2010

A Study of Evolvable Hardware Adaptive Oscillators for Augmentation of Flapping-Wing Micro Air Vehicle Altitude Control

Bharath Venugopal Chengappa
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/364

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
A Study of Evolvable Hardware Adaptive Oscillators for Augmentation of Flapping-Wing Micro Air Vehicle Altitude Control

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

By

Bharath Venugopal Chengappa
B. E., Visvesvaraya Technological University, Belgaum, India, 2005

2010

Wright State University

____________________________
John C. Gallagher, Ph.D.
Thesis Director

____________________________
Mateen M. Rizki, Ph.D.
Department Chair

Committee on Final Examination

____________________________
John C. Gallagher, Ph.D.

____________________________
Michael Raymer, Ph.D.

____________________________
Mateen M. Rizki, Ph.D.

____________________________
John A. Bantle, Ph.D.
Vice President for Research and Graduate Studies and Interim Dean of Graduate Studies
Abstract


The control of insect-sized flapping-wing micro air vehicles is fraught with difficulties. Even when adequate control laws are known, limits on computational precision and floating-point processing can render it difficult to field implementations that provide sufficiently accurate and precise vehicle body placement and pose. Augmentation of an existing altitude controller with an Evolvable Adaptive Hardware (EAH) oscillator has been proposed as a means for an on-board altitude controller to correct control precision and accuracy difficulties during normal flight. This thesis examines a range of setting of the internal learning algorithms for the EAH oscillator and provides empirical evidence about which setting are most optimal for the control of a flapping-wing micro air vehicle (FW-MAV) based on the Harvard MicroFly. Implications for future multi-degree of freedom control are also considered.
# Contents

1 Introduction ............................................. 1  
   1.1 Motivation and Introduction .......................... 1  
   1.2 Altitude Control: A detailed View .................. 2  
   1.3 Flapping-Wing Micro Air Vehicle .................. 3  
   1.4 Objectives and Organization of this Thesis ......... 5  

2 Background and Literature Review ....................... 7  
   2.1 Evolutionary Computation ........................... 7  
      2.1.1 Genetic Algorithms .............................. 10  
      2.1.2 Terms Frequently Used ........................... 14  
   2.2 The Mini Population Algorithm ..................... 18  
   2.3 Evolvable Hardware ................................ 21  
   2.4 The Flapping-Wing Micro Air Vehicle (FW-MAV) ...... 21  
      2.4.1 Altitude Control using the ACTC ............... 23  
      2.4.2 Split-Cycle Control ............................ 25  

3 Methodology and Model .................................. 29  
   3.1 Introduction ........................................ 29  
   3.2 Model Overview .................................... 30  
   3.3 Non-EAH Based Controller Architecture ............... 35  
   3.4 EAH Based Controller Architecture ............... 40  
   3.5 Modified Minipopulationary Algorithm ............ 43  

4 Simulation Setup and Performance Analysis ............ 47  
   4.1 Simulation Setup .................................... 47  
   4.2 Assessment Parameters for Altitude Tracking ....... 51  
   4.3 Performance Assessment of the Modified Minipop Algorithm .... 52
List of Figures

1. The 6 Degrees of Freedom in a 3-D Space ............................................. 3
2. First insect scale flying robot able to take off.[1] ................................. 4
3. Hummingbird Hovering in mid air. ....................................................... 5
4. General Scheme for a GA as a flowchart. .............................................. 11
5. Pseudo-code for a GA. ................................................................. 11
8. The FW-MAV Orthographic View[4]. .................................................. 22
9. The Block Diagram of the Altitude Command Tracking Controller[4]. .... 24
10. Sample Generated outputs from EAH-Oscillator. ............................... 28
11. The Basis Functions stored in the Wave Table ROM. ........................... 29
12. General Assembly of the FW-MAV [5]. .............................................. 31
13. The Wing Design used for Force Calculations ................................. 32
16. Split-Cycle Cosine Module for the EAH Based Controller.[4] ............... 41
17. A Better Look at the Shuffle LUT and Wave Table ROM .................... 43
18. Pseudocode for the Modified Minipop Algorithm. ............................ 44
19. Test Pattern Design ..................................................................... 50
20. Data Analysis for Mutation Rate Sweep ............................................ 55
21. Data Analysis for Population Size Sweep .......................................... 58
22. T-Test Results for Population Size Sweep ........................................ 59
23. Surface Plot for Combined Sweep of Population Size and Mutation Rate 60
24. Surface Plot of Standard Deviation for Combined Sweep of Population Size and Mutation Rate ............................ 61
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Surface Plot of Fitness Value for Combined Sweep of Population Size and Mutation Rate</td>
</tr>
<tr>
<td>26</td>
<td>Surface Plot Standard Deviation for Fitness Value</td>
</tr>
<tr>
<td>27</td>
<td>Surface Plot for Flaps per Evaluation Parameter at 40</td>
</tr>
<tr>
<td>28</td>
<td>Surface Plot of Standard Deviation for Flaps per Evaluation Parameter at 40</td>
</tr>
<tr>
<td>29</td>
<td>Surface Plot of Fitness Value for Flaps per Evaluation Parameter at 40</td>
</tr>
<tr>
<td>30</td>
<td>Surface Plot Standard Deviation for Fitness Value Flaps per Evaluation Parameter at 40</td>
</tr>
<tr>
<td>31</td>
<td>Surface Plot for Flaps per Evaluation Parameter at 60</td>
</tr>
<tr>
<td>32</td>
<td>Surface Plot of Standard Deviation for Flaps per Evaluation Parameter at 60</td>
</tr>
<tr>
<td>33</td>
<td>Surface Plot of Fitness Value for Flaps per Evaluation Parameter at 60</td>
</tr>
<tr>
<td>34</td>
<td>Surface Plot Standard Deviation for Fitness Value Flaps per Evaluation Parameter at 60</td>
</tr>
<tr>
<td>35</td>
<td>Summary of the Learning Parameter Sweeps</td>
</tr>
</tbody>
</table>
## List of Tables

1. Minipop Parameters ................................................. 20
2. FW-MAV Model Parameters ......................................... 30
3. FW-MAV Model Constant parameters ............................... 31
4. Input parameters for the Non-EAH Oscillator module. .......... 36
5. Output parameters for the Non-EAH Oscillator module. ........ 37
6. Initial settings for Mutation Rate Sweep .......................... 48
7. Initial settings for Population Size Sweep ........................ 48
8. Initial settings for Population Size and Mutation Rate Sweep ........ 49
9. Combination list for the Population Size and Mutation Rate Sweep .... 49
10. Parameter settings for Flaps per Evaluation Sweep .............. 50
11. Learning Times for Varying Mutation Rate ....................... 53
12. Fitness Values for Varying Mutation Rate ....................... 53
13. T-Test Results for Varying Mutation Rate ....................... 54
15. Fitness Values for Varying Population Size ..................... 57
16. ANOVA Results Flaps per Evaluation - 50 ....................... 63
17. ANOVA Results Flaps per Evaluation - 40 ....................... 66
18. ANOVA Results Flaps per Evaluation - 60 ....................... 69
ACKNOWLEDGMENTS

I would like to acknowledge my gratitude to my advisor Dr. John Gallagher, for the unconditional support, guidance, extended patience, and encouragement throughout my graduate career. I would also like to thank Dr. Mateen Rizki and Dr. Michael Raymer for their patience, feedback and suggestions about my work. I would like to thank all my friends for their support and friendship. Lastly I would like to thank my father, mother and brother for helping me be strong throughout my studies.
Dedicated to

Dr. John Gallagher, papa and mummy
1 Introduction

1.1 Motivation and Introduction

Numerous attempts to achieve controlled flight in insect-sized robots using the driving force of flapping wings have recently been recorded in the literature[1],[6],[7],[8]. This motivation to develop a Flapping-Wing Micro Air Vehicle is mainly for reconnaissance robots which are small in size, highly maneuverable, and can be controlled remotely in a three dimensional space. The uses for such a vehicle are numerous, such as monitoring large crowds, reaching places where only inch-sized vehicles can reach, and many more.

One such effort is underway at the Wright Patterson Air Force Research laboratory to build an insect-sized Flapping-Wing Micro Air Vehicle, a suitable closed loop controller and a oscillator that controls the wing beat of the vehicle through various flight patterns. Such controllers are application-specific and are designed using traditional design rules. Gallagher proposed an adaptive Evolvable Hardware approach for the oscillator [4] in the initial stages of the design to help the traditionally-designed controller in altitude control of the vehicle [5]. The initial designs were focused on altitude and hover control. The EAH design is focused towards learning to occur from the initial steps of flight considering the physical limitations of the vehicle and effects of the atmosphere on the vehicle.

This thesis will analyze and test the learning algorithm used to design the oscillator for hover mode of flight and identify the optimal setting of the various learning parameters. The learning algorithm used here is a variant of the Mini-population (MiniPop) [3] genetic algorithm which is light weight and has been implemented in hardware to show that it has a small hardware footprint [9],[10]. The rest of this chapter will introduce us to the methods that were used for the design of the Evolvable Adaptive Hardware for the oscillator in a short but detailed manner.
1.2 Altitude Control: A detailed View

Micro Air Vehicles are based on either bird-sized or insect-sized robots. The force required to counteract gravity for a flapping-wing MAV is provided by the flapping wings. A controller for such applications should be able to command wing motions in a manner that regulates the altitude in accordance with the required operational needs. In this Flapping Wing Micro Air Vehicle (FW-MAV) the altitude control is achieved by controlling the frequency at which the wings beat to generate lift. The frequency $\phi$ of the wing is directly related to the angular velocity of the wings which is in turn proportional to the lift produced by them over each complete wing beat. The structure of the wing is generally a triangular shape that has been demonstrated to produce the required lift for the vehicle [1], [11].

The EAH oscillator enhances the altitude control as it widens the available search space used to generate the control wave around the hover frequency, fed to the wings of the FW-MAV. This helps sharpen the ability of the FW-MAV to track a specified altitude. The ability for the FW-MAV to adapt and learn to track changes in altitude on its own is the most important requirement for the FW-MAV. Hovering mode of flight is the most important flight mode for a FW-MAV. A model written in C that simulates the forces generated by the wings is used to simulate and test the altitude control by the Modified Minipop Algorithm. The altitude control is cycle averaged over a wing beat and centered around a Split-Cycle Cosine wave at the hover frequency which is manipulated to control the forces produced by the wings of the simulated FW-MAV. The frequency manipulation is made around a central frequency which has been calculated to be about 120 Hz for hover [5] for this model of the FW-MAV with a single degree of freedom in flight.
1.3 Flapping-Wing Micro Air Vehicle

The Flapping-Wing Micro Air Vehicle (FW-MAV) considered here is a variant of the Harvard Robofly [1], this Robofly was the first insect-scale MAV to achieve takeoff. FW-MAV’s are based on the flying motion of insects of the order Diptera (di = two, ptera = wings), which include mosquitoes, gnats, midges and flies. The Harvard Robofly has only one degree of freedom of control considering that a single piezoelectric actuator drives both the wings to produce lift, where as the FW-MAV designed at the Air Force Laboratory has two piezoelectric actuators which drive the two wings hence allowing two degrees of freedom for control of this FW-MAV. The FW-MAV controller needs to be designed to allow controlled flight in all the six degrees of freedom found in a three dimensional space. In this thesis the learning of the oscillator for the hover mode of the simulated vehicle is tested.

![Figure 1: The 6 Degrees of Freedom in a 3-D Space](image)

The Robofly consists of four main mechanical components: the airframe, actuator, transmission, and the airfoils. The functions of these parts are very basic, the airframe provides a solid body for the MAV in which a payload consisting of battery, controller and required basic electronics can be accommodated, the actuator provides the motion for the wings with maximum power, the transmission which efficiently impedance-match the actuator to the load, and lastly the airfoils must be rigid enough to hold their shape when subjected to the large aerodynamic loads. The Harvard Robofly which was able to successfully take off while tethered to a pair of wires along its
central axis as shown in Figure 2.

![Figure 2: First insect scale flying robot able to take off.][1]

The Robofly uses a single piezoelectric actuator to drive both its identical wings. Here the tangential motion of the actuator tip is converted to the forward and backward stroking motion of the two wings. This motion can be described as a person in a swimming pool with his arms wide apart along the surface of the water. The plane of motion for the wings is compared to the surface of water in a swimming pool and the central axis of the vehicle being a vector along the human spine. The wings swing back and forth along the surface of the water like when the person treads the water along the surface. Each wing has a triangular plane form that hangs down from the sweeping arm and passively rotates to a maximum elevation due to air resistance during motion. The flapping motion of the wings can also be closely related to the motion of the wings of a hummingbird when it is hovering in mid air. In the model considered here the single piezoelectric actuator is replaced with two piezoelectric actuators, one to drive each wing respectively and independently.
1.4 Objectives and Organization of this Thesis

The objective and goal of this thesis is to test and analyze the design of the genetic algorithm used in the Evolvable Hardware Synthesized oscillator for the control of the Flapping-Wing Micro Air Vehicle for a specific flight mode of hover with a broken right wing. The genetic algorithm used here is a variant of the MiniPop algorithm, which is driven by mutation and employing tournament selection [12], [13]. The analysis from this thesis might help support the future modifications to the MiniPop Algorithm. While tuning the various learning parameters of the Minipop Algorithm two main goals were considered,

1. To find the optimal setting of the Minipop Algorithm for its learning parameters. This setting help the EAH-Oscillator learn to make the FW-MAV achieve sub millimeter precision while tracking a specified altitude. The performance of the algorithm over the considered range of the learning parameters is analyzed to learn the limitations of the algorithm.

2. Minimization of the learning time required for the EAH Oscillator to command the FW-MAV to achieve the specified altitude is a very important goal, as it plays a vital role in the flight of the FW-MAV. The EAH-Oscillator acts as a bridge between the traditionally-designed controller based on the mathemati-
cal model and the vehicle in real world situations and materials used to build the FW-MAV, hence the EAH-Oscillator needs to learn fast to adapt to new physical conditions. Hence to help understand the performance of the Minipop Algorithm for a condition of the FW-MAV with a broken wing, simulations were conducted, and these simulation results were analyzed based on the learning time required for the EAH-Oscillator to adapt to the new physical condition over a range of parameter settings, and in this process to find the setting which has the least learning time.

3. Submillimeter precision and learning time for altitude tracking are two cases that are so closely related, that a balance between the two settings is necessary. The best performance of the FW-MAV in real conditions to track altitude and hover when the vehicle is subjected to damage is observed at the optimal setting of both the parameters. Analyses of the results from the simulations to find a good balance for the above two main cases is an objective of this thesis.

The literature review and background is discussed along with the explanation of necessary terminology, the methods and approaches discussed in this thesis are briefly explained in chapter two. In Chapter three the model of the FW-MAV and the specific EAH methods employed along with the hardware synthesized oscillator for the control of the FW-MAV are discussed and described. Chapter four explains the test procedures, simulation procedures used to analyze the learning parameters of EA. The results of the simulations and comparisons of the test results are also discussed in detail.
2 Background and Literature Review

2.1 Evolutionary Computation

Biological evolution can be viewed as a population-based stochastic optimization method that continuously improves a population’s ability to survive their environments. The most salient features of biological evolution, from the perspective of computational scientists, might be its relative robustness against getting stuck in non-adaptive niches and its ability to buffer organism features that are not necessarily adaptive at the moment, but might become so in the future. The field of Evolutionary Computation (EC) attempts, in a number of ways, to create computer optimization techniques that in some way mimic natural evolution and, hopefully, retain some of natural evolution’s benefits.

Evolutionary methods are used in Computer Science to address a wide variety of problems. Broadly these problems are most clearly understood as optimization problem in which the goal of the Evolutionary Algorithm (EA) is to find parameter settings of an objective function to produce a minimum error value, or alternatively, a maximum gain value. Under that umbrella of optimization, there are myriad specific types of problems. For example, objective function optimization can be applied to classification problems in which the goal is to minimize the error of a pattern classification system. Also, one could employ EC to minimize an objective function representing the performance of a electromechanical device with particular tunable device characteristics. In this application, the EA would be tuning parameters of the device or its controller and measuring performance with respect to an objective function that measures how well that device operates. This thesis will focus on an application of this second example type. Note also that real evolution, as opposed to the EC methods we will consider here, do not optimize an explicit objective function. Rather, they "bias organism form and behavior" against implicit objectives designed
to improve the survivability of the population as a whole. This observation has led to some alternative views of EC that are not discussed in this thesis as they are beyond the scope of the work done here.

It is commonly agreed that there are three main lineages of Evolutionary Computation. Evolutionary Programming [14], Genetic Algorithms [15], and Evolutionary Strategies [16]. All three of these lineages are similar in intent, although they do differ in the applications to which they were originally applied and in specific choices of problem representation and algorithmic details. In general, all EC methods have most, if not all of the following algorithmic features:

1. **A Representation**: A representation is a means of encoding parameter settings for an objective function into a data structure that serves as the roll of an individual candidate solution’s ”DNA”. Genetic Algorithms, for example, generally represent individuals as strings of zeros and ones. Other possible representations are strings of floating-point values, or even structures with more complex topology like rings, trees, or graphs.

2. **A Population**: An EA maintains a population of individual solution representations.

3. **A Selection Method**: An EA has some means of selecting individuals from a population who are destined to survive into a future generation. Selection methods vary widely. The canonical Genetic Algorithm, for example, uses something termed ”roulette wheel selection” in which a population member’s probability of surviving into the next generation is proportional to its fitness measured relative to other members in the population. Another popular selection method is ”tournament selection” in which population members are selected randomly and allowed to compete, with the winner gaining a place in a future population.
4. A **Recombination Method**: Population members that survive into a new generation are also given an opportunity to "mate" and create children that intermingle the parent representations. These children then compete with other members of the population for their own survival. Recombination methods vary and are highly dependent on the particular representation chosen. Generally, one attempts to choose representations and recombination methods that allow for meaningful blends of parent characteristics.

5. A **Mutation Method**: An EA generally has some means to inject random variation into their populations. Genetic Algorithms, for example, employ bit-wise mutation in which every bit in a genome has a small chance of flipping. Mutation is a means to introduce novelty into a population and allow it to escape from "inbreeding" in which all population members have become so similar that no innovation is possible.

In practice, applying an EA to a particular problem involves making intelligent choices for each of the above generic features in a manner that is consistent with the needs of the problem at hand. In theory, any set of choices should work for any problem given infinite computational resources. In practice, one needs to carefully choose representations and operators that exploit structure inherent in the problem at hand to both minimize the number of times "bad" solutions are evaluated and to maximize the number of times the EA visits "good" portions of the entire search space. Doing this correctly is a constant challenge.

In this background section, we will focus on a specific form of the Genetic Algorithm that had been previously designed and optimized for use as an on-chip optimizer of analog neural network settings. That algorithm most arguably in the lineage of genetic algorithms, though it shares some features of other lineages. Therefore, this section will focus on GAs first and then on the specific algorithm used in this thesis.
(MINIPOP). In this work, the most salient features of the search are that there is very little physical space on the chip to implement a complex learning algorithm and that the time to evaluate candidate solutions is orders of magnitude greater than the speed of the computational clock (tenths of seconds as opposed to millionth of seconds). Thus, in designing an EA for this problem, the most important features would be getting maximum value out of each candidate evaluation and keeping the computational circuitry small. This thesis will take it as given that the MINIPOP has these features with respect to the problem of this thesis and test that assumption with respect to it. The conclusions section will offer some comment on how well MINIPOP does and what might be done to make it perform better.

2.1.1 Genetic Algorithms

Algorithms that simulate Evolution by encoding or computing a solution for a very specific problem by using recombination operators on simple bit strings are called Genetic Algorithms(GA). Genetic algorithms are used to address very specific problems by encoding a possible or potential solutions on simple bit strings. The functioning of a GA can be understood by looking at a general scheme followed while designing Evolutionary Algorithms.

The working of a GA can be explained in simple terms by observing Figure 4 and 5. The terms used here like population, parent selection, recombination will be explained in the next section. A Genetic Algorithm is designed to address a problem which can be defined and represented in a function. The population is a set of candidate solutions for that function, these candidate solutions need to be initialized or can be subjected to evolution without initialization. Initialization helps the process of evolution to start from a particular condition of the problem and forms a baseline. Evolution is simulated by first selecting a single or pair of parent candidate solutions, from which a offspring solution is generated by evolving the parents based
Figure 4: General Scheme for a GA as a flowchart.

on variation operator used. The generated solution is subjected to a fitness test where the fitness of the generated solution is tested with respect to the objective function of the problem at hand.

Figure 5: Pesudo-code for a GA.

The implementation of a GA can be itemized into a couple of steps,
1. The objective function for the problem at hand is defined and the required fitness criteria for any possible solution is found.

2. The population as explained earlier in the general scheme consists of a set of possible solutions called population members. These members which mainly consist of bit-strings (0, 1) can be initialized depending on the requirement of the problem at hand. This process of initialization simulates a defined starting point for all the candidate solutions.

3. Each of the above population members are evaluated with respect to the objective function and a respective fitness score is assigned.

4. An appropriate selection process is used to select members to be parents, from the population. The new set of selected members form a new population.

5. Member pairs are selected and subjected to variation operators suitable for the problem at hand to generate a new population of possible solutions.

6. The old population is replaced with the new population of offspring’s generated from the variation operator.

7. The population is checked if a suitable solution is found for the objective function of the problem. Once the suitable solution is found the process of evolution is stopped.

Genetic Algorithms are suitable for hardware implementations as they are aimed at solving very specific problems. GA’s are preferred to have reasonably sized population sizes which also supports the effective hardware implementation. Hardware designed with the implementation of a GA falls into the category of Evolvable Hardware. The Selection and Variation process involved in a Genetic Algorithm is very
crucial, because these are the processes that implement evolution in Genetic Algorithm to try and search for a better solution with in a defined search space of solutions for the objective function of the problem considered.

Figure 6: Selection and Recombination in a Genetic Algorithm.[2]

There are several types of selection, recombination, and mutation schemes which are explained in the next section. Figure 6 shows one of many schemes for selection and recombination, it shows the population members being assigned positions during the selection for the parents and in the recombination (crossover in this example) it shows how the offspring is generated from the selected parents. The formation of the new population, which consists of the generated offspring’s is the actual evolution process taking place to search for a better solution in a defined search space for the objective function at hand. Mutation is a unique process which can replace a recombination process. In mutation the main difference is that there will only be a single parent for every offspring generated in a the new population. There are many
implementations of Genetic Algorithms, like Goldberg’s (1989) implementation called the Simple Genetic Algorithm (SGA), the Compact Genetic Algorithm (cGA) where the GA is designed for very small populations sizes and many more.

2.1.2 Terms Frequently Used

A number of components and procedures have to be specified to define a genetic algorithm. In this section some basic terms like encoding, population, selection, recombination and mutation are explained.

**Encoding**

Encoding can be explained as a process of representing a real world problem into a format that is suitable for an genetic algorithm to function properly. Encoding can be in the form of any meaningful symbols, binary valued strings or real valued vectors. But classically the encoding for a genetic algorithm is done as binary valued strings.

**Population**

A population is the pool of candidate solutions. One or more candidates are chosen or are the current champions and the other members of the population represent sample points in other regions of the search space, where there is a possibility of finding a better solution later.

**Selection**

During evolution of a GA various generations of the population are generated and replace the old population, in each successive generation a portion of the existing population is selected to breed a new generation. This selected portion are the most fit members of the previous generation, the least fit members are eliminated. The notion of fitness depends on whether a solution is feasible and also partly on its objective function value. Certain selection methods rate the fitness of each member and most only rate a random sample of the population to save time. There are a
number of selection schemes, a few of which are Rank-based fitness scheme, Roulette Wheel selection, Stochastic Universal sampling, Tournament selection. Tournament selection applied for mutation as the genetic operator would be a direct competition between the individuals present in the population, based on a fitness criteria for the objective function of the problem.

Some of the selection methods are discussed here,

1. **Roulette Wheel Selection:** Roulette wheel selection is the simplest selection scheme. It is also called stochastic sampling with replacement. The individuals are mapped to a contiguous segment of a line such that each individual’s segment is proportional in size to its fitness. A random number is generated and the individual whose segment spans the random number generated is selected. This process is repeated until a desired number of individuals are obtained.

2. **Stochastic Universal Sampling:** In Stochastic universal sampling individuals are mapped to contiguous segments of a line, such that each individual’s segment is equal in size to its fitness just like in roulette wheel selection. Here equally spaced pointers are placed over the line as many as there are individuals to be selected. If \( N_p \) are the number of individuals to be selected, then the distance between the pointers will be \( 1/N_p \) and the position of the first pointer is given by the random number generated in the range \([0, 1/N_p]\).

3. **Tournament Selection:** In Tournament selection a group of \( T_{our} \) individuals are chosen randomly from the population and the best individual from this group is selected as a parent. This step is repeated for the number of individuals to be selected. These parents produce uniform random offspring. The parameter for tournament selection is the tournament size \( T_{our} \) and \( T_{our} \) takes values in
the range 2 to \( N_{\text{ind}} \).

\[
\text{Selectionintensity}(T_{\text{our}}) \approx \sqrt{(2.\ln(T_{\text{our}}))} - \ln(\sqrt{4.14.\ln(T_{\text{our}}}))
\]

(1)

\[
\text{Loss Diversity}(T_{\text{our}}) = T_{\text{our}}^{-1/(T_{\text{our}}-1)} - T_{\text{our}}^{-T_{\text{our}}/(T_{\text{our}}-1)}
\]

(2)

\[
\text{Selection Variance}(T_{\text{our}}) \approx \frac{0.918}{\ln(1.186 + 1.328.T_{\text{our}})}
\]

(3)

Recombination

When the information contained in two or more parents are combined to form new individuals, this process is called recombination. The two or more chromosomes (parents) values which can be variable are combined together to form a new chromosome (offspring). This is done so that the new chromosome gets the best characteristics from each of the parents. Recombination occurs during evolution according to a user-defined recombination probability.

Binary -Valued recombination is used on individuals with binary variables. The other name by which this recombination method is referred is crossover. During a recombination only parts of an individual are exchanged between the parents. The individuals are divided depending on the number of parts, before the exchange of the variables. The encoding method used classically in Genetic Algorithms is binary-valued strings, following are a few basic types of crossover mechanisms used in Genetic Algorithms.

1. **One-point Crossover:** In one-point crossover a random point \( k \) is chosen, where \( k \in \{1,2,...,N_{\text{var}}\} \) where \( N_{\text{var}} \) is the number of variables in an individual. \( K \) is chosen uniformly at random and the variables are exchanged between the individuals about this point to form the offspring’s.
Consider the below example,

Individual 1 01101010
Individual 2 11010001
Let k (crossover point) = 3
Offspring1 011—10001
Offspring2 110—01010

2. **Multi-point Crossover:** In multi-point crossover the variables between successive crossover points are exchanged between the two parents to produce two new offspring’s. Here the section between the first variable and the first crossover point are not exchanged between the individuals. Let us consider the individuals from the above example and let the crossover points be (3,6).

Offspring1 011—100—10
Offspring2 110—010—01

3. **Uniform Crossover:** Uniform crossover makes every variable in an individual a potential crossover point. A mask can be generated at random, which is of the same length as the individual structure. The parity of each bit in the mask indicate from which parent the bit for that offspring will come from, this method is similar to the discrete recombination. Let us consider the same set of parents from the above examples.

Individual 1 01101010
Individual 2 11010001
Mask for offspring1 01100101
Mask for offspring2 10011011
Offspring1 00001111
4. **Shuffle Crossover:** Here a single crossover position is selected and before the variables are exchanged the variables in each of the individuals are randomly shuffled and then the crossover is done. After the recombination the variables of the offspring’s are not shuffled. This helps remove positional bias as the variables are randomly reassigned each time a crossover is performed.

**Mutation**

In mutation the individuals are randomly altered, the mutation steps (variations) are small and are applied to the individuals with a mutation rate (low probability). Both these parameters are constant during a whole evolutionary run. Also one or both these parameters are used or adapted to previous mutations. Mutation avoids stagnation of a population by introducing fresh individuals or variation. Mutation can be performed on real or binary valued vectors but as mention earlier the encoding method in a Genetic Algorithm is binary valued strings. Binary mutation for individuals with binary values means flipping of the variable values as they have only two different states. Hence the mutation step size is always 1. A uniform random value is chosen for every variable value in a individual to be changed. Let us look at an example for binary mutation, where the variable 6 is mutated in an individual with 8 variables.

Before mutation 01101010

After mutation 01101110

### 2.2 The Mini Population Algorithm

The Mini Population algorithm is in short called the Minipop algorithm, this is a compact genetic algorithm which is tournament based and driven by mutation. Minipop has a very efficient hardware implementation due to the use of a small population like
a Micro Genetic Algorithm. The main characteristics of the Minipop Algorithm are mutation hyper-mutation and its rejection of recombination operators. In this project variants are introduced and tested to examine their effects, also if EAH methods add benefit to the design of the controller.

<table>
<thead>
<tr>
<th>MINIPOP(N, L, MRATE, MAXEVALS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. eval := 0</td>
</tr>
<tr>
<td>2. for i := 1 to N do</td>
</tr>
<tr>
<td>3. pop[i] = RANDOM_BITSTRING(L)</td>
</tr>
<tr>
<td>4. fitness[i] = EVALUATE(pop[i])</td>
</tr>
<tr>
<td>5. eval := eval + 1</td>
</tr>
<tr>
<td>6. done</td>
</tr>
<tr>
<td>7. i := 1</td>
</tr>
<tr>
<td>8. while eval &lt; MAXEVALS do</td>
</tr>
<tr>
<td>9. if eval MODULO RF = 0 then</td>
</tr>
<tr>
<td>10. j := BEST_SOLUTION(pop)</td>
</tr>
<tr>
<td>11. old_fitness := fitness[j]</td>
</tr>
<tr>
<td>12. new_fitness := EVALUATE(pop[j])</td>
</tr>
<tr>
<td>13. eval := eval + 1</td>
</tr>
<tr>
<td>14. fitness[j] := (1-RW)<em>old_fitness + RW</em>new_fitness</td>
</tr>
<tr>
<td>15. else if i &lt;= N then</td>
</tr>
<tr>
<td>16. mutant := MUTATE(pop[i], MRATE)</td>
</tr>
<tr>
<td>17. mfitness := EVALUATE(mutant)</td>
</tr>
<tr>
<td>18. eval := eval + 1</td>
</tr>
<tr>
<td>19. if mfitness &gt; fitness[i] then</td>
</tr>
<tr>
<td>20. pop[i] := mutant</td>
</tr>
<tr>
<td>21. fitness[i] := mfitness</td>
</tr>
<tr>
<td>22. endif</td>
</tr>
<tr>
<td>23. i := i + 1</td>
</tr>
<tr>
<td>24. else</td>
</tr>
<tr>
<td>25. mutant := RANDOM_BITSTRING(L)</td>
</tr>
<tr>
<td>26. mfitness := EVALUATE(mutant)</td>
</tr>
<tr>
<td>27. eval := eval + 1</td>
</tr>
<tr>
<td>28. j := WORST_SOLUTION(pop)</td>
</tr>
<tr>
<td>29. if mfitness &gt; fitness[j] then</td>
</tr>
<tr>
<td>30. pop[j] := mutant</td>
</tr>
<tr>
<td>31. fitness[j] := mfitness</td>
</tr>
<tr>
<td>32. endif</td>
</tr>
<tr>
<td>33. i := 1</td>
</tr>
<tr>
<td>34. endif</td>
</tr>
<tr>
<td>35. done</td>
</tr>
<tr>
<td>36. j := BEST_SOLUTION(pop)</td>
</tr>
<tr>
<td>37. return pop[j]</td>
</tr>
</tbody>
</table>

Figure 7: Pesudo-code for a Standard Minipop Algorithm.[3]

The pseudocode for a Standard Minipop Algorithm is shown in figure 7, the
variables and functions referenced in the pseudocode are listed in table 1. If we look at the pseudocode we can see that in the lines 1-6 of the algorithm the initialization of the population is done by first creating and then evaluating N randomized bit strings. The rest of the lines contain the algorithm’s main loop where the solution is found by the process of evolution. In the main loop the search is driven by mutation and hyper-mutation tournaments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>4</td>
</tr>
<tr>
<td>Genome Length</td>
<td>1088</td>
</tr>
<tr>
<td>Resample Rate</td>
<td>Every 25 evals</td>
</tr>
<tr>
<td>Bitwise Mutation Rate</td>
<td>0.01</td>
</tr>
<tr>
<td>RNG Seed</td>
<td>System Clock</td>
</tr>
<tr>
<td>Max Evaluations</td>
<td>180,000</td>
</tr>
</tbody>
</table>

The mutation tournament is coded in lines 16 through 23. Here a member of the population which was initialized, A, is compared or competes with a mutated version of itself, if the mutated version wins it replaces the population member A in the population. The hyper-mutation tournament can be found in lines 25-33. In the hyper-mutation tournament the member with the worst fitness, B, then the hyper-mutant which was initialized earlier competes with B, if the hyper-mutant wins it replaces B in the population. Hyper-mutation tournament is used to prevent weak member to remain in the group and allows the algorithm to make large jumps in a search space. In this project the hyper-mutation tournament is not used due to its property of large jumps across a search space which will not be suitable for FW-MAV. This will be explained in detail in chapter 4. However in the standard Minipop Algorithm only one tournament is run in any iteration, the selection of the tournament is based on the index variable i, which shows the population member that is selected to compete in the next tournament. The algorithm begins by running N mutation
tournaments, once for each member of the population. Once this is completed the hyper-mutation tournament run to eliminate the worst member of the population. Then the index variable $i$, is reset to point to the first member of the population and the whole algorithm repeats. The evolution process halts when MAXEVALS evaluations are completed. Once these evaluations are completed it returns the best solution in the whole population.

2.3 Evolvable Hardware

Evolvable Hardware (EH) [17] is an emerging sub-specialty of Evolutionary Computation (EC) in which Evolutionary Algorithms are used to optimize the specific configurations of reconfigurable hardware devices. EH is similar to EC as applied to optimizing processes (I.E. the objective function measure performance of a device over an evaluation period), except that it often applied in the context of real-time evaluations of real hardware in actual environments. This places severe restrictions on tolerance to evaluating bad, and possibly catastrophic, candidate solutions; the amount of computational hardware available. It often skews the more normal relationship between candidate evaluation time and EA operation time. In EH work, the amount of time to evaluate solutions (either via simulation or in real hardware) is very long compared to the clock rates of the computers running the code. Thus, objective function evaluation become far more expensive and EH methods are often under extended pressure to extract maximum useful information from each candidate evaluation. The work of this thesis is an apt example of these restrictions and challenges in practice.

2.4 The Flapping-Wing Micro Air Vehicle (FW-MAV)

The FW-MAV considered here is similar to the FW-MAV called ROBOFLY developed by Wood et.al. at Harvard University. The ROBOFLY was the first insect-scale bio
mimetic flapping wing micro air vehicle to achieve take off and fly. This was achieved by constraining the aircraft to vertical translation on a pair of wires keeping the degrees of freedom for the vehicle down to one. The ROBOFLY was designed with a single bi-morph piezoelectric actuator to actuate both the wings simultaneously. But in the model considered here, the FW-MAV has two bi-morph piezoelectric actuators, one for each wing. The tangential motion of the tip of the actuator is converted to the basic motion of the wings constrained to the perpendicular plane to the central axis of the vehicle by means of a linkage. The linkage elements are designed such that impedance is matched between the wing and the actuator forces and also to amplify the small motion of the tip of the bi-morph strip into the large angular displacement of the wing in its stroke plane. Figure 8 shows the orthographic view of the Flapping wing micro air vehicle (FW-MAV).

![Figure 8: The FW-MAV Orthographic View][4]

From the orthographic view of the FW-MAV, the shape of the wing can be visualized, the triangular section which hangs down is responsible for the lift generated by the air resistance during the sweeping motion of the wings. The air resistance is maximized by limiting the rotational motion of the wings along the z axis as the wings sweep back and forth. When the wings rotate through an angle $\phi$, the air resistance
lifts the triangular segments of the wing to a limited angle of $\alpha$ radians under a base vector embedded in the plane of $\phi$ motion. Hence under very specific schedules for the wing beat one can achieve a controlled vehicle hover, translation and rotation. The complete dynamics and kinematics of the modified FW-MAV are explained in [18] and [19]. We assume that one can directly control wing angular position via the two actuators, as complete three dimensional simulations of the vehicle exist.

### 2.4.1 Altitude Control using the ACTC

Flapping-Wing Micro Air Vehicles are insect-sized vehicles which require forced and periodic inputs to drive the wings. These inputs to the wings being periodic need to have a higher wing beat frequency than the dynamics of the physical model itself as the real insect requires several wing beats to complete any single maneuver. The wing beat pattern in the flight of an insect does not vary dramatically from one beat to another, but there are recognizable changes when pairs of wing beats are compared. Oppenheimer et. al [18] have provided a feedback controller for hover under the main restriction that the FW-MAV moves along a pair of vertical wires. These wires reduce the degrees of freedom down to one by controlling the motion in the other five degrees of freedom in a three dimensional space. These wires hence control the effect of yaw, pitch, and roll in x, y, and z axes on the FW-MAV and also control the motion in y and z axes.

The above restrictions are very similar to the ones imposed for the Robofly. These restrictions helped the development of EAH concepts into the controller in the initial stages as it provided a simple environment to test initially. The feedback controller provided is called the Altitude Command Tracking Controller (ACTC). This feedback controller is set in line with the main oscillator and the Plant model of the FW-MAV. The Altitude Command Tracking Controller can be explained qualitatively based on the cosine oscillator, which drives the wings based on the real time values of altitude,
and velocity of the vehicle at any instant.

Figure 9: The Block Diagram of the Altitude Command Tracking Controller[4].

1. As shown in Figure 9, the top view of the model shows that the movement of the wings is along the Z axis. The physical movement of the wings is measured by ($\phi$) which is controlled directly by the cosine oscillator shown in Figure 10. The range of the cosine oscillator is set to [-1 to +1] radians, from which we can understand that one wing beat cycle starts with both the wings in the forward position at [-1] radian. This position is set to be a default hence becomes the start and end of a complete wing beat. The ACTC is allowed to adjust the frequency of the cosine wave only when the wings are at [-1] radians, in other words only once in one complete wing beat cycle. This control is achieved by the Cycle-ZOH block shown in Figure 10, where it is shown that it constantly receives the position of the wings and only allows the controller to change the frequency when the wings reach the full forward position [-1] radian. This lock on the controller to adjust the frequency does not keep the controller idle between wing beats. The Cycle-ZOH block can be eliminated once EAH methods are well trained and tested to work with the ACT Controller. The main purpose of the Cycle-ZOH block is to make sure that frequency changes occur at the beginning of a wing beat as, if the new frequency has a large
difference from the last wing beat frequency, it will have a adverse effect on the
FW-MAV.

2. The Altitude Tracking Controller provides frequency adjustments by computing
the force \( F_x \) from the control law that produces a second order response to a
given altitude command, \( x_{des} \).

\[
F_x(t) = m(-2\zeta_a \omega_a \dot{x} - \omega_a^2 x + \omega_a^2 x_{des} + g)
\]

This control law computes a force \( x_{des} \) which helps the vehicle to reach or move
towards the commanded altitude \( (x_{des}) \). The law derived in [5] is such that force
applied to the body over a complete wing beat cycle is calculated and is then
used to compute a frequency that will be suitable for the next wing beat cycle
to help the vehicle reach its desired altitude \( x_{des} \). In this law \( (F_x) \) is constrained
to be periodic, but it has been postulated that if the bandwidth of the tracking
law is much less than the wing beat frequency \( \omega_a \ll \omega \) then the cycle average
force can be specified. These two parameters are related by,

\[
W_{F_x} = \sqrt{\frac{2F_x}{\rho I_AC_L(\alpha)}}
\]

2.4.2 Split-Cycle Control

As the name suggests Split-Cycle control of the FW-MAV is due to the flapping
motion of the wings, such a flapping motion can be controlled by a periodic control
wave such as a sine or cosine to generate the basic flapping motion. Doman et.al., in
[5] and [18] provide the mathematical derivation of a controller that is able to control
the position of the wings with the help of Split-Cycle cosine. One complete cycle is
the motion of the wing starting from forward position ( \(+1 \) radian \) which can be
seen in Figure 9 as the angular displacement of the wings along the Z axis, there is a upstroke motion from +1 radian to -1 radian and a down stroke motion from -1 radian to +1 radian. The split cycle cosine wave is controlled by controlling the frequency in both the upstroke and the down stroke. Hence in this generation of the controller the envelop of the wing motions are defined by a cosine wave whose frequency is delayed or impeded by $\delta$ radians in the upstroke and is impeded or advanced in the down stroke to make sure that the complete wing beat cycle has a constant time or the time it would have taken if it were driven by a unimpeded cosine wave at the specified frequency. A Split -Cycle Cosine wave can be understood with the help of the following equations,

$$\sigma = \frac{\delta \omega}{\omega - 2\delta}$$  \hspace{1cm} (6)$$

$$\xi = \frac{-2\pi \delta}{\omega - 2\delta}$$  \hspace{1cm} (7)$$

$$\phi_u = \cos[(\omega - \delta)t]$$  \hspace{1cm} (8)$$

$$\phi_d = \cos[(\omega + \sigma)t + \xi]$$  \hspace{1cm} (9)$$

$\omega$ in the above equations is the frequency of the wings, $t$ is the time and $\delta$ is the parameter that controls the shape. As we can see there are two separate control equations, one for the down stroke and one for the upstroke. These deltas which are computed separately for each wing once for every wing beat cycle by the controller are aimed to produce the desired force for the lift and body moments. The advantage of having two separate deltas for the respective wings is that in the next few generations of the controller this split-cycle control can be used for various flight modes. In other words there are actually two specific parts of the cosine wave for each wing hence four
degrees of freedom to control the flight. Symmetrical control works for hovering, but when other flight modes are required each wing will have to be controlled separately. Control of the individual wings at different frequencies in the upstroke and down stroke to balance out the body torques generated are needed to achieve the desired flight pattern.

The Split-Cycle Cosine waves represented by the Equations (9) and (10) are stored in a lookup table called the Wave Table ROM in the hardware. This ROM becomes the search space for the Genetic Algorithm used. There are different Split-Cycle Cosine waves designed to be addressed by 256 different \( \delta \) values for the cosine basis function.

The Wave Table ROM was expanded to make available a few more basis functions which contain linear combinations of cosines during the process of this thesis. The addition of these new basis functions expanded the search space for the Minipop Algorithm, one might say that this might increase the learning time but when the effects were analyzed the new basis functions helped the EAH-Oscillator generate final waveforms that were never possible with the previous cosine wave, which have a direct effect on the lift produced by a individual wing in a single wing beat. A sample waveform generated by the EAH-Oscillator from the new set of basis functions can be seen in figure 11. These new set of combinations of the cosine wave are represented by the following equations,

\[
\cos(x) \tag{10}
\]

\[
\frac{\cos(x) + \cos(3x)}{2} \tag{11}
\]

\[
\frac{2\cos(x) + \cos(3x)}{3} \tag{12}
\]
\[
\frac{4\cos(x) + \cos(3x)}{5}
\]  \hspace{1cm} (13)

Initially the Wave Table ROM was set at 256x256, with the addition of these new basis functions the size of the ROM was increased to 1024x256. Each of the new basis functions have 256 split-cycle versions of themselves in the impeded and advanced state which creates the size of the new ROM as 1024x256. These four basis functions present in the new Wave Table ROM can be seen in Figure 11.
3 Methodology and Model

3.1 Introduction

In this chapter the details regarding the model under consideration are discussed in detail, in the model overview section, which gives the readers a clear idea about the FW-MAV, its physical structure is explained and related to the mathematical model of the controller. Here the parameters that have been used to configure the model for the experimental stage are tabulated and explained to the reader.

The section 3.3 the architecture of the Non-EAH based controller which was initially designed by Gallagher [4] to drive the wings of the FW-MAV is briefly explained, to give the reader’s a clear picture about the architecture and its different modules. The controller was redesigned to make it a Evolvable Adaptive Hardware. These modules in the new design architecture is briefly explained in section 3.4.

The Minipopulation algorithm was selected as suitable genetic algorithm for this
model in consideration and was modified to adapt it to the Flapping-Wing Micro Air Vehicle project. In section 3.5 the modified version of the Minipop algorithm which was used is explained as it is necessary to understand the working of the GA used, to understand the simulation and results of this thesis. The last section of this chapter gives a detailed description of the simulation process and the simulation setup which was used to tune the Minipop Algorithm to better suit the control problem in hand in the initial stages of the controller’s design.

3.2 Model Overview

Although briefly discussed previously, the complete model of the FW-MAV will be explained in this section. As mentioned in the earlier chapters this FW-MAV is based on the Harvard Robofly[1] which had its first controlled flight with a single degree of freedom. The general assembly of the proposed FW-MAV for which this controller is being designed [5] can be explained by looking at Figure 12,

From Figure 12 we can note that there are two Bimorphic Piezoelectric Actuators which will power each wing through separate mechanical linkages. The two wings are attached to these linkages with the help of a rotation joint which allows the wing to rotate by an angle $\alpha$. There is a weight placed in the fuselage to help in the balancing of the vehicle. Keeping in mind this physical model the following fixed values were assumed for the simulated model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Mass</td>
<td>60x10^{-6}</td>
<td>Kg</td>
</tr>
<tr>
<td>Vehicle Width</td>
<td>4x10^{-3}</td>
<td>m</td>
</tr>
<tr>
<td>Vehicle Height</td>
<td>11x10^{-3}</td>
<td>m</td>
</tr>
<tr>
<td>Vehicle Depth</td>
<td>1x10^{-3}</td>
<td>m</td>
</tr>
<tr>
<td>$r_{wing}$</td>
<td>15x10^{-3}</td>
<td>m</td>
</tr>
<tr>
<td>$\tau_{wing}$</td>
<td>4x10^{-3}</td>
<td>m</td>
</tr>
<tr>
<td>$b_{wing}$</td>
<td>3x10^{-3}</td>
<td>m</td>
</tr>
</tbody>
</table>
The constants considered in this model are listed in the table below, and the units for the various calculated parameters are also tabulated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gravity</td>
<td>g</td>
<td>9.82</td>
<td>m/s²</td>
</tr>
<tr>
<td>Wing Moment of inertia</td>
<td>Iₐ</td>
<td>9.35 x10⁻¹⁰</td>
<td>m⁴</td>
</tr>
<tr>
<td>Air Density</td>
<td>ρ</td>
<td>1.225</td>
<td>Kg/m³</td>
</tr>
<tr>
<td>Wing Lift Stop</td>
<td>α</td>
<td>π/4</td>
<td>rad</td>
</tr>
<tr>
<td>Controller damping ratio</td>
<td>ζα</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Coefficient of lift</td>
<td>C_L(α)</td>
<td>1.34222</td>
<td></td>
</tr>
</tbody>
</table>

The parameters from tables 2 and 3 are used to calculate the various forces being generated instantaneously during a simulation of the FW-MAV in flight. The model
of the Altitude Command Tracking Controller in C was provided to help simulate the flight of the FW-MAV for the Modified Minipop algorithm. This model contains standard set of function modules which help in doing housekeeping functions as in initializing, printing present status of the FW-MAV to the screen, file, set a value in the structure, get a value, and clear the structure. The main modules which does the calculation of the forces, frequency and other calculations which are preformed in the ACTC are explained in the remaining part of this section. The structure of the wing used to calculate the generated forces is shown in Figure 13.

![Figure 13: The Wing Design used for Force Calculations](image)

1. **Calculation of the Lift Coefficient**: This function calculated the instantaneous lift coefficient for the wings, this is represented as $C_{lift}$. This coefficient is required for the calculation of lift produced by each wing, which is used to calculate the instantaneous forces produced by the FW-MAV. The input for this function is the angle $\alpha$ in radians which gives us the angle of the wing from the vehicle structure. The angle $\alpha$ is converted to degrees and the relative angle...
in degrees is calculated and transformed back to radians. The equations for the calculations in this function are as follows [5],

\[
\theta_{\text{deg}} = (\alpha_{\text{rad}} \times 180) / \pi \tag{14}
\]

The transformed angle of the wing in degrees,

\[
\theta_{\text{deg}} = (2.13 \times \theta_{\text{deg}}) - 7.2 \tag{15}
\]

\[
\theta_{\text{rad}} = (\theta_{\text{deg}} \times \pi) / 180 \tag{16}
\]

The coefficient for the lift is got from the following calculation,

\[
C_{\text{lift}} = (0.225 + 1.58 \sin(\theta_{\text{rad}})) \tag{17}
\]

2. Calculation of the Lift Produced : The lift by the wing is then got from the best fit estimate of the quasi-steady lift coefficient from the last function and a few constants, the constants considered here are the air density \(\rho\), and the wing moment of inertia \(I_A\), the equation is given as follows,

\[
K_{\text{lift}} = (\rho / 2) \times C_{\text{lift}} \times I_A \tag{18}
\]

3. Force Calculations : In this function the instantaneous forces generated along the X-axis are calculated. The forces acting on the FW-MAV calculated here are only the ones along the X-axis, the vehicle will not be subjected to spinning is another assumption, and the force calculated here is in newtons. There are three forces acting on the vehicle, they are
(a) Force from gravity on the body.

\[ F_{\text{gravity}} = (\text{Vehicle mass} \times \text{Gravity}) \]  \hspace{1cm} (19)

(b) Force generated by the right wing. To simulate the broken wing of the vehicle the force generated by the right wing is cut by half.

\[ F_{\text{RWing}} = (0.5 \times K_{\text{Wing}} \times (\text{Wing Angular Velocity})^2 \]  \hspace{1cm} (20)

(c) Force generated by the left wing.

\[ F_{\text{LWing}} = (K_{\text{Wing}} \times (\text{Wing Angular Velocity})^2) \]  \hspace{1cm} (21)

Once these separate forces are calculated the total force is calculated by the sum of the forces generated by the wings, which is subtracted from the gravitational force on the body. This total downward force is returned from this function.

4. **Cycle Averaged Force (Actual)**: The cycle is quantized into step size of 256, hence the actual cycle average force is calculated by dividing the generated forces by 256.

5. **Cycle Average Force (Estimated)**: The estimated cycle average force is got from the following equation,

\[ F_{\text{est}} = (\theta^2 \times \rho \times I_A \times C_{Lift}) \]  \hspace{1cm} (22)

6. **Vehicle Update**: This function updates the vehicle structure with the acceleration produced by the vehicle, in turn performs Euler’s method of integration
to calculate the velocity of the vehicle along the X-Axis and the instantaneous position of the vehicle in the air.

\[
FreeBodyForces = (F_{LWing} + F_{RWing} - F_{Gravity}) \tag{23}
\]

\[
BodyAcceleration = \frac{FreeBodyForces}{VehicleMass} \tag{24}
\]

\[
x_{vel} = x_{vel} + (x_{acceleration} \times stepsize) \tag{25}
\]

\[
x_{pos} = x_{pos} + (x_{vel} \times stepsize) \tag{26}
\]

### 3.3 Non-EAH Based Controller Architecture

Gallagher presented a digital design for a basic controller and brought to light the gaps present in the design which make the controller adaptable for the evolution process to be integrated. In this section I will be briefly explaining the initial design for the Split Cycle Cosine module of the controller which is needed to understand the design for Evolvable Adaptive Hardware proposed by Gallagher explained in the next section. The architecture for the Split Cycle Cosine module of the Non EAH Controller is shown in Figure 14.

The Split Cycle Cosine Module is the oscillator part in the controller for the vehicle, the main inputs and outputs for this module are listed in Table 4, 5.

Let me start by giving a brief explanation of the Split-Cycle cosine Oscillator. This core is designed around a 256 X 256 element lookup table. Each element in this lookup table contains 8-bit fixed precision data for a split cycle cosine wave with a particular delta value. There are 256 delta values considered here as the delta ranges
Figure 14: Split-Cycle Cosine Module for the Non EAH Controller.[4]

Table 4: Input parameters for the Non-EAH Oscillator module.

<table>
<thead>
<tr>
<th>Input Parameter</th>
<th>Input Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay delta for the Left wing Cosine</td>
<td>$\delta_L$</td>
</tr>
<tr>
<td>Delay delta for the Right wing Cosine</td>
<td>$\delta_R$</td>
</tr>
<tr>
<td>Frequency</td>
<td>$\omega$</td>
</tr>
<tr>
<td>Acknowledge</td>
<td>Ack</td>
</tr>
<tr>
<td>Request</td>
<td>Req</td>
</tr>
</tbody>
</table>

between (-1.5, 0.38) with a step size of 0.007344, hence the lookup table holds data for 256 delayed or impeded cosine waves. This makes it possible to generate any split cycle cosine wave within the delta range of (-1.5, 0.38) to drive the individual wings. This selection process is made possible with the help of a few more modules which
Table 5: Output parameters for the Non-EAH Oscillator module.

<table>
<thead>
<tr>
<th>Output Parameter</th>
<th>Output Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual Wing Position command for the Left wing</td>
<td>$\phi_L$</td>
</tr>
<tr>
<td>Actual Wing Position command for the Right wing</td>
<td>$\phi_R$</td>
</tr>
<tr>
<td>Mode Select</td>
<td>Mode</td>
</tr>
<tr>
<td>Reset line</td>
<td>Reset</td>
</tr>
</tbody>
</table>

are shown in the architecture. The lookup table was expanded to hold a few more basis functions (3 new basis functions). This had a direct effect on the size of the Wave Table ROM which increased to 1024x256. The new lookup table still contains 256 delta values but now it has 256 delta values for each of the four basis functions. the delta range remains the same and the step size also remains the same but the table is expanded to hold data for 256 delayed or impeded waves for the cosine wave and three more basis functions. The ACK and REQ lines are provided to ask for the next set of values for $\delta_L$, $\omega$ and $\delta_R$; this request and acknowledge takes four cycles to communicate with a higher controller, the Mode pins are provided for any future expansion to add more modes to the wing control. This explains the inputs and outputs into the oscillator module, now the modules that are used in the oscillator for the selection process are individually explained below,

1. **DL, DR Registers**: These two registers act as a receiving point for the $\delta_L$, $\omega$ and $\delta_R$ 8-bit fixed precision binary data which is calculated by a higher controller to be in the range of $[-1, 1]$. These registers are based on standard register design with load, clear and select lines but are 8bit registers in this case as that is the requirement in this model.

2. **PL, PR Registers**: These registers are similar to DL and DR registers but hold the $\phi$ values for the left and right wings. In this case these registers are read from instead of being written into like the DL and DR registers. PL and PR registers get their data from time to time from the lookup table depending
on the clock and the selection addresses. The data in these registers can drive the wings when passed through a digital to analog converter. But for simulation purposes the digital data is sufficient.

3. Multiplexers (M1, M2): Multiplexers are used to select between a number of inputs, so that data is acquired sequentially or sequential data is distributed correctly to various outputs in the right order. In this model M1 functions as a multiplexer with 2 inputs (DL, DR) and M2 functions as a multiplexer with 2 outputs (PL, PR). M1 and M2 are selected by the micro-controller present and also commanded as to which register to read or write.

4. Trace Time Count (TTC): To address any one element of the 256X256 lookup table a 15 bit address is required, 8 bits which select the row and 8 bits which select the column. TTC is a register which holds the lower 8 bits of the address to the lookup table. These 8 bits in the TTC are incremented by the micro-controller with respect to the internal clock of the micro-controller. With the increase in the size of the lookup table there is no effect on the TTC as it generates the lower 8 bits and points to the columns of the lookup table, however the 15 bit address increased to 17 bits due to the expansion in size.

5. Delay Counter (DC): This counter is used to introduce a delay in between the increments of the TTC register. This is achieved by a 19 bit register whose 8 bits from the MSB side are compared with the value from the Omega Frequency register which is input to the comparator with its positions inverted. When these two values are equal TCC is incremented and DC is reset to zero, if the comparator returns a zero saying the values are not equal then DC is incremented with reference to the clock ticks of the micro-controller present.

6. Omega Frequency (OF): OF is also a 8 bit register that receives 8 bit data
which represents the base beat frequency provided from the higher controller $\omega$. Here to convert the frequency $\omega$ to the respective period the bit positions are reversed and the bit at the 1’s position before the inversion to be zero. This in-place conversion method used is a coding trick used to convert frequency to period.

7. **Binary Comparator**: This is a basic comparator which compares the 8 bits from the MSB from the DC register and 8 bits with inverted position and returns ’0’ if not equal and returns ’1’ if they are equal.

8. **Micro Controller**: With all these above registers and other modules present a micro-controller is used to control these modules at the right time, provide a clock tick for the counters which are present on the oscillator. The Figure 15 shows the Algorithmic State Machine (ASM) developed for the micro-controller. This shows a requirement for a dedicated ROM which is yet to be designed for the oscillator. The ASM shows a number of register transfer language directives which refer to the various registers and control inputs present in the oscillator.

This briefly covers all the modules and their functionality to help control the various modules of the Non-EAH oscillator.
3.4 EAH Based Controller Architecture

The EAH based oscillator implements almost the same architecture as the Non EAH based oscillator except for two new modules which implement a different way of
lookup in the Split-Cycle table. The working of the EAH oscillator can be better explained after looking at the architecture and understanding the two new modules incorporated into this architecture. The Figure 16 shows the architecture for the EAH based Split Cycle Cosine Oscillator module.

\[\text{Figure 16: Split-Cycle Cosine Module for the EAH Based Controller.}[4]\]

The two new modules added in this architecture are the Shuffle LUT RAM and the Temporary Phi.

1. **Shuffle LUT RAM**: The $\delta$ values which are provided by the higher controller actually map to a particular delta setting in the wave table ROM, with the
introduction of this module Gallagher introduces a bridge to fill the gap and introduce the evolutionary search procedure. This shuffle LUT RAM is a lookup table with 8 columns and 256 rows, the rows represent the 256 different delta’s available for the split cycle cosine wave, and the 8 columns are split as 4 lookup’s for each wing. There are some hard limitations followed in the EA so that the resulting waveform does not have any sharp changes in frequency which would be harmful for the vehicle. These four separate lookup addresses are processed by the other new module Temporary Phi. This 256 X 8 lookup table consists of 10 bits to map the requested evolved delta to the expanded Wave Table ROM. This expansion of the ROM was made to include the three new basis functions. The corresponding change in the micro-controller ASM was required, these changes can be incorporated in the later stages of digital design.

2. **Temporary Phi (TP)**: As in the shuffle LUT RAM the number of readings for each wing were increased to 4, but only one reading can be fed to the output register PL, PR to drive the wing through the DAC. Hence the use for this module comes into picture, this module acts as a accumulator for the running sum of the 4 readings, to help accommodate this sum the register is 10 bits wide. Once the running sum for all four lookup’s is calculated an average of this value is calculated and fed into the corresponding register by the multiplexer. This averaging technique also helps maintain a smooth shape of the wave.
Figure 17: A Better Look at the Shuffle LUT and Wave Table ROM

3.5 Modified Minipopulationary Algorithm

The version of the Minipop Algorithm used for this model is constantly being modified to adapt to the advances in the design process of the controller. The modified version of the algorithm makes use of elite drift tournament instead of the standard hyper-mutation tournament, where after one complete sweep of the population members hyper-mutation is given a chance to give the algorithm a new member from an extreme end of the search space. Elite drift is not explained in chapter 2, hence its function in the Modified Minipop Algorithm
can be explained as an operation which eliminates the deepest pit or the worst population member with the best population member for that sweep, hence flattening the deepest valley among the population members. In the tournament between the populations for survival the elite drift used for this model helps the higher controller to be in control at all times and never allow the evolutionary algorithm to take control of the vehicle. The modified version of the minipop algorithm is shown Figure 18.

```
1  Generate initial population
2  Initialize population (Randomize or initialize to wave table ROM)
3  While (not finished)
   Begin
   5    for (n=0 ; n < population_size ; n=n+1 )
   6       Begin
   7          generate mutant member
   8          for (m = 0 ; m < flaps_per_eval ; m=m+1 )
   9             Begin
   10                evaluate best_population_member
   11             End
   12          End
   13          for (m = 0 ; m < flaps_per_eval ; m=m+1 )
   14             Begin
   15                evaluate population_member(n)
   16             End
   17          End
   18          for (m = 0 ; m < flaps_per_eval ; m=m+1 )
   19             Begin
   20                evaluate mutant_member
   21             End
   22          if (mutant is superior to original)
   23             replace population_member[n] with mutant_member
   24       End
   25  find best_population_member
   26  find worst_population_member
   27  if (random_number < chance of elite drift )
   28     replace worst_population_member with best_population_member
   End
```

Figure 18: Pseudocode for the Modified Minipop Algorithm.

The modified version of the minipop algorithm is a simple but specific version of the original minipop algorithm [3]. Gallagher modified the minipop algorithm to effectively suit the requirement of the FW-MAV. The main changes or modi-
fications include the adaptation of the algorithm to address the LUT encodings instead of random bit-strings as in the standard algorithm. Hence the beginning of the algorithm is from a known set of data rather than randomly generated data. The effect of starting from a random set of data versus initialized set of data (Initialized to the Wave Table ROM) has a significant effect and needs to be checked each time the design is modified.

The Modified Minipop used for the simulations has a modified selection process which selects the best performing population member for the FW-MAV model. A new criterion of the number of flaps the wings could execute in every evaluation was introduced to compute the fitness of a population member for the selection process, which is a tournament as explained in chapter 2. This parameter flaps per eval is user specified in the simulations and represents the number actual wing beats of the FW-MAV. The FW-MAV model is evaluated based on its performance for the population member, at the end of each simulation the model might be in a good position or it might not have learned during the simulation. Hence when the next population member gets control over the model it might either have very little to learn or it might have to learn rapidly to meet the fitness criteria. To avoid a condition where the previous population member was not able to learn and the model is in a very bad position the control of the FW-MAV is passed to the best population member for a complete evaluation cycle, where the model is subjected to the similar process as explained above but the difference is that the model will be in a good starting position when control is passed to the next population member and that this run is not subjected to the fitness test, this can be observed in the pseudo code shown. Simulations were run to see the performance of the model at various flaps per evaluation to help analyze the right value which helps the model perform optimally. The
hyper-mutation tournament in the normal Minipop algorithm was replaced with the elite drift operation. The drift operation depends on a probability which can be user specified, which keeps the searches in a healthy area. The main while loop for the learning cycle in the figure shows that it never terminates until a mission is completed and the vehicle stops, but for simulation purposes this is changed to stop when the maximum number of evaluations are exceeded or the evaluation function is satisfied. In case the evaluation function was the vehicle with the broken wing tracking a specified hover height and holding the displacement with minimal error. The Evaluation function used in this thesis is explained in the next section along with the fitness function and simulation setup.
4 Simulation Setup and Performance Analysis

4.1 Simulation Setup

The Minipop Algorithm has two main parameters which directly affect the learning process, they are the population size and the mutation rate for the algorithm. One simple mode of the FW-MAV model was considered for which tests were simulated for all the possible settings, to analyze and hence find the best parameter settings for this model.

From EAH architecture for the split-cycle cosine oscillator we can see that the Shuffle LUT ROM has 8 columns that is 4 lookup’s for each wing and 256 rows which refer to the waveforms associated with the delta value ranging from (-1.5 to +0.38) with a step size of (0.007344). Tests run on the Non-EAH Oscillator showed that the row 204 functioned perfectly for the hover condition of the FW-MAV. The introduction of the Shuffle LUT ROM made available the complete split-cycle cosine wave table for evolution of the cosine wave. Hence the 256 rows represented various flight modes and row ”204” represents ”Hover”. In the model considered in this thesis only the row number 204 is subjected to the learning algorithm as the case considered here is a FW-MAV model with a broken wing to hover at a user specified height.

The following table shows the initial settings for the first condition of finding a suitable mutation rate for the Minipop algorithm. Mutation rate as explained in Chapter 2 is the limit for the number of bits the in the genome under consideration can be subjected to mutation. This limit does not hard limit the EA to mutate the specified number of bits, but a random number is chosen between 0 and the specified mutation rate. Hence this sets the upper limit for the random number generator which can have a maximum of 64 bits as this is
the genome size. The effect of varying this limit is tested and analyzed. The flaps per evaluation parameter is the number of times a population member is tested in a simulation, the result of each evaluation for that population member is accumulated and an average of these tests is considered for the fitness of the population member.

Table 6: Initial settings for Mutation Rate Sweep

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Evaluations</td>
<td>4000000</td>
</tr>
<tr>
<td>Target Height for Hover</td>
<td>1.0 m</td>
</tr>
<tr>
<td>Flaps per Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>Population Size</td>
<td>8</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>4, 8, 16, 32, 64</td>
</tr>
<tr>
<td>Genome length</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

The Minipop Algorithm is aimed at addressing small population sizes but there is a specific population size for every implementation. To find the right population size for this controller, simulations were run for various population sizes. Here the mutation rate was set at 75

Table 7: Initial settings for Population Size Sweep

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Evaluations</td>
<td>4000000</td>
</tr>
<tr>
<td>Target Height for Hover</td>
<td>1.0 m</td>
</tr>
<tr>
<td>Flaps per Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>Population Size</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>32</td>
</tr>
<tr>
<td>Genome length</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

A mutation rate sweep and a population size sweep was done, since these are parameters that compose the same algorithm and the effect of one is significant on the other and vice versa, a complete sweep for both the parameters was designed to help better analyze the effects of these parameters in a combined
manner on the learning of the controller. The below table shows the parameter settings for these simulations.

Table 8: Initial settings for Population Size and Mutation Rate Sweep

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Evaluations</td>
<td>4000000</td>
</tr>
<tr>
<td>Target Height for Hover</td>
<td>1.0 m</td>
</tr>
<tr>
<td>Flaps per Evaluation</td>
<td>50</td>
</tr>
<tr>
<td>Genome length</td>
<td>64 bits</td>
</tr>
</tbody>
</table>

Table 9: Combination list for the Population Size and Mutation Rate Sweep

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Population Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
<tr>
<td>8</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
<tr>
<td>16</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
<tr>
<td>32</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
<tr>
<td>64</td>
<td>2, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32</td>
</tr>
</tbody>
</table>

Table 8 shows the parameter settings for the simulations and table 9 shows the combinations of the population size and mutation rate, simulations were run for each of these combinations and the results were analyzed to find a suitable setting for the population size and mutation rate for the Minipop Algorithm. The data for these results helped to prove a significant result for the EA, the parameter flaps per eval was considered to be significant as it sets the number of evaluations each of the population member is subjected to, before their respective fitness values are calculated. In other words this parameter tests the consistency of the performance of that population member. The table below shows the parameter settings for the simulations run by varying the flaps per eval parameter.
Table 10: Parameter settings for Flaps per Evaluation Sweep

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum Evaluations</td>
<td>4000000</td>
</tr>
<tr>
<td>Target Height for Hover</td>
<td>1.0 m</td>
</tr>
<tr>
<td>Flaps per Evaluation</td>
<td>40, 50, 60</td>
</tr>
<tr>
<td>Genome length</td>
<td>64 bits</td>
</tr>
<tr>
<td>Population Size</td>
<td>Refer Table 9</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>Refer Table 9</td>
</tr>
</tbody>
</table>

The design of the final test pattern to analyze the effects of the population size, mutation rate and flaps per evaluation can be visualized by looking at Figure 19.

Figure 19: Test Pattern Design
4.2 Assessment Parameters for Altitude Tracking

The aim to tune the Minipop algorithm leads to define the ways in which the significant performance of the algorithm in response to a single or multiple parameters can be assessed. The test results in this thesis were assessed mainly on two criterion’s which can be addressed in a related manner as the Time to Achieve a Acceptable Solution. Relating this to the datasets collected from the simulations we can address the assessment in two ways.

(a) **Learning Time**: The simulation time taken for the algorithm to learn is measured in seconds. To facilitate this calculation the period for the input waveform is calculated, from which a related step size is found and this step size is used to update the time taken for the simulation. This time is the complete time taken for the algorithm to learn and satisfy the fitness criterion of reaching the specified height, which in this case is 1.0 meters. This time taken is updated each time the vehicle structure is accessed to calculate forces generated by the wings in every step size of the cosine input. Hence the learning time which is relatively minimum, when compared to the other settings of the algorithm will be considered for the selection of the population size and mutation rate parameters for the algorithm.

(b) **Fitness Score**: To calculate the fitness score, the algorithm is run for a preset number of evaluations for each population member and the average error score for these set of evaluations is considered as the fitness value for that population member. This error score is an absolute value of the difference between the height achieved by the vehicle using the algorithm and the required height of 1.0 meters. This error score is the main criterion
on which the tournament for the population members and their mutated versions is held. In this algorithm, the learning is halted after the following steps,

i. Every population member has received a chance to compete with its mutant version.

ii. The best population member is selected based on how small its error score is, compared to the other members.

iii. The error score of the best population member is less than 0.001 meters. In other words sub-millimeter precision is achieved by the vehicle.

The analysis of the data based on the above explained assessment parameters for the individual sweeps of the parameters considered, are presented and explained in the next chapter.

4.3 Performance Assessment of the Modified Minipop Algorithm

This Chapter provides the analysis for the data collected while simulating the various settings for the modified Minipop Algorithm. There were three main factors which were considered for tuning purposes of the algorithm as explained in the earlier sections. These three factors were analyzed by running simulations for each combination as tabulated in the earlier section.

4.3.1 Analysis for the Mutation Rate Sweep

Simulations were run for the different settings of the mutation rate parameter, the data collected was for a fixed population size of 8 and a fixed
flaps per evaluation of 50.

Table 11: Learning Times for Varying Mutation Rate

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1448.977</td>
<td>430.607</td>
</tr>
<tr>
<td>8</td>
<td>823.313</td>
<td>232.952</td>
</tr>
<tr>
<td>16</td>
<td>605.191</td>
<td>209.233</td>
</tr>
<tr>
<td>32</td>
<td>540.434</td>
<td>231.653</td>
</tr>
<tr>
<td>64</td>
<td>761.994</td>
<td>491.883</td>
</tr>
</tbody>
</table>

Table 12: Fitness Values for Varying Mutation Rate

<table>
<thead>
<tr>
<th>Mutation Rate</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.00063888</td>
<td>0.000245262</td>
</tr>
<tr>
<td>8</td>
<td>0.00065596</td>
<td>0.00023565</td>
</tr>
<tr>
<td>16</td>
<td>0.0006677</td>
<td>0.000230645</td>
</tr>
<tr>
<td>32</td>
<td>0.000681005</td>
<td>0.000224843</td>
</tr>
<tr>
<td>64</td>
<td>0.000695386</td>
<td>0.000222272</td>
</tr>
</tbody>
</table>

The above tables provide the analyzed data for the 5 different settings of mutation rate. The first table shows us the mean and standard deviations for the 5000 simulations run for each of the five settings. From the analyzed data we can see that the mutation rate setting of 32 bits took the least learning time when compared to the other four cases. The standard deviation calculated for the 5000 simulations compares the learning times, from the table we can see that mutation rate setting for 16 bits has got the least standard deviation compared to the other settings. No simulation runs were found to be terminated due to exceeding the maximum number of evaluations (4000000), where the algorithm never learned. The other table shows us the analyzed data for the fitness values obtained from the simulations. Each simulation is terminated when the vehicle achieves a sub-millimeter precision value lower than (0.001).
the learning time plot we can observe the relationship between the mutation rate and learning time, this plot proves that mutation as a genetic operator has a significant effect on the learning times of the Minipop Algorithm. The minimum mutation rate of 4 shows that the time required for the algorithm is almost twice that compared to mutation rate 8. There is a linear drop in the learning time as the mutation rate is increased. But the setting of mutation rate 64 bits conveys a different picture, that the effect of allowing the algorithm to mutate all the bits or in other words opening up the search space completely increases the time required for the algorithm to learn. The consistency increases for higher mutation rates.

The above presented results tell us the mutation rate of 32 is optimal to obtain a good learning time, but to increase our confidence in above results T-Tests were conducted among the mutation rates considered. The T-Test results are tabulated in the following table, from these results we observed that the data obtained was statistically significant and the confidence level for all the T-Tests is found to be (96-99) percent. These T-Tests prove the hypothesis that there are no significant similarities among the data obtained for the five mutation rates.

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.022763667</td>
<td>-0.009755465</td>
<td>-0.000980232</td>
<td>0.004000484</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0.001487703</td>
<td>0.004434811</td>
<td>0.005771409</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>0.004479371</td>
<td>0.013087579</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td>-0.006230323</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 20: Data Analysis for Mutation Rate Sweep
4.3.2 Analysis for the Population Size Sweep

Simulations for the vehicle were run by varying the population size, the mutation rate was kept a constant at 32 bits for this sweep. The value for the mutation rate was picked from the results of the last section where a detailed analysis of the data from the mutation rate sweep was conducted. The following table shows the results from the data collected from the simulations run by varying the population size.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>918.919055</td>
<td>809.8489809</td>
</tr>
<tr>
<td>4</td>
<td>446.8579567</td>
<td>272.4847151</td>
</tr>
<tr>
<td>6</td>
<td>471.7117099</td>
<td>247.9147834</td>
</tr>
<tr>
<td>8</td>
<td>540.4344281</td>
<td>231.653434</td>
</tr>
<tr>
<td>10</td>
<td>642.4655782</td>
<td>254.220836</td>
</tr>
<tr>
<td>12</td>
<td>750.9346969</td>
<td>277.5551236</td>
</tr>
<tr>
<td>14</td>
<td>862.1266499</td>
<td>288.6662497</td>
</tr>
<tr>
<td>16</td>
<td>985.7302835</td>
<td>328.4356653</td>
</tr>
<tr>
<td>18</td>
<td>1113.615208</td>
<td>348.5672224</td>
</tr>
<tr>
<td>20</td>
<td>1243.612369</td>
<td>384.5932733</td>
</tr>
<tr>
<td>22</td>
<td>1382.435686</td>
<td>424.7031303</td>
</tr>
<tr>
<td>24</td>
<td>1523.707534</td>
<td>466.3976904</td>
</tr>
<tr>
<td>26</td>
<td>1644.842646</td>
<td>494.0297053</td>
</tr>
<tr>
<td>28</td>
<td>1798.993977</td>
<td>541.5175679</td>
</tr>
<tr>
<td>30</td>
<td>1944.63638</td>
<td>575.8411045</td>
</tr>
<tr>
<td>32</td>
<td>2087.767285</td>
<td>622.4419105</td>
</tr>
</tbody>
</table>

The above tables provide the analyzed data for the simulations run by varying the population size for the Modified Minipop Algorithm. Similar to the analysis made in the previous section the data for all the 5000 runs for each of the 16 population sizes considered for testing is conducted, from the results we can observe few things which show the behavior of the algorithm in this project of the FW-MAV. These results can be better
Table 15: Fitness Values for Varying Population Size

<table>
<thead>
<tr>
<th>Population Size</th>
<th>Mean Value</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.00062994</td>
<td>0.000248764</td>
</tr>
<tr>
<td>4</td>
<td>0.000667438</td>
<td>0.00023557</td>
</tr>
<tr>
<td>6</td>
<td>0.000675477</td>
<td>0.000229232</td>
</tr>
<tr>
<td>8</td>
<td>0.000681005</td>
<td>0.000224843</td>
</tr>
<tr>
<td>10</td>
<td>0.000683938</td>
<td>0.000225872</td>
</tr>
<tr>
<td>12</td>
<td>0.00068676</td>
<td>0.000221653</td>
</tr>
<tr>
<td>14</td>
<td>0.000688762</td>
<td>0.000220917</td>
</tr>
<tr>
<td>16</td>
<td>0.000684425</td>
<td>0.000227335</td>
</tr>
<tr>
<td>18</td>
<td>0.000686555</td>
<td>0.000221275</td>
</tr>
<tr>
<td>20</td>
<td>0.000687635</td>
<td>0.00022064</td>
</tr>
<tr>
<td>22</td>
<td>0.000687784</td>
<td>0.000219302</td>
</tr>
<tr>
<td>24</td>
<td>0.000687319</td>
<td>0.000221318</td>
</tr>
<tr>
<td>26</td>
<td>0.000695047</td>
<td>0.000217013</td>
</tr>
<tr>
<td>28</td>
<td>0.000691967</td>
<td>0.00021775</td>
</tr>
<tr>
<td>30</td>
<td>0.000692151</td>
<td>0.000220619</td>
</tr>
<tr>
<td>32</td>
<td>0.000695337</td>
<td>0.000216844</td>
</tr>
</tbody>
</table>

understood by looking at the following plots.

The effect of varying the population size on the learning time of the algorithm is clearly visible from the plot. This relationship can be explained by saying that increasing the population size has a linearly increasing effect on the learning time of the Minipop Algorithm. The population size to have the least learning time can be seen as 4, there is a small effect on the consistency but this effect can be overlooked and considered as a constant. The effect of varying population size on the fitness value in the sub-millimeter precision values is very little. To further improve our confidence on the above results T-Tests were conducted to test the significance of the above results. The results of the T-Tests are tabulated below, these results show us that the data is statistically significant with a confidence level of (96-99) percent. These T-tests help to prove the hypothesis that
Figure 21: Data Analysis for Population Size Sweep
there are no significant similarities in the data obtained by varying the population size.

![T-Test Results for Population Size Sweep](image)

Figure 22: T-Test Results for Population Size Sweep
4.3.3 Analysis for the Combined Sweep

A complete sweep of the mutation rate and the population size for the Minipop Algorithm would cover all the 5 mutation rates for every one of the 16 population sizes. This brings total of 80 unique simulations with 5000 runs each. The analysis of such a sweep will aid to the clear understanding of the effect of these parameters on the Modified Minipop Algorithm. To prove the significance of such a parameter sweep T-Tests would be very tedious and time consuming, hence the ANOVA test was conducted. The following Figure shows a surface plot of the analyzed data for learning times and the fitness values for a fixed flaps per evaluation parameter of 50 evaluations per population member.

Figure 23: Surface Plot for Combined Sweep of Population Size and Mutation Rate

The Surface plots seen in Figures 23 and 24 represent the learning time taken for a variation in both the Mutation Rate and the Population Size for the flaps per evaluation set at 50 evaluations per population member. We can observe the variation in the performance of the Minipop Algorithm or these ranges. The observations can be explained as follows,

i. The results for the learning time for the Minipop Algorithm to learn
ii. The main goal is to find the parameter combination which can yield the lowest learning time, from the surface plot we can see that the combination \([ (\text{pop-size}), (\text{mut-rate}) ]\), \([4,16]\) and \([4,32]\) have best learning times for this sweep.

iii. From the Figure 24 we can also observe that area of \([8,16]\) \([8,32]\) \([10,16]\) \([10,32]\) is pretty flat. This area has learning times in the range 550-700 seconds, which tells us that these combinations are the next best.

iv. A linear rise in the learning time as the population size increases can be observed, this rise is smooth and tells us that population size has a significant effect on the learning time of the algorithm. As there are two parameters involved here, the mutation rate has almost a constant effect as the population size increases, the curve we saw in Figure 20 which represents the effect of mutation rate on the algorithm holds its shape as the population size is varied.

v. All the above observations are backed by the surface plot of the
standard deviations for this sweep. This plot shows that the area for population size 8, 10 in the middle is pretty flat representing the minimum deviation seen in the 5000 runs. Also the effect of mutation rate remains almost constant and the effect of population increases the deviation observed linearly as the population size is increased.

vi. One observation in these plots is that a very small search space (2) with a maximum mutation rate has a significant effect on the learning time for the Minipop Algorithm. The standard deviation observed for this setting is maximum representing a very low consistency in the 5000 runs.

![Surface Plot of Fitness Value for Combined Sweep of Population Size and Mutation Rate](image)

Figure 25: Surface Plot of Fitness Value for Combined Sweep of Population Size and Mutation Rate

The Figures 25, 26 show the plots for the fitness values achieved and there standard deviations, by looking at these surface plots we can see that the fitness values achieved at consistent over the 5000 runs. There is a little uneven surface seen for the fitness values that were analyzed, from the graph it is also observed that the fitness value of (0.001) is satisfied. The observed uneven surface can be considered significant when the accuracy
required by the vehicle increases by a decimal point.

The data collected for this setting was analyzed above but to prove the significance of these observations ANOVA test was conducted on the data. The ANOVA test returned a P-value of zero, which clearly rejects the null hypotheses that the means of all the groups are equal. The calculated F value shows that there is a relatively large difference between the groups. The results of the conducted ANOVA test are tabulated below.

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>df</th>
<th>F</th>
<th>P-Value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>79</td>
<td>10269.14861</td>
<td>0</td>
<td>1.27533684</td>
</tr>
<tr>
<td>Within Groups</td>
<td>399920</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>399999</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The effect of population size and mutation rate on the learning time of the Minipop Algorithm was tested and the data collected was analyzed in the above sections. As mentioned earlier these simulation were run by fixing the number of evaluations per population member to 50. To study the effect on the learning time, this flaps per evaluations parameter was varied.
in the range [40, 50, 60]. In other words two more cases were considered, and the respective simulations were run and the data was analyzed. The surface plots for the Learning time, standard deviation, and effect on the fitness value can be found below in the figures 27, 28, 29, and 30.

![Figure 27: Surface Plot for Flaps per Evaluation Parameter at 40](image)

![Figure 28: Surface Plot of Standard Deviation for Flaps per Evaluation Parameter at 40](image)

The most visible effect of decreasing the flaps per evaluation is that the algorithm was not able to learn at a high mutation rate. The observations made by looking at this data can be explained as follows,
i. The Learning times range has increased three fold compared to flaps per evaluation setting of 50, the learning times for this sweep range between (500 - 15000) seconds.

ii. The rise in learning times is limited to the variation of population size when the mutation rate is at 64 bits. When the data was examined closely, there were simulations which never were able to achieve the fitness accuracy of 0.001 meter and hence the simulations were terminated due to the condition of maximum evaluations being exceeded.
iii. From the plot it can be observed that the rest of the cases have a similarity to the previous setting of 50 flaps per evaluation. It has the same linear rise in learning times as the population size is increased and the effect of mutation rate can be noticed only for 64 bits.

iv. From the standard deviation plot we can conclude that the maximum deviation is seen at high mutation rates, which shows the presence of simulation runs that have not learned in 4000000 evaluations.

v. In Figures 29 and 30 it is clearly visible that the fitness criterion was not met at high mutation rate of 64 bits.

The Anova test was conducted on the data collected for the above parameter settings. The P-value for the Anova test was found to be zero, which rejects the null hypotheses that the means of all the groups are equal. The F-value was found to be a high value which tells us that there exists a large difference between the groups under consideration. The Anova test results are tabulated below,

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>df</th>
<th>F</th>
<th>P-Value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>79</td>
<td>4338.746527</td>
<td>0</td>
<td>1.27533684</td>
</tr>
<tr>
<td>Within Groups</td>
<td>399920</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>399999</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The effect of decreasing the number of flaps for each population member on the learning time of the algorithm was analyzed in the above section. The effect of increasing the number of flaps per evaluation to 60 is analyzed below by plotting the analyzed data from the simulations run at these settings. Figures 31,32,33 and 34 show the surface plots for the learning
time, standard deviation, fitness value and its standard deviation.

The effects of increasing the flaps per evaluation parameter observed from the above surface plots are explained in the following points.

i. The learning times range between [500 - 8000] seconds, which when compared to the above to conditions is a significant change in the range.

ii. From the surface plot it can be observed that the high learning times are caused in the low mutation rate settings and low population sizes.
as this is the area where the learning times have shot up to 8000 seconds.

iii. The surface plot for the standard deviation within the groups supports the above observation that the increase in flaps per evaluation has a direct and significant effect on the learning of the algorithm in the low mutation rates. This effect is more towards the low population sizes, in other words when the search space is limited the algorithm cannot learn to meet the required fitness value.

iv. The raw data was analyzed to look for particular cases where the algorithm had timed out due to exceeding the maximum evaluations and most of the simulations that had timed out lie in the region as observed in the surface plot.

v. The rest of the area in the plot has the same observations seen in the flaps set at 50 case, where the increase in population size has a small but linear increase in learning times, effect of mutation rate is also similar in the higher settings.

vi. The fitness values plotted in Figure 33 and its standard deviations in the groups in Figure 34 confirm the observations made earlier and clearly show that in the low mutation rate and small population sizes the algorithm did not satisfy the required fitness value. Figure 34 supports Figure 33 by showing large deviations in the groups in the same area.

The Anova test was conducted on the data obtained for flaps per evaluation set at 60 and are tabulated above. The P-value returned by the
Figure 33: Surface Plot of Fitness Value for Flaps per Evaluation Parameter at 60

Figure 34: Surface Plot Standard Deviation for Fitness Value Flaps per Evaluation Parameter at 60

Table 18: ANOVA Results Flaps per Evaluation - 60

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>df</th>
<th>F</th>
<th>P-Value</th>
<th>F crit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between Groups</td>
<td>79</td>
<td>3828.363685</td>
<td>0</td>
<td>1.27533684</td>
</tr>
<tr>
<td>Within Groups</td>
<td>399920</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>399999</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Anova test is zero which rejects the null hypotheses that the means of all the groups considered are equal.
5 Conclusions

A overview of the results observed and the conclusions drawn from these observations are discussed in this chapter.

5.1 Results and Summary

The results and analysis from the previous chapter help us to better understand the performance of the Modified Minipop algorithm designed specifically for this EAH-Oscillator in the controller for the FW-MAV. The summary of the three main learning parameter sweeps can be seen in Figure 35.

The observations from the results shown in the previous chapter can be summarized as follows,

i. Population Size, Mutation Rate and Flaps per Evaluation parameters have a significant effect on the learning time of the Modified Minipop Algorithm used here in the design of the EAH enabled controller for a Flapping-Wing Micro Air Vehicle.

ii. Population Size has a direct effect on the learning time as more number of population members means more computation time required. The balance of accuracy and time, population size has a significant effect, hence will remain a parameter to tune as progress is made in the controller design.

iii. Mutation Rate is a parameter which directly affect the learning of the algorithm. A very high mutation rate will open the search space for learning and hence the cosine generated by learning might end up producing more force than the required amount hence more time
is needed to correct this error. When low mutation rates are used the search space for the driving cosine is very limited and hence the time required for the controller to learn will increase. From the above results it is observed that 75 percent is a good mutation rate to achieve a good learning time.

<table>
<thead>
<tr>
<th>Learning Parameters Setting</th>
<th>Fitness Criteria</th>
<th>Winner</th>
<th>Next Best Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-40</td>
<td>Learning time</td>
<td>Mut_Rate-16 Pop_Size-4</td>
<td>Mut_Rate-8 Pop_Size-4</td>
</tr>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-40</td>
<td>Minimum error score in altitude tracking</td>
<td>Mut_Rate-(4,8,16,32) Pop_Size-(2-32)</td>
<td>Area is flat except Mut_rate-64</td>
</tr>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-50</td>
<td>Learning time</td>
<td>Mut_Rate-16 Pop_Size-4</td>
<td>Mut_Rate-16,32 Pop_Size-8</td>
</tr>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-50</td>
<td>Minimum error score in altitude tracking</td>
<td>Mut_Rate-(4,8,16,32) Pop_Size-2</td>
<td>Area is flat except Pop_size-2</td>
</tr>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-60</td>
<td>Learning time</td>
<td>Mut_Rate-16 Pop_Size-2</td>
<td>Mut_Rate-16 Pop_Size-4</td>
</tr>
<tr>
<td>Mut_Rate-(4-64) Pop_Size-(2-32) Flaps_per_eval-60</td>
<td>Minimum error score in altitude tracking</td>
<td>Mut_Rate-(8,16,32) Pop_Size-2</td>
<td>Area is flat except Mut_rate-4</td>
</tr>
</tbody>
</table>
iv. Flaps per evaluation also have a very significant effect on the learning of the algorithm. This is the only parameter that caused the algorithm to time out and not achieve the required fitness value. From the above test results it can be observed that it is centered around 50 flaps, if decreased the algorithm times out at high mutation rates and high population sizes and if increased the algorithm times out at low population sizes and low mutation rates. This reflects the relation that more flaps tested on few population members and the search space is limited to minimum keeps the algorithm from finding the right solution and when flaps are reduced from 50 it shows that the vehicle has insufficient flaps when tested with more number of population members and maximum search space.

The data obtained for all the above test cases are significant assuming the mathematical model, ideal natural conditions for a single degree of freedom FW-MAV and this particular version of the Modified Minipop Algorithm. These concluded settings will not be significant in any way for the final EAH enabled controller, but the performance of the algorithm when subjected to variable learning parameters is significant.

5.2 Future Work

The work of this thesis has demonstrated that the FW-MAV MINIPOP algorithm can construct custom oscillators capable of correcting wing drag fault induced hover deficits. The algorithm as presented seems somewhat tolerant to different settings of learning algorithm parameters. Except for ”extreme settings”, the evolutionary search is well-behaved and produces workable solutions in minutes of flight time. This is acceptable for the
application at hand. However, it has become clear from ongoing work that has occurred in parallel to this thesis that learning oscillators for the control of multiple degrees of freedom is significantly more difficult and extends learning times onto the scale of hours of flight time. This information, combined with the parameter sweep information given in this thesis, strongly suggests that we turn our attention to issues of representation and meaningful recombination as a means to make maximally effective use of the information drawn from each expensive candidate evaluation. This thesis demonstrates that solutions are robustly obtainable. What remains is to make FW-MAV MINIPOP modifications that obtain those solutions as quickly as possible.
References


