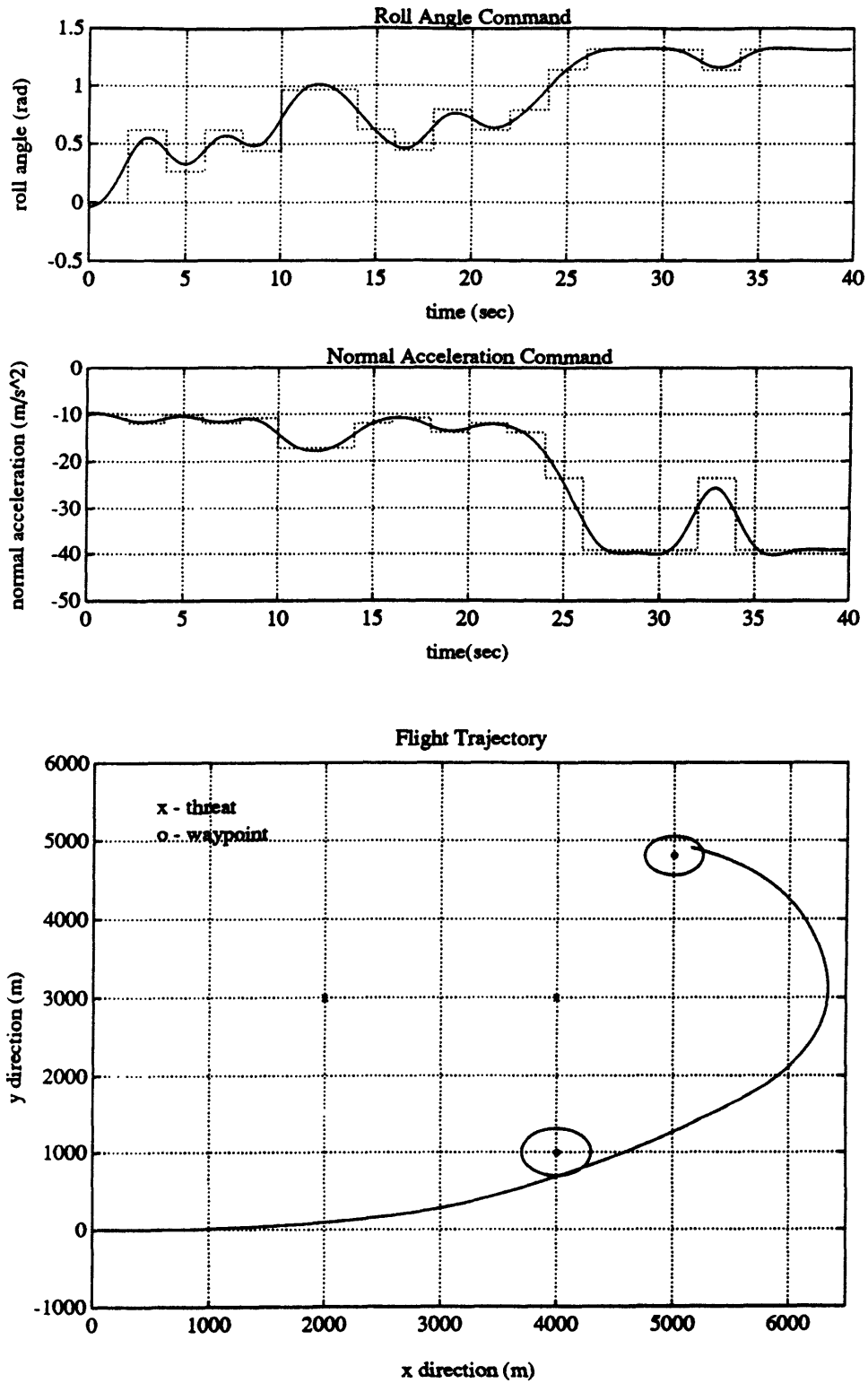


Figure 5.1: Result of genetic algorithm optimization of test segment 1.

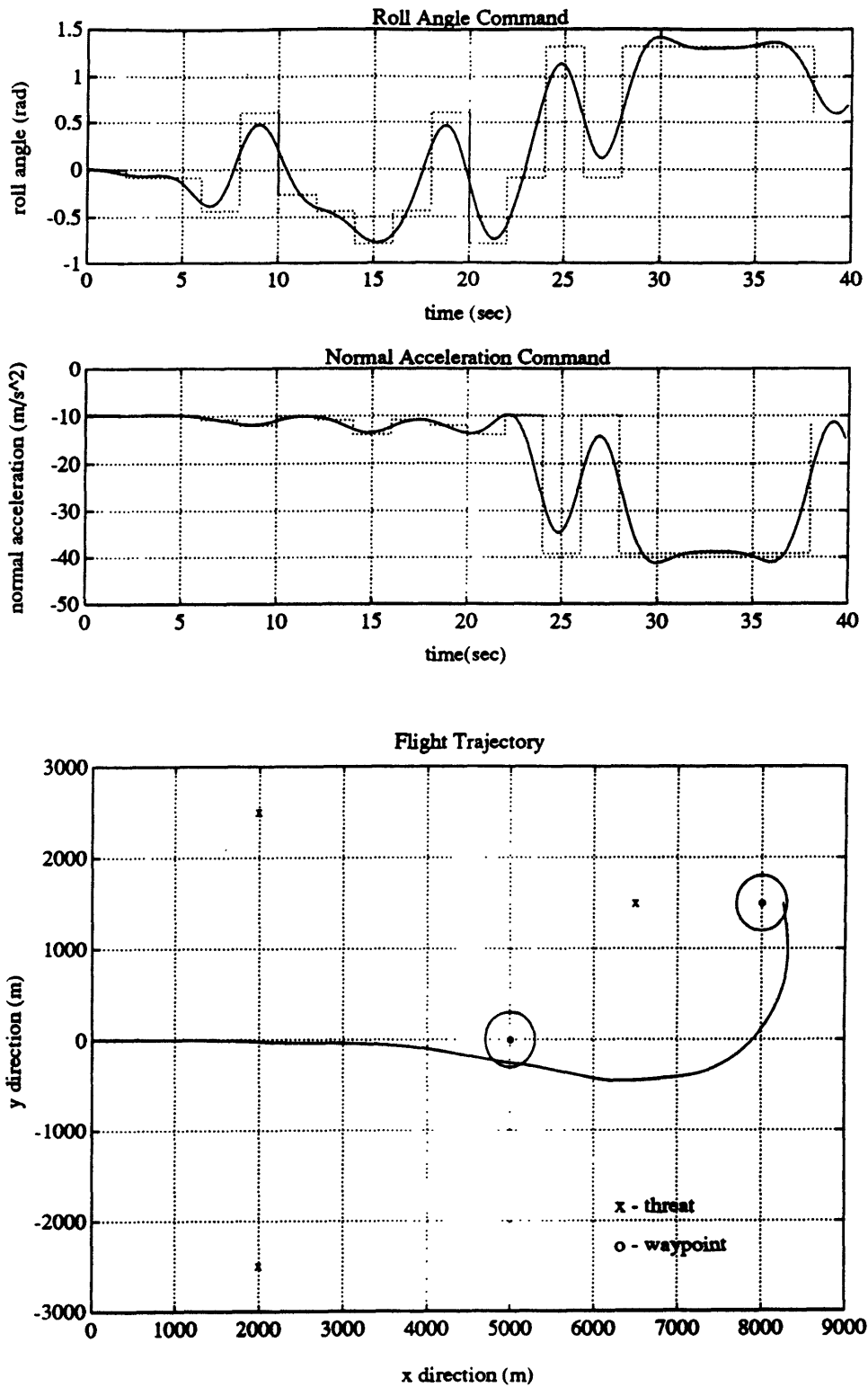


Figure 5.2: Result of genetic algorithm optimization of test segment 2.

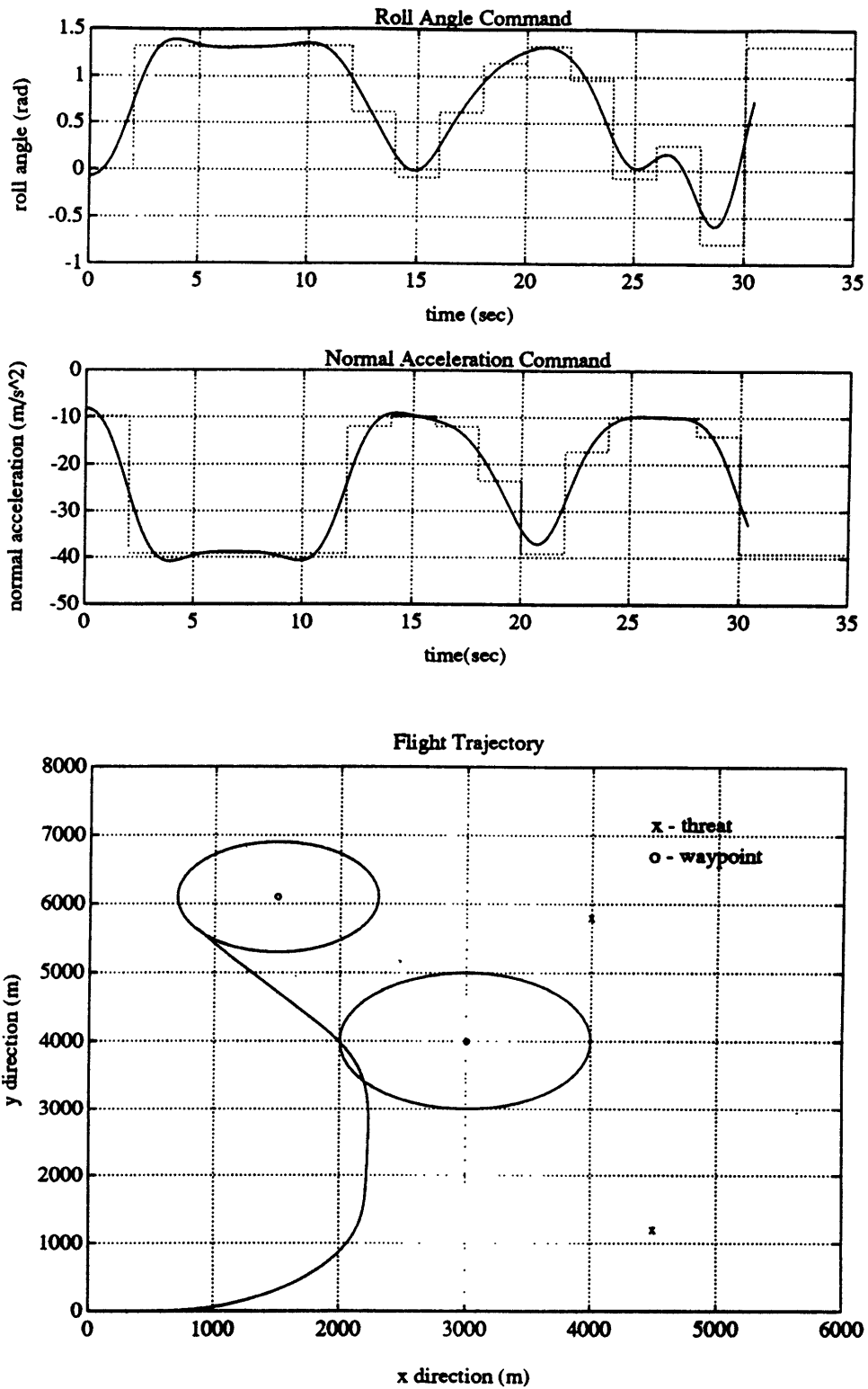


Figure 5.3: Result of genetic algorithm optimization of test segment 3.

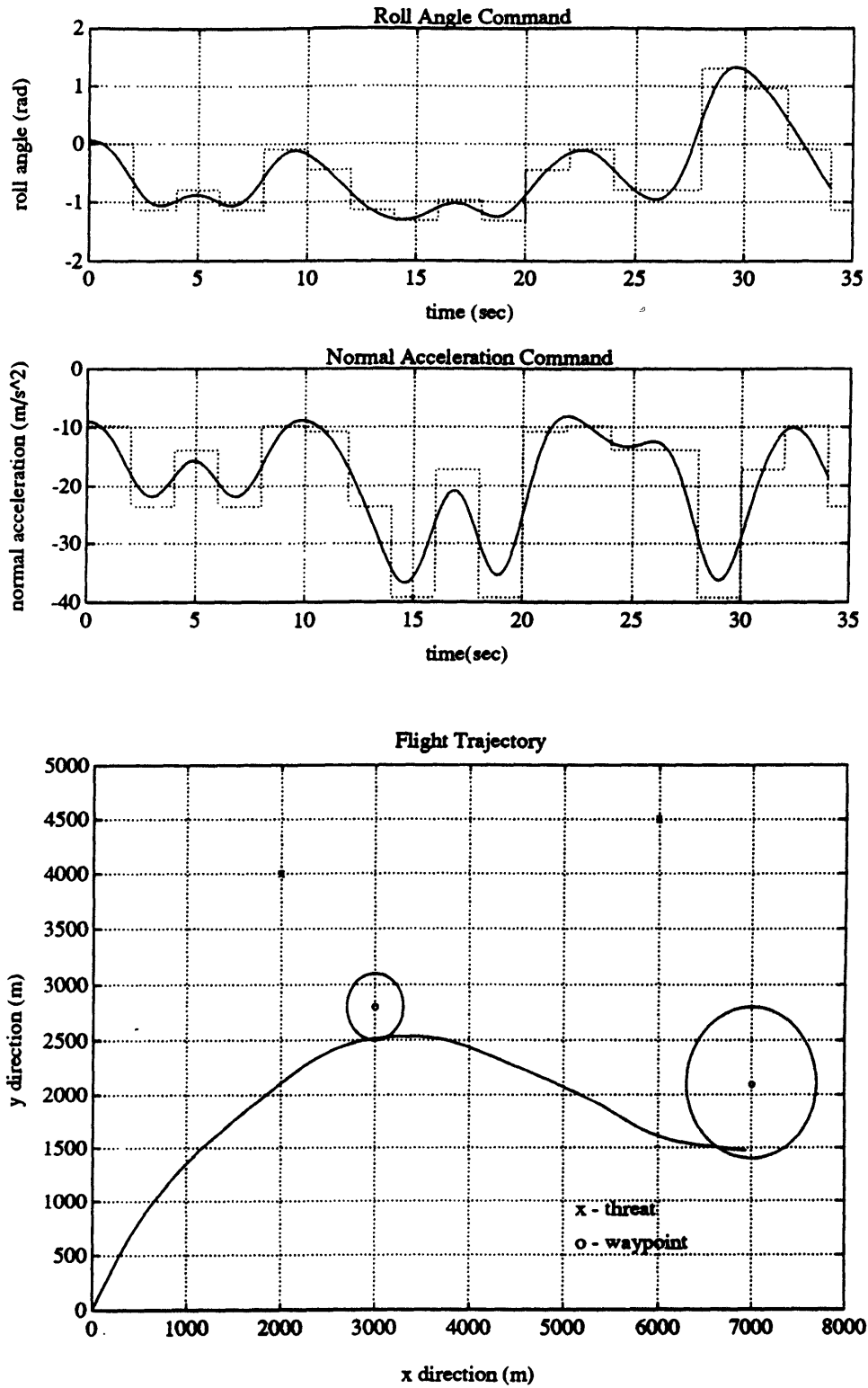


Figure 5.4: Result of genetic algorithm optimization of test segment 4.

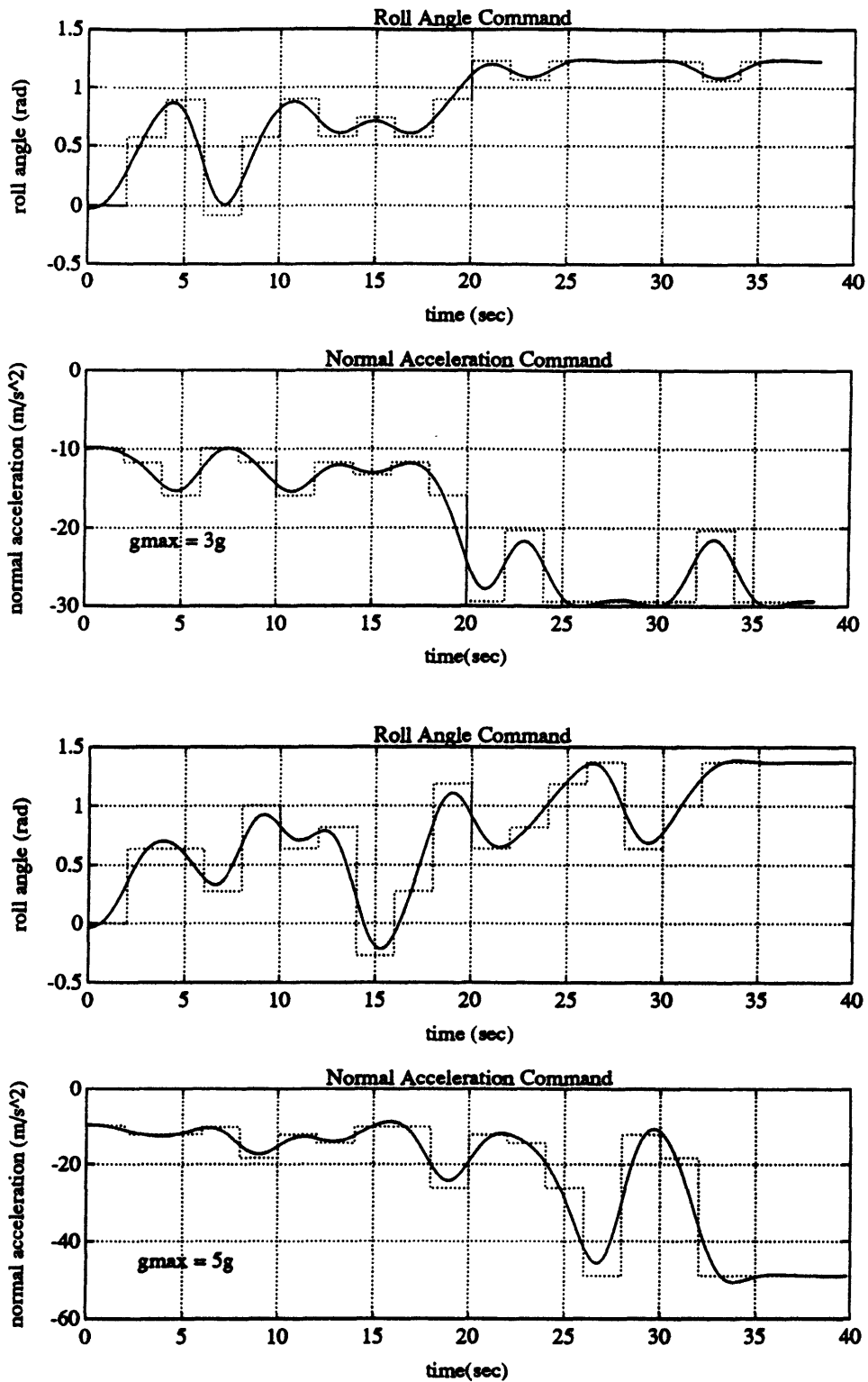


Figure 5.5: Flight control commands for different values of maximum normal acceleration.

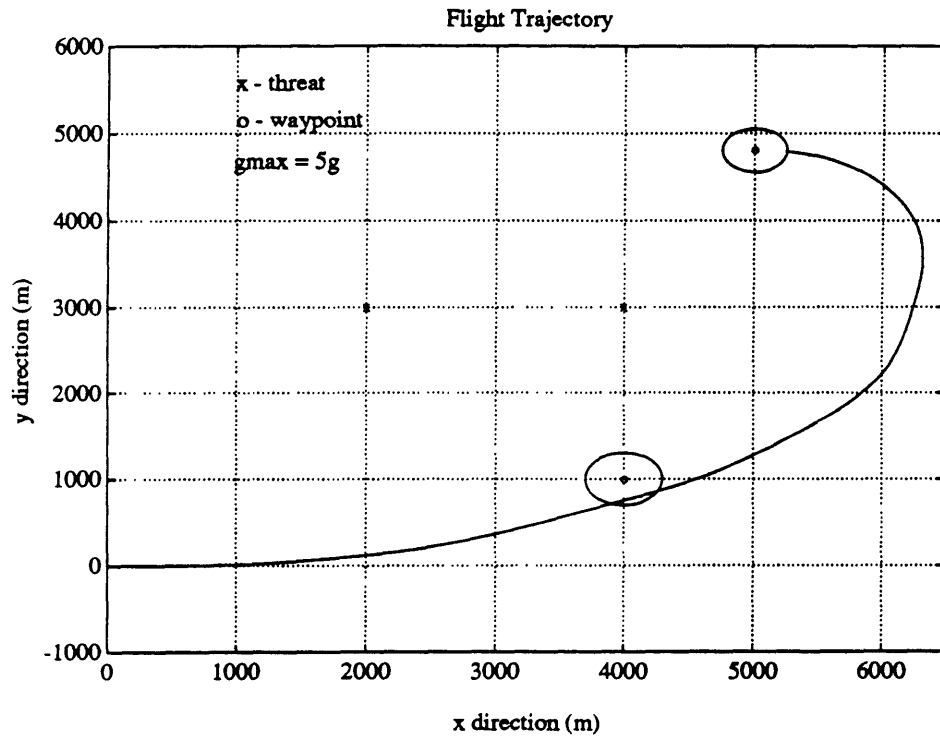
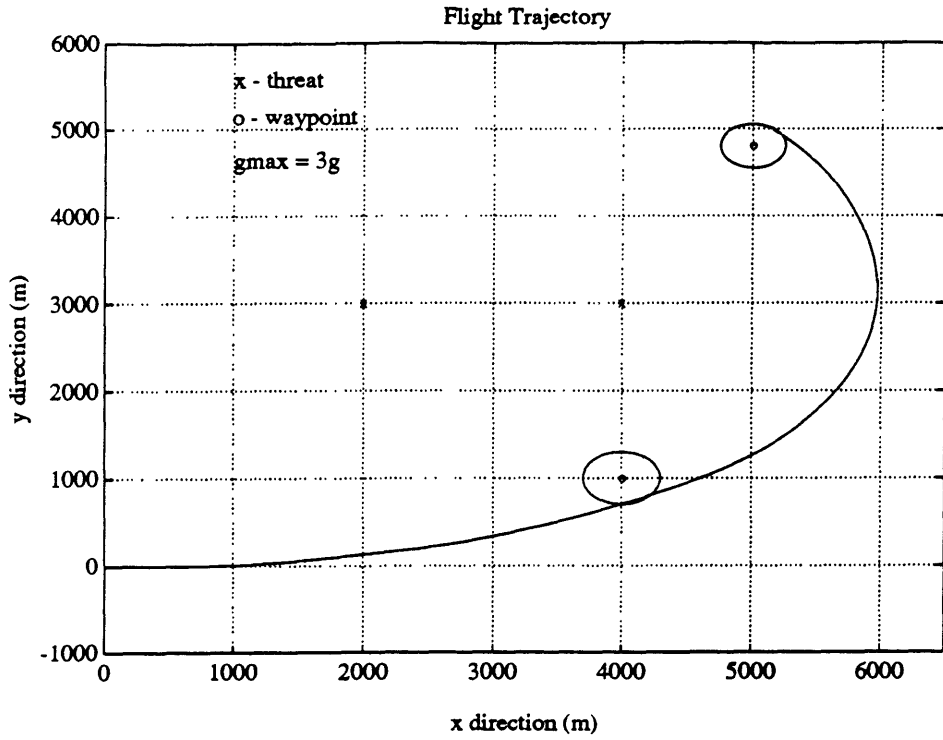


Figure 5.6: Flight trajectories for different values of maximum normal acceleration.

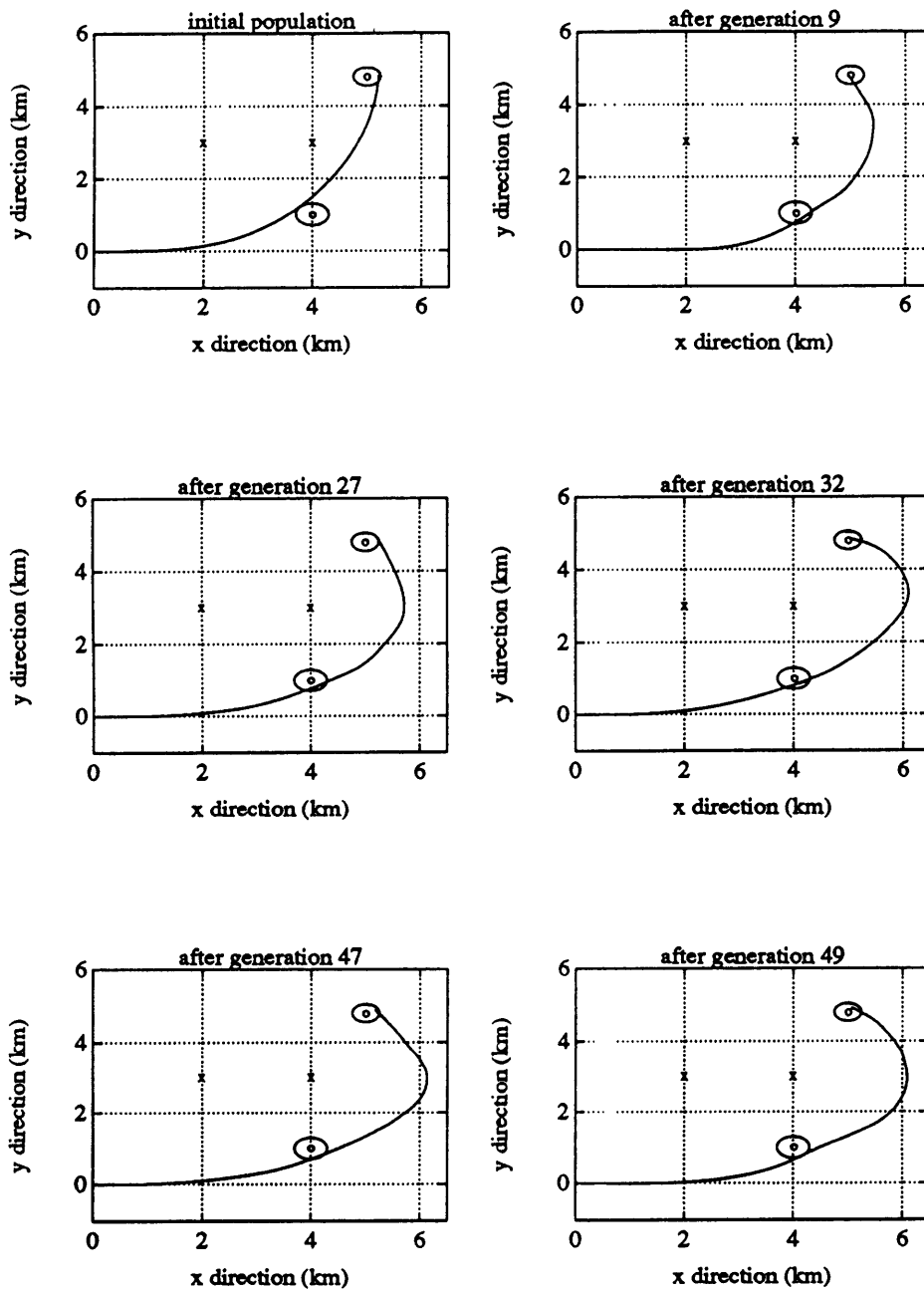


Figure 5.7: Evolution of the best constrained solutions available to the FCS for test segment 1.



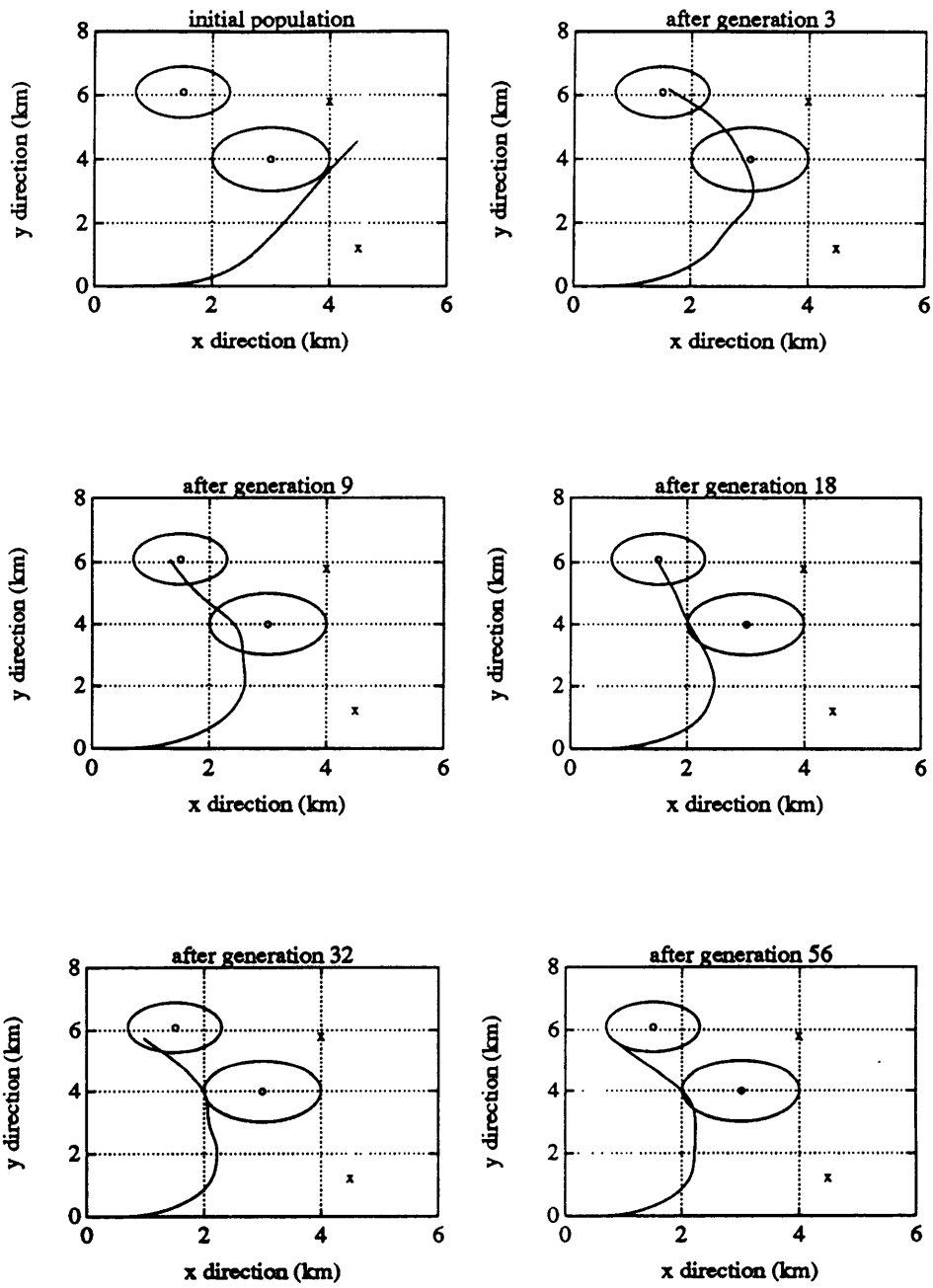


Figure 5.8: Evolution of the best constrained solutions available to the FCS for test segment 3.

# Chapter 6

## Conclusions

### 6.1 Summary

A mission planning system has been developed that directly optimizes the control commands for a high performance military vehicle's flight control system so that the resulting flight trajectory minimizes the defined measure of associated risk. However, the required computation time using Broyden-Fletcher-Goldfarb-Shannon (BFGS) minimization routines is enormous. The efficiency of the optimization algorithm used in the flight trajectory planner must be interpreted in the context of the real-time nature of the system. The trajectory planning is done in two-waypoint segments. To ensure continuity, the trajectory is followed only to the first waypoint after which a new trajectory segment is used. The system is therefore required to converge to a solution before the first waypoint is reached. Furthermore, the dynamics of the aircraft and certain trajectory objectives such as capture radii of waypoints place constraints on both the flight control commands and the resulting flight path. It is therefore essential to have an algorithmic structure that will rapidly produce a solution within these constraints and also allow for further improvement if time permits.

This thesis implemented a genetic algorithm as the optimization procedure in the flight trajectory planner. It proved that in the real-time context of the trajectory planning, the genetic algorithm optimization is preferable to the BFGS minimiza-

tion method used previously. The advantage of genetic algorithm optimization is that a solution satisfying the given constraints is available in the population after an average of only 200 evaluations (trajectory integrations) while still allowing for further improvement. The BFGS minimization could not assure a legally constrained trajectory solution until the very last iteration, which was an average of 6000 evaluations. Using genetic algorithm optimization does not guarantee an optimum trajectory solution at any time. However, the analysis showed that trajectory solutions which are very close to the optimum are available after approximately 2000 evaluations. Seen in the real-time context, it is clear that the genetic algorithm has superior performance.

Previous flight trajectory optimizations using the BFGS minimization routines were done on a 20MHz 386 personal computer and a constrained minimization took about 3 hours. Using a 33MHz 486 personal computer, the genetic algorithm took approximately 28 minutes for the same amount of trajectory evaluations as required on average by the BFGS routines. Both routines were coded using MATLAB. It should be kept in mind that Section 2.2 simplified the problem formulation used in the previous work, which reduced the subsequent search space. The increase in computation efficiency can therefore be ascribed to both better hardware technology and a smaller search space. It is possible to decrease the computation time radically by utilizing the binary nature of the genetic algorithm, as well as its ability to be implemented in parallel. Taking these properties of genetic algorithms into account, it is very probable that the flight trajectory planner can be implemented in a real-time system with current technology.

These positive results were realized only after giving careful attention to the method of coding the control commands histories as binary structures and to the method of generating the initial population. Optimum values were also found for the population size, and probabilities of crossover and mutation. Much time was also spent finding numerical values for the constants in the objective function, Equation 4.12, that lead to realistic solutions. The implementation of the genetic algorithm is therefore not as simple a procedure as it would seem at first glance.

## 6.2 Recommendations

The results shown in figures 5.1–5.4 have very good flight trajectories. This must be expected as the flight trajectory is the measure of the relative fitness of a member in the genetic population which determines its ability to reproduce. However, the control commands shown in these figures are not very smooth, and it is clear that the same flight trajectories can be obtained by much smoother control commands. This can be achieved in two ways; adjusting the time constant of the low pass filter or by penalizing the objective function for high activity in the control commands. The first method would be preferable, as it can be included in the modifications that have to be done to the low pass filter to eliminate the small overshoots. Furthermore, experience has shown that minor modifications to the objective function can result in very poor performance.

Some missions may have threats or waypoint objectives which are functions of altitude. To include altitude variations, it is necessary to implement the original problem formulation where the normal acceleration and roll angle commands are optimized independently. This can be achieved by modifying the objective function and appending the strings modeling each of the commands to form longer string structures. The modification of the objective function may require some effort to achieve good results as mentioned above.

The two properties of genetic algorithms which can greatly reduce the required computing time are the ability of most of the genetic algorithm operators to be implemented in parallel and the binary nature of the genetic individuals. By implementing the genotypes as actual bits in a register using a low-level programming language, it will be possible to realize the genetic algorithm operators as bit operators which will greatly reduce the computing time. As mentioned in Section 3.4 two advantages are gained by using a parallel genetic algorithm; the operators act in parallel and therefore require less computing time, and it is possible to evolve more than one population in parallel which results in a much more thorough walk through the search space. The implementation of a parallel genetic algorithm is therefore perhaps the most important area for further work towards achieving superior real-time efficiency in the MPS.

# References

- [1] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In John J.Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Lawrence Erlbaum Associates, 1987.
- [2] Mark F. Bramlette. Initialization, mutation and selection methods in genetic algorithms for function optimization. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 100–107, Morgan Kaufman, 1991.
- [3] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
- [4] Toni El-Dirani. *Fidelity of Flight Control Systems in a Real-Time Optimal Trajectory Planner*. Master’s thesis, Massachusetts Institute of Technology, 1991.
- [5] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc, 1989.
- [6] Chrisila B. Pettey, Micheal R. Leuze, and John J. Grefenstette. A parallel genetic algorithm. In John J.Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 155–161, Lawrence Erlbaum Associates, 1987.
- [7] Jon T. Richardson, Mark R. Palmer, Gunar Liepens, and Mike Hillard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–195, Morgan Kaufman, 1989.

- [8] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting on-line performance of genetic algorithms for function optimization. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60, Morgan Kaufman, 1989.
- [9] William J. Walker. *Flight Control Command Generation in a Real-Time Mission Planning System*. Master's thesis, Massachusetts Institute of Technology, 1990.