2019

Islands of Fitness Compact Genetic Algorithm for Rapid In-Flight Control Learning in a Flapping-Wing Micro Air Vehicle: A Search Space Reduction Approach

Kayleigh E. Duncan
Wright State University

Islands of Fitness Compact Genetic Algorithm for Rapid In-Flight Control Learning in a Flapping-Wing Micro Air Vehicle: A Search Space Reduction Approach

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

by

KAYLEIGH E. DUNCAN
B.S.C.E., Wright State University, 2013

2019
Wright State University

WRIGHT STATE UNIVERSITY
GRADUATE SCHOOL

December 11, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Kayleigh E. Duncan ENTITLED Islands of Fitness Compact Genetic Algorithm for Rapid In-Flight Control Learning in a Flapping-Wing Micro Air Vehicle: A Search Space Reduction Approach BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Computer Engineering.

John C. Gallagher, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on
Final Examination

John C. Gallagher, Ph.D.

Mateen Rizki, Ph.D.

Michael L. Raymer, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Duncan, Kayleigh E. M.S.C.E., Department of Computer Science and Engineering, Wright State University, 2019. *Islands of Fitness Compact Genetic Algorithm for Rapid In-Flight Control Learning in a Flapping-Wing Micro Air Vehicle: A Search Space Reduction Approach.*

On-going effective control of insect-scale Flapping-Wing Micro Air Vehicles could be significantly advantaged by active in-flight control adaptation. Previous work demonstrated that in simulated vehicles with wing membrane damage, in-flight recovery of effective vehicle attitude and vehicle position control precision via use of an in-flight adaptive learning oscillator was possible. Most recent approaches to this problem employ an island-of-fitness compact genetic algorithm (ICGA) for oscillator learning. The work presented provides the details of a domain specific search space reduction approach implemented with existing ICGA and its effect on the in-flight learning time. Further, it will be demonstrated that the proposed search space reduction methodology is effective in producing an error correcting oscillator configuration rapidly, online, while the vehicle is in normal service.

Contents

List of Figures

List of Tables

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Numbers CNS-1239196, CNS-1239171, and CNS-1239229.

Dedicated to

All who have helped me along this journey.

Introduction

1.1 Motivation

The Harvard RoboFly triggered tremendous theoretical and practical advances towards achieving a stable and autonomous flight in Flapping-Wing Micro Air Vehicles (FW-MAV) [1] [2] [11] [13]. There are rather obvious applications for FW-MAV technology and controlling such vehicles in 3D space - search, recovery and reconnasiance.

In previous work, researchers at Wright State studied evolvable hardware and software approaches to exploring in-flight learning of control methods on FW-MAVs. These vehicles, especially, at insect sized scales, have limited payload capacity. This restricts the size and weight of the computers that can be carried. The payload limits also restrict weight of batteries that can be carried. This thesis adopts a variant on the compact genetic algorithm in an attempt to ameliorate the above mentioned computational and power limits. In so doing, it explicitly explores the use of dynamic control of the underlying genome representation to enable initial coarse scale search followed by more detailed search if such becomes necessary with the goal of shortening overall time to solution. The remainder of this chapter will expand upon motivation and explain deliverables. The remainder of this thesis will explain expiremental setup and conclusions.

1.2 Problem Overview

Most proposed approaches to the control of flapping-wing air vehicles are based on empirically verified models of the relationship between wing motion and net forces and torques applied to the vehicle by those motions. The idea is that an outer-level controller decides what forces and torques should be applied to the body, on average, over a flap of the wings, and then an inner-level controller, using a model of wing force generation, chooses how to move the wings to achieve the forces required by the outer-level controller. In a sense, the outer-level controller makes a decision about whole body forces and torques once per wing flap and then an inner-level controller moves the wings in a way that the net sums of the forces and torques on the body meet that specification. The prescription for how to move the wings is found by inverting a wing force generation model that would have been derived analytically and tuned and verified empirically.

Success of a static model based control approach depends on continuous consistency between the model and the underlying wing properties. However, the likelihood of manufacturing a vehicle that has no fabrication faults and does not suffer damage during flight is not probable at micro scales [12]. Even for real insects, normal flight can result in permanent physical wing damage that is not healed. In fact, the age of many flying insects can be reliably estimated via accumulated wing damage, but insects in nature do adapt to these damages overtime to sustain stable flight behaviors [3].

It has been demonstrated that mismatches between actual and modeled wing force generation introduces difficulties with maintaining pose and position control of the vehicle. Changes in wing force generation can be caused by acute or chronic wing damage. An example of acute damage could include a tear after hitting a wall in flight, or manufacturing faults such as the membranes of the wings not layering correctly. Chronic damage can include the wearing down of the parts over time such as changes in stiffness of the wing due to excessive flexing. One might consider multiple strategies to employ to maintain acceptable flight behavior in the face of ongoing and accumulating wing damage. One strategy

would be to adapt the wing models inside the control laws. Another strategy would be to learn new wing flap motions that allow broken wings to produce the forces predicted by the model that is already present. Calculating the control laws of the vehicle every wing flap is computationally expensive and impracticle while the vehicle is in service. Previous research at Wright State studied the idea of learning wing motion patterns that allowed damaged wings to comply with the motion-to-force models derived for undamaged wings. In short, the controllers for undamaged wings presumed cosine wing motion that was modulated by speed (frequency) and a single shape parameter that warped the cosine envelope in a manner defined later in this thesis. Previous work employed an Evolutionary Algorithm (EA) embedded in the oscillator to learn new periodic wing motion envelopes that would be modulated via the same frequency and shape parameter based warping functions. Over time, the vehicle would learn new wing motions that restored appropriate flight performance leaving the main controllers intact.

Early work utilizing EAs to evolve wing flap motions focused entirely on showing that such learning was possible and could be accomplished consistently. EA learning in this context faces at least three challenges that stem from the requirement to modify wing motion envelopes without taking the vehicle out of service:

1. Learning must be accomplished while the vehicle is in normal service. There is no resetting of the vehicle state between evaluations of candidate wing motion definitions. This opens the door to serialized deceptive evaluations. Evaluating a particularly bad candidate could place the vehicle in such a state that even a very good candidate cannot fully recover. Therefore a good candidate would receive a bad fitness score essentially inherited by the poor performance of the candidate previously evaluated.

2. The EA system must find a workable, error-correcting solution as quickly as possible. Taking several hours of flight time to correct a problem is not likely acceptable to users.

3. No candidate can be so bad that it crashes the vehicle.

With item (1) being addressed at least in an empirical sense in previous work [7] and item (3) being beyond the scope of this work, this thesis will focus on improvements to item (2). This thesis will demonstrate how to leverage search space features to reduce the effective size of the search space and thereby reduce the amount of time it takes to learn corrective wing motions when a wing is damaged.

1.3 Thesis Organization

Chapter 2 begins with an overview of evolutionary algorithms (EAs), genetic algorithms (GAs), compact GAs (CGAs) and how the ”islands” concept can be applied to a CGA. Chapter 2 will also introduce the vehicle kinematics and touch on the restrictions this poses to learning alorithms.

Chapter 3 will provide detailed descriptions of the experimental setup employed and present results of the proposed search space reduction technique across a variety of fault types that would effect net vehicle lift.

Finally, in Chapter 4, this thesis will discuss insights gained from the experiments performs and propose future work that can improve the in-flight learning performance for practical real world flight scenarios.

Background

This chapter will focus on background information. It is meant to be representative to understanding the approach of this thesis. Topics discussed will include Evoltionary Algorithms (EAs), Genetic Algorithms (GAs) and Compact Genetic Algorithms (cGAs) and introduce the Islands of Fitness approach. It will also discuss the FW-MAV model and modified adaptive controller employed for in-flight learning of corrections for membrane wing faults.

2.1 Evolutionary Computation

This section will give an overview of evolutionary alogrithms. Based on this foundation, this section will then detail genetic algorithms, and compare those to compact genetic algorithms, which is core to understanding the islands of fitness approach of compact genetic algorithms applied in this work.

2.1.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a problem solving model that take heavy influence from the biological process of evolution. In natural evolution, populations of organisms adapt to meet the demands of their environment. The fittest members reproduce and create offspring that are, ideally, a combination of suitable traits from their parents. These offspring then

compete with the previous generations. Population size is maintained by age and survival against predators and natural resources. Variety can be introduced to the population via mutation, which explores new traits at random. These new traits can be advantageous, such as a mutation of albinoism in a snowy enviroment or detrimental, such as a mutation of albinoism in a forest environment. If ideal, these traits will be perpetuated through reproduction. If detrimental, the new traits will be weeded out through survivor selection.

Determining how to map the ”real world” solution (phenotype) to a corresponding genotype is a primary factor in selecting the type of EA to utilize. This bridge is known as representation, which can refer to the encoding (the genotype space) or decoding (the phenotype space). An individual can be in reference to the original problem or the genotype representation. The elements within the individual are referred to as genes or alleles. In a genetic algorithm, the genotype could be a bit string, where each 0 or 1 within the string would be the gene at that position.

EAs fall into a variety of classifications. The principles can be generalized to the model in Figure 2.1. Two of the categories of EAs, genetic algorithms and evolution strategies, are the models that will be referenced in this work. For a more thorough overview of the different algorithms, reference Introduction to Evolutionary Computing [14].

There are generally six components to evolutionay algorithms:

1. Initial population - population can be initialized at random, at zero, or with a default value

2. Fitness function - scores the individual or population with how well it meets the needs of a potential solution

3. Selection - the set of individuals that move to the next generation or is choosen to be a parent - usually based on age and / or fitness

4. Recombination - the creation of one or more children from the traits of one or more parents

5. Mutation - one parent creates one offspring with changes to the traits that may or may not be based on values of the parent

6. Termination - a set of finishing conditions for the algorithm

An EA begins with an initial population set to values that cast a net over the space of possible solutions. During each iteration of the EA, the population can experience two stages of selection - parent and survival selection. Both processes rely on evaluations of the individual or the population by a fitness function. Population size is usually determined by how much variance is desired in the population. A larger population, depending on initialization, can perhaps better explore a search space of potential solutions with greater variety, but may evolve slowly. A smaller population may have less variance, and settle on a less than ideal solution without exploring the entirety of the search space.

The fitness function, also referred to as the evaluation function, can be thought of as a scoring for individuals, where the most correct individuals minimizes the error between the actual and desired features. It assigns a quality measure to genotypes which is usually based on the quality measure from the phenotype space. This measure can be direct, such as a solution to a factoring problem, based on subjective measures, such as aesthetics, or based on physical contraints, as the vehicle is in this work. Here, the fitness of a candidate solution can be scored by calculating the absolute value of the difference between the actual altitude of the vehicle and the desired height of the vehicle.

Parent selection is utilized in recombination or mutation, which are known as variation operators. The process of creating one or more offspring from two or more parents is known as recombination, or crossover. Recombination typically takes two individuals with desirable traits (as determined by their fitness scores) and combines them to create new individuals with desirable features. Determining which portions of each parent is subject to the type of EA being employed. Mutation creates an offspring from one parent by changing to the child's genes. Mutation is a good way to explore or exploit the search space to find solutions that may not have been tried by parents using recombination. In

genetic algorithms, mutation is the primary variation operator, while recombination is the primary search operator. Survival selection will determine which individuals, be it parents, offspring, or high-scoring candidates, participate in the next generation.

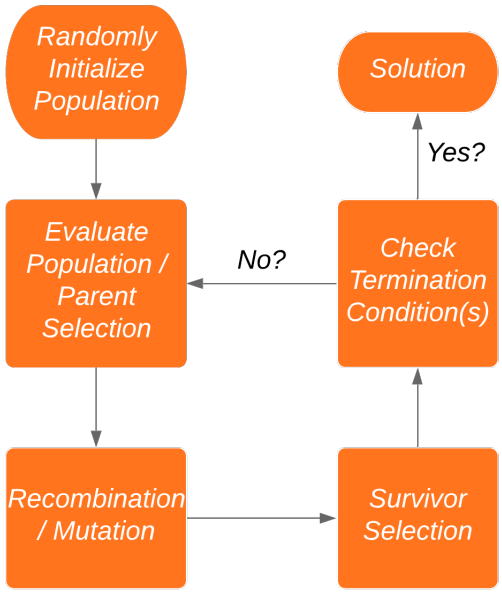


Figure 2.1: Evolutionary Algorithm General Scheme.

Survival selection (or replacement) determines which members of the population remain for the next iteration of the algorithm, usually after offspring are created. Two methods are traditionally used. One method evaluates all individuals of the population, ranks them based on score, then keeps top performing individuals - this is referred to as elitism. The other is to replace all parents with their offspring in a generational replacement strategy. Some function of both methods can be used so that the continuing population factors in rank and age, and keeps a portion of low ranking individuals in case they may create high ranking individuals over time.

The parts of the cycle described above continue until a termination condition is met. Termination conditions can include any of the following:

1. After a set number of computational (CPU) cycles.

2. After a set number of fitness evaluations.

3. Due to a lack of improvement to the candidate ranking.

4. Due to stagnation of population diversity.

Conditions one and two are typically based on time or hardware limitations. For example, the flapping-wing vehicle needs to aquire a solution that recovers flight quickly, and is therefore time restricted. It would be apropriate to utilize the second or third conditions to check for a good enough candidate being found quickly (making it futile to continue searching), or lack of population diversity (all candidates favoring one wing movement exclusively, despite attempts by mutation to combat stagnation).

2.1.2 Genetic Algorithms

Genetic Algorithms (GAs) reflect the process of natural selection, and borrows much terminology from biology. For the purposes of understanding the algorithm used in this thesis, this work will focus on information needed to compare genetic algorithms method against compact genetic algorithms in the following subsection. Table 2.1 describes the most generic GA format.

Representation	Bit-strings
Recombination	1-Point crossover
Mutation	Bit flip
Parent Selection	Fitness proportional
Survival selection	Generational

Table 2.1: Simple Genetic Algorithm [14]

To build a GA, first the representation of individuals needs to be mapped from genotype (the encoded representation) to phenotype (its real-world form). Binary representation is the simplest, where the genotype consists of a binary string. This is best suited for boolean decision variable problems. Integer representation is suitable for phenotypes,

such as ordinal coordinates, that are best assigned a unique number. For this work, a wing motion is mapped to a 12 bit binary representation. There are 8 wing motions per wing, creating a genome of length 16. A more comprehensive description of the genome is provided in Section 3.1.

Mutation is applied to a parent to create a child after a randomized change as determined by a mutation rate. For binary representations, bits on the string can be randomly flipped, usually with a low probability. This process can be applied per generation, or per offspring. For integer representations, mutation can take the form of random resetting (similar to bit flipping in a binary representation), where a new value is chosen for that position at random, or creep mutation, where a small chance (positive or negative) of mutation can be added to each gene, that should make small changes.

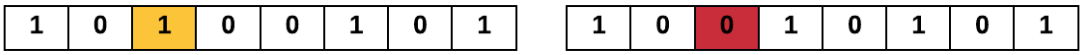


Figure 2.2: Mutation Applied to 2nd Position of Genome.

The process of selecting parents and combining them to create offspring recombination (also known as crossover), has similar options for binary and integer representations. These methods require a minimum of two parents to create two offspring. Offspring could then be evaluated by the fitness function in order to determine if only one of the two should be added to the population. For two-parent crossover, the genomes are split at one-point or n-points and recombined to create two children. Uniform crossover divides the parents based on a random number at each gene, then creates the first offspring based on the chance of each gene being carried over from each parent (the second offspring is then the inverse mapping of the first). One-point and n-point crossover can be subject to positional bias - genes that coadapted next to each other may stay next to each other. Uniform crossover, on the other hand, is subject to distributional bias, which means it may lose genes that coadapted next to each other. Understanding which bias may work in favor of the problem being

solved may allow explotation of the search space. Figure 2.3 shows a diagram of one-point crossover recombination, with the crossover point set to the 3rd allele.

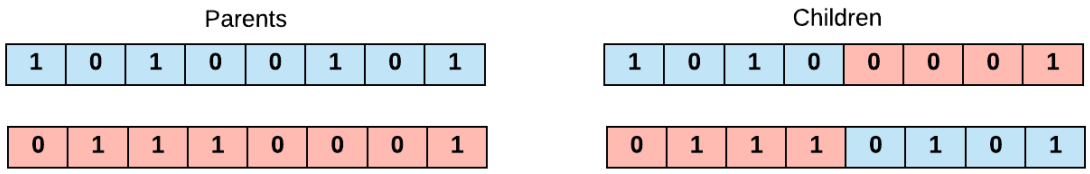


Figure 2.3: One-Point Crossover Recombination.

Once a population is randomly initialized, it is then a matter of which individuals create new populations over time. There are two primary models for evolving a population - generational and steady-state. In a generational model, all members of the population recombine and produce offspring, and then the offspring replace all parents. In a steady state model, some number of parents are selected (not the whole population), and those parents are replaced by their offspring. The competitive elements (selection and replacement) are based on individual scores given by the fitness function.

Parent selection is the competition among individuals to create offspring. Selection methods include fitness proportional, rank, and tournament based. Fitness proportional selection compares the rank of the individual to the rank of the population. This method can fall prey to premature convergence (when one outstanding individual takes over the rest of the population), and lack of selection pressure (fitness values of individuals of high rank are nearly indistinct from each other). Rank based selection is an answer to maintianing selection pressure - the population is sorted based on fitness, then assigns a probability of selection to individuals. Both rank and fitness proportional selection require knowledge of the population. Tournament selection is best for large populations or complex search spaces, and is the selection method used in the algorithm in this work. Tournament selction takes a portion, or sampling, of the individuals in a population, then compares their fitness values to identify the best members of the population sample.

Survivor selection determines the set of parents and offspring that continue to the next generation. There are four categories of schemes for survior selection:

- Age-based. No dependency on fitness, only on how many generations an individual may stay in the population. In a SGA, all members are replaced by their offspring every generation.
- Fitness-based. Contains some basis of age, and then use fitness to decide which parents move to the next generation.
- Replace worst. Worst members are removed from the population. This allows fast convergence, and therefore is best partnered with large populations or by disallowing duplicate individuals.
- Elitism. The fittest individual is kept until a higher ranking offspring is generated.

2.1.3 Compact Genetic Algorithms

This works uses the Compact Genetic Algorithm (cGAs), introduced by Golberg [4]. The cGA was chosen in previous works related to this thesis as a solution to the limited computational power and memory capabilities on the FW-MAV. There are some key differences between a standard GA and a cGA. One of these is that the population in a cGA is represented as a probability distribution of the set of solutions.

The primary reason for investigating cGAs are their space saving memory requirements. A traditional GA has a memory requirement of $l * n$ bits, the cGA requires only $l * log2(n + 1)$ bits.

The probability distribution of a population stores the odds that the best allele for a position on a genome should generate a 1. The probabilities at each position are updated over time via tournaments, where candidates are generated according to the distributions,

the candidates are evaluated, and then the probability vectors are adjusted to favor the winner.

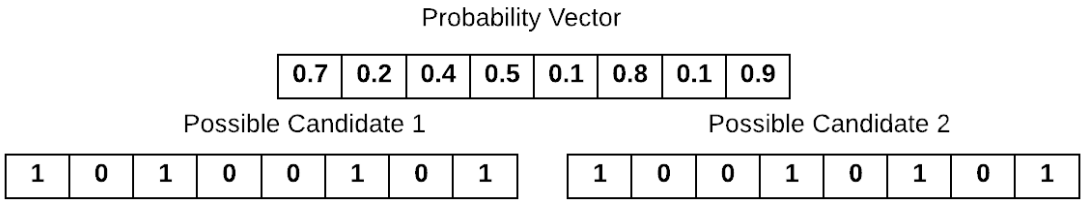


Figure 2.4: Sample Probability Vector Generating Candidates.

cGA's does not have mutation capabilities in the same sense as GAs. A GA would reset an allele to 0 or 1. For a cGA, mutations set a position in the probability distribution back to 50%. This allows the next candidates generated by the probability vector to be given a fresh chance of a 0 or 1 at that position. A hypermutation, as used in this work, is defined as reset of the whole probability distribution, or setting all probabilities to 50%, essentially forcing the population probability distribution to evolve anew.

GA's allow subpopulations in the search space - essentially clouds of good candidates that have no common properties. cGA's, due to the nature of the probabillity vector representing the population, generate population members with fixed properties over time. If a cGA has a population probability vector of $0, 0.5, 1, 0, 1, 0.5$, then the members of the population generated from the probability vector would be the same at positions 0, 2, 3, and 4 and only vary for positions 1 and 5. This removes the ability to look at completely new search spaces (short of hypermutation) that have no connective features to other good search spaces, which is why this work utilizes islands in combination with cGA.

2.1.4 Islands of Fitness

Islands of Fitness is utilized in this work as an addition to the Compact Genetic Algorithm (cGA) [4] [10]. It can be thought of as a community of CGAs, each acting as an island

in an islands-of-fitness arrangement [5]. Islands-of-Fitness approaches are often used for multi-objective problems. Islands-of-Fitness approaches could be explored for cases in which one would desire evaluation of the oscillators abilities to correct for multiple types of faults (altitude, roll, yaw, translation, etc.). Alternatively one could turn to islands-of-fitness approaches for cases where one would expect a superposition of manufacturing faults across a vehicle and individual damage faults. Islands running on different vehicles could presumably help find common solutions to solve systemic problems while individual adaptation in local islands could tune those for local needs [6]. This work uses simulated islands within a single vehicle with the goal that each island explores a unique area of the search space.

Figure 2.5: Sample of Islands with Varying Probability Vectors.

Like the cGA, the ICGA uses a probability vector to represent a population, or population island. That probability vector maintains the likelihood that each bit in the candidate solution should generate a 1. While the population is not stored directly as a collection of individuals, candidate tournaments can be held by generating candidates according to the distributions, running a tournament, and adjusting the probability vectors to in the future favor the winner. cGA, due to the probability vector that makes up the population, would explore spaces with common properties. Introducing islands allows the solution space to explore solutions with non-connective properties. This is demonstrated in 2.5, where each island can generate candidates unique to their probability vectors.

ICGA, like cGA, does not implement variation operations in the manner they would

be in an EA that stores individual members of a population. Unmodified cGA simulates uniform crossover among candidates constructing candidates from the population represented by the probability vector. We add immigrants by specifying a probability that the champion genome of one island is copied into the champion slot of another island. In our initial studies, the islands probability vector is adjusted to be 25% more similar to that of the genome that immigrated. Novelty is introduced through an aggressive hypermutation, which randomizes the island. Hypermutation, like immigration, occurs at a set probability. If hypermutation is triggered, the local champion genome, and its probability vector are reset, and the local champion score is recalculated accordingly. It should be noted that hypermutation was found to be a vital piece of the ICGA in terms of finding good solutions and not converging on unacceptable solutions, which implies that the search space has wide swaths of unacceptable solution valleys in which non-hypermutated islands could get trapped.

2.2 Vehicle and Basic Controller

This section will begin with a description of the physical vehicle, which is loosely based on the Harvard RoboFly [12]. An overview will be provided on the controller for the vehicle, as well as the basis set of wing motions used in this research.

2.2.1 Vehicle Model

An orthographic drawing of the top, front, and side views of the vehicle can be found in Figure 2.6. The two triangular wings are passively hinged to their respective support spars, which can be independently driven to angles $[-\phi, +\phi]$ (see the front view in Figure 2.6). The forward and backward stroking of the wing spars results in lifting the triangular wing plan forms to an angle α under the plane of the spars due to dynamic air pressure. These

movements results in net body forces and torques based on the lift and drag forces they produce by using the kinematic and dynamic models derived in other work, as will be reviewed later in this section [9] [13] [15].

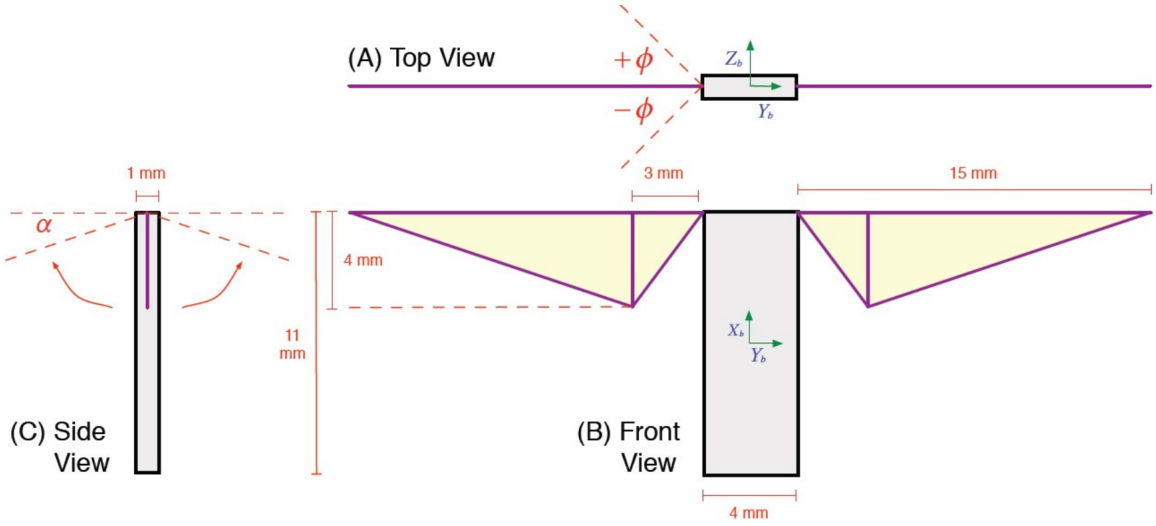


Figure 2.6: Orthographic View of Insect Scale Flapping Wing Vehicle.

The vehicle relies on two coordinate systems - body and world. The origin of the body coordinate system is at the center of mass, displayed as the rectangle in Figure 2.6. The body coordinate system is used to calculate forces and accelerations on the vehicle. The world coordinate system is conventional, where Z is the vertical (or elevation) coordinate, and X and Y are positions on planes orthogonal to Z.

The library used to simulate the vehicle in this research supports two modes: a model that constrains movement to the world Z axis (the body x axis), and a model that constrains movement to the world X and Y axis on a frictionless puck system. Both models set the wings to flap at the same frequency and both wings are at fully forward positions at the end of each wingbeat. This research is only concerned with achieving hover after wing damage, so only the first model will be considered. Since the vehicle is constrained, roll, pitch, and yaw are constant. The next section will explore split-cyle control and the altitude tracking controller for the vehicle.

2.2.2 Split Cycle Control and Wing Motion

Split-cycle control [9] provides each wing with a wing-beat frequency and a waveform shape parameter at the beginning of its wing stroke. The wing motion is defined by a split-cycle cosine wave in which the upstroke phase (motion from $+1$ to -1 radians) is a cosine whose frequency is impeded or advanced by an amount δ rad/sec, and whose down-stroke phase (motion from -1 radians back to $+1$) is governed by a cosine that is impeded or advanced so that it reaches 1 radian at the same time it would have if it had been driven by a nominal cosine with the base frequency. The wing beat cycle-averaged body forces and torques can be related to wing frequency and shape parameters through blade element analysis. Further, correction to the body pose or position can be given by a single-input and single-ouput (SISO) control law, which maps the computed desired body force or torque to wing shape parameters, which are applied to the appropriate wing on the next wing beat.

Over the course of a complete wing beat cycle, each wing produces forces and torques at the point of attachment of the wing to the body and these can be resolved to the forces and torques that will be applied to the center of the body. These forces applied to the body, determine the motion of the body and from this, the position and pose of the vehicle in the world coordinate system can be determined. Cycle-averaged methods model vehicle motion by applying body forces and torques and updating the bodys position and pose once per wing beat. Simulations of this form would be accurate in position and pose at clock ticks corresponding to the end of a wing-beat cycle. Considering the fast nominal flapping frequency of the wings, vehicle motions between wing flaps would be slight and it is likely safe to not model them directly.

The controller used in this vehicle operates on cycle-averaged values and employs split-cycle control where the wing parameters could differ between the upstroke and down-stroke phases of the wing motion [9]. Cycle-averaged control [9] is control of the vehicle based on the average of the torques and forces acting on the body over the course of an entire wing-beat cycle.

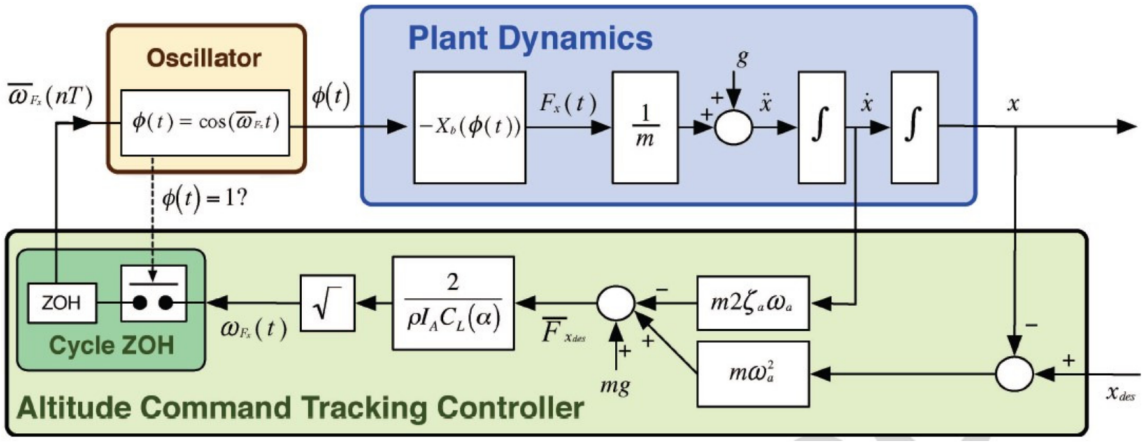


Figure 2.7: Altitude Command Tracking Controller.

2.2.3 Altitude Command Tracking Controller

The simulated vehicle in this work is restricted to one degree of freedom for a constrained hover. In constrained hover, the vehicle can only translate along the world Z axis (altitude). Given that both wings beat at the same frequency on a cycle-averaged basis, all forces and torques with the exception of the upward force are canceled in cases where the wing motions are otherwise symmetric. The motions made by the wings are metaphorically similar to the arm motions of a swimmer treading water to maintain position the swimmer must make symmetrically timed and equivalent motions. A basic altitude error feedback and cycle-averaged controller is combined conceptually in Figure 2.7.

This controller, referred to as the Altitude Command Tracking Controller (ACTC) consists of an oscillator and a plant dynamics module that feed into the controllers output. The plant dynamics module uses the drive angles, ϕ, to calculate the force produced by the movement, which is then twice integrated once to get the current velocity of the vehicle ($\dot{x}$) and again to get the vehicles position, or altitude. This information and the desired position of the vehicle (x_{des}) is fed to the controller, which computes a wing flap frequency that is passed to the cosine oscillators on each wing. Further details of ACTC can be found in [7] [8].

The issue with the ACTC as decribed is that the oscillator cannot learn new wing motions. A new oscillator model was created to ”adapt” to wing damage scenarios and generate forces needed to regain elevation control. The following section will provide more detail.

2.2.4 Adaptive Oscillator Controller

In this and previous work the simple cosine oscillator used by the Altitude Command Tracking Controller (ACTC) has been replaced with an Adaptive Learning Oscillator - (ALO) [7], a schematic for which can be found in Figure 2.8. The original oscillator worked for models of the vehicle that had no flaws. The model of the vehicle used in this work needs learning applied in order to generate wing motions after damage to the wing(s).

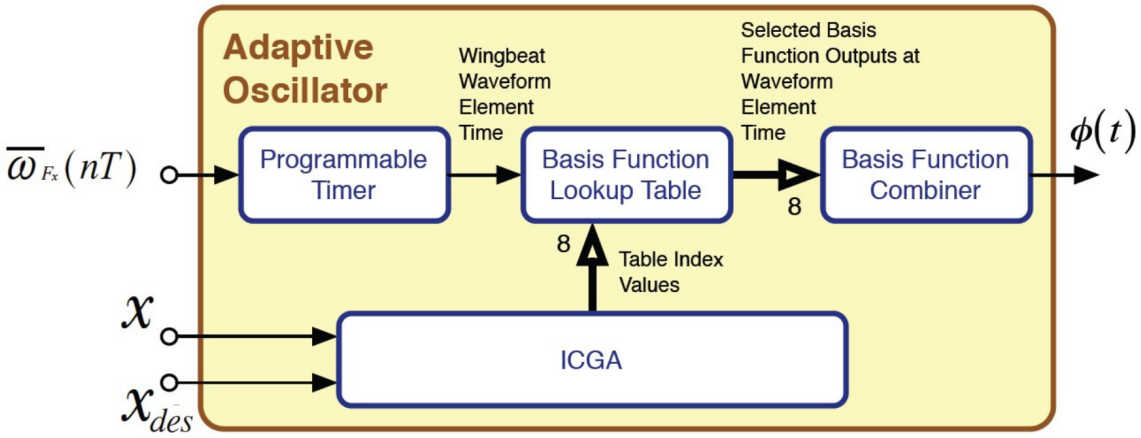


Figure 2.8: Adaptive Learning Oscillator Schematic.

The ALO learns new wing motion schedules, or waveforms, that better approximate and restore the desired relationship between wing flap frequency and upward force. The adaptive oscillator maintains an internal library of pre-computed wing motion basis functions that are combined to produce specific wing motions. Machine learning is used in flight to combine the basis functions to enable near-optimal control for specific vehicles

damaged in service or suffering from manufacturing flaws in the wings. The oscillator learns which basis functions should be combined based on real time samples of the error between the desired (x_{des}) and actual (x) altitude. Both the basis function and the learning algorithm used in this thesis, cGA, were designed to minimize the amount of digital components required for implementation and to limit the number of computational cycles required to achieve learning. A reference implementation of the required hardware can be found in previous work [8] [9].

2.2.5 Wing Motion Basis Functions

This thesis uses a modified version of cGA as the learning engine. The learning engine, once every one hundred and fifty wing flaps, receives a measure of the desired and actual vehicle altitude and computes the error towards learning to find apt indices to mix basis functions to produce unique left and right wing motion functions, to minimize the error in the altitude. Internally, each of the two wing motion functions is stored as an eight element coded vector, where each of the eight positions correlates to the table index of one of the pre-computed wing position tables. When the vehicle is in operation, a digital timer advances through one of 256 time steps in each wing position table and adjusts each wings position to the average of the eight basis functions associated with that wing.

The four wing motion core basis functions are given below:

$$A(x) = cos(x)$$

$$B(x) = \frac{cos(x) + cos(3x)}{2}$$

$$C(x) = \frac{2cos(x) + cos(3x)}{3}$$

$$D(x) = \frac{4cos(x) + cos(3x)}{5}$$

These four basis functions satisfy the following constraints:

1. Wings are fully forward ($\phi = 1$) at the beginning and end of each wing beat

2. Each is a cosine function sometimes with faster frequency cosines superimposed over them

3. They encapsulate non power-of-two divides and multiplies into pre-computed basis table functions

The lookup table inside the oscillator stores 16 classes of pre-computed functions that combine the upstrokes and downstrokes of the above equations. For hardware implementations, only shifters and adders would be required for the computational portions of the circuit. Each of the 16 combinations of waveforms comes in 256 time shifted varieties where the lowest valley is time shifted along the x-axis [9]. Note that this implies there are $16 * 256 = 4096$ distinct basis functions, which correlates to the range of each of the eight indices for each wing. The 16 basis functions can be seen in Figure 2.9.

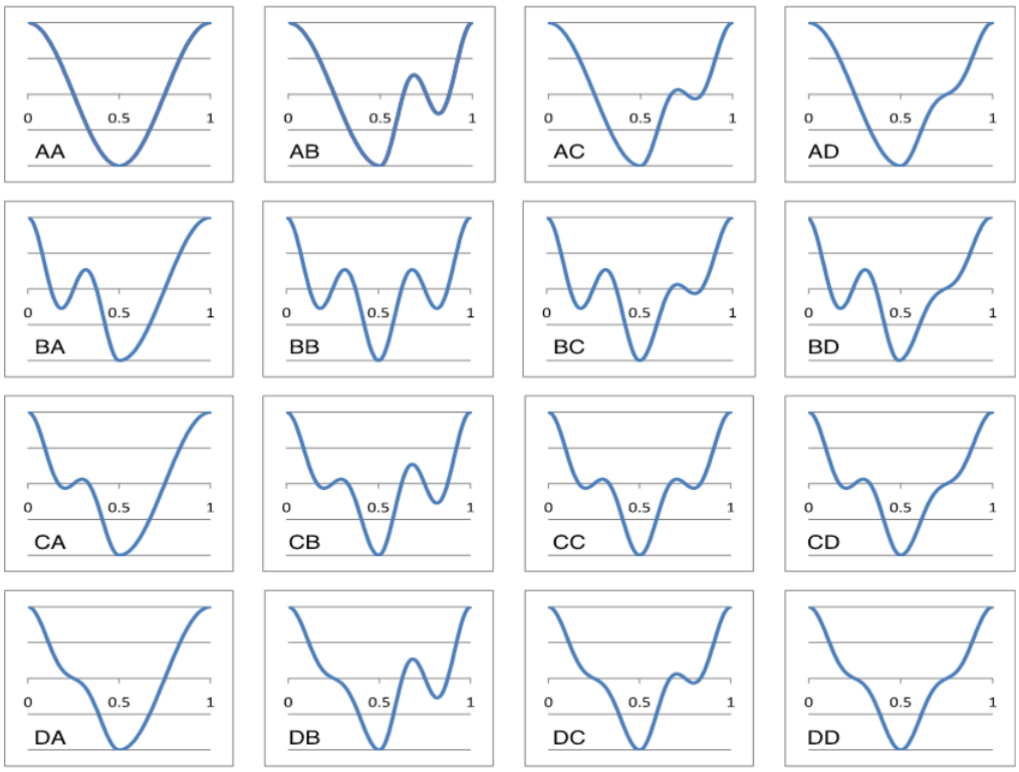


Figure 2.9: The 16 Composite Up/Down Stroke Basis Functions.

Experimental Setup and Results

This section will discuss the expirements designed to shed light on questions raised in the introduction. It will describe a method of state space restriction unique to this genome and expirements that will help uncover the potential value of those restrictions.

3.1 Describing the Genome

The genome representation for this problem is somewhat unusual and requires additional explanation. The genome consists of 16 basis function indices, split into 8 indices per wing. This number was originally chosen based on hardware consideration and ease of implementation on comodity FPGAs. Each index is a 12 bit number, which represent the binary value of one of 4096 possible basis functions defined by the basis table set as defined in the previous chapter. The average of the eight functions specified by the indices specifies a wing motion function used by the wing. Because the allele values are indices into a table of basis functions that are averaged to produce a composite function, the location of the alleles on the genome is irrelevant as an index in any one allele location has exactly the same effect on the composed average wing motion function as it would in any other location.

This effect imbues the genome for this problem with several interesting properties. First, there is high chance of redundancy in the encoding. In this case, redunancy means

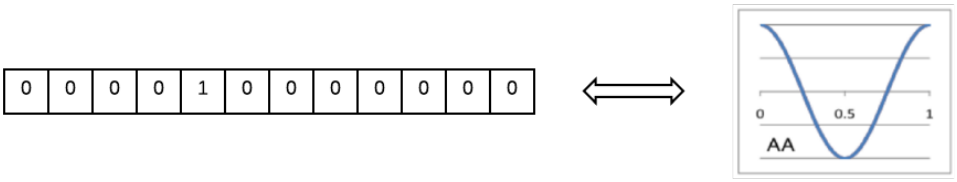


Figure 3.1: A Genome Bit Position Representing Cosine Basis Function.

there is more than one genome that will decode to the same pheotype. The genome is actually a multi-set of fixed cardinality, and there are multiple encodings for most wing motion functions. This multi-set genome contains the following properties: order of allele values does not matter, and weight is based on redundancy (or multiplicity) in the set. Any permutation of the genome averages to the same result if reordered, same as multiset $\{1, 2, 3\}$ is the same as $\{3, 1, 2\}$. Figure 3.2 shows an evolved genome where each wing has a multiset of $\{AA : 3, AD : 2, CD : 1, DC : 1, DD : 1\}$, thus each wing would average to the same wing motion funtion. Second, because of the encoding, forcing symmetries internal to the genome have the effect of evolving on a smaller genome. For example, restricting the genome so that positions $0 - 7$ are identical to positions $8 - 15$ would functionally be equivalent to just evolving on a four element genome that had only positions $0 - 7$. The implications of the second point is the basis for reducing the search space.

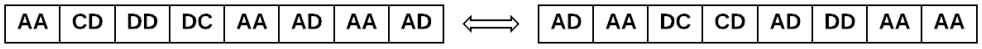


Figure 3.2: Genome with Basis Functions at Each Bit Position.

3.2 ICGA Evaluation Function

The algorithm stores a global champion, which is initialized to a set of indices that encode a pure cosine function. Over time, that global champion is updated to hold the genome

that, to present time, produced the least altitude error - the absolute value of the difference between the actual and desired height. This is an aggressive form of elitism. The algorithm is encoded such that every island maintains its own local champion and probability vector. A tournament is held between one randomly generated candidate and the islands champion, then again between the randomly generated candidate and the global champion. If the random candidate wins, it replaces the local champion, updates the local champion score to reflect that of the random candidate, and updates the probability vector to favor the random candidate. If the random candidate is not better than the local champion then probability vector of the island is updated to better reflect the local champion. If the randomly generated candidate is better than the global champion, then the global champion genome is replaced with the random candidate and its score. Pseudo-code for the ICGA implementation is shown in Psuedo-code 1.

Termination of the ICGA search will occur should any of the following be met:

1. The champion genome allows the vehicle to fly within 0.1 mm of the desired target height
2. The population is completely converged (all probability vector bit positions are at 0.0% or 1.0%)
3. The maximum number of evaluations has been exceeded (in this case, 80,000)

The first termination condition reflects a successful solution the target goal has been reached. The second condition reflects complete convergence on an unacceptable solution and would eliminate the possibility of finding an acceptable solution. For this work, the very aggressive hypermutation essentially excludes this outcome. The third condition acts as a timeout should no good solution be found. At 80,000 evaluations, this correlates to about 28 hours of vehicle flight time. Since this is likely an unreasonable amount of time to expect the vehicle to fix itself, it is a reasonable cut off for further learning. In addition, in this work is counting any solution that takes more than eight hours of flight time to be

unacceptable. In practice, however, mission requirements might cause us to bring that limit significantly lower.

Algorithm 1 Psuedocode for ICGA Champion Update Function

```
function ICGA_UPDATE
    while termination_condition ≠ 1 do
        for all islands do
            EVALUATE(global_champion)
            global_champion_score ← EVALUATE(global_champion)
            local_champion_score ← EVALUATE(local_champion)
            EVALUATE(global_champion)
            GENERATE_RANDOM_CANDIDATE()
            random_candidate_score ← EVALUATE(random_candidate)
            if random_candidate_score ≤ local_champion_score then
                UPDATE_PROBABILITY_VECTOR(random_candidate)
                local_champion ← random_candidate
                local_champion_score ← random_candidate_score
            else
                UPDATE_PROBABILITY_VECTOR(local_champion)
            end if
            if random_candidate_score ≤ global_champion_score then
                global_champion ← random_candidate
                global_champion_score ← random_candidate_score
            end if
            if immigration_flag = 1 then
                UPDATE_PROBABILITY_VECTOR(global_champion)
            end if
            if hypermutation_flag = 1 then
                RANDOMIZE_LOCAL_CHAMPION()
                RESET_PROBABILITY_VECTOR()          ▷ Bit positions to 0.5
            end if
        end for
    end while
end function
```

3.3 Search Space Reduction

As mentioned earlier, each genome is encoded to consist of 16 basis functions, which is divided into 8 basis functions for the left and right wing. The eight basis functions per wing

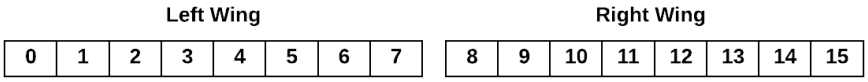
are averaged to create a composite function that correlates to one of 4096^{16} combinations of cosine functions. Because of the averaging operator on 8 basis functions (per wing), there exists a possibility that any one solution can have multiple correlating average sets, meaning that multiple sets can have a variation of basis functions, but they can average to the same solution. This encoding, where solutions are multi-sets and not directly positional, makes it reasonable to use multi-fold symmetry constraint to reduce the number of multi-sets that average to the same solution when attempting to evolve wing patterns. Reducing the effect of cardinality in the multi-sets reduces the search space, thus aiding it to learn a good solution faster and improve yield.

In this vein, three different types of multi-fold symmetric constraints have been implemented and experimented in the current work. These three types are:

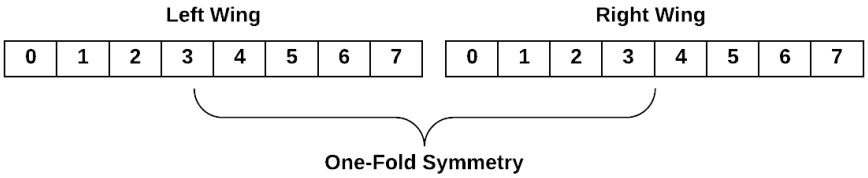
- Zero-fold symmetric constraint (ZSC).
- One-fold symmetric constraint (OSC).
- Two-fold symmetric constraint (TSC).
- Four-fold symmetric constraint (FSC).

As shown in Figure 3.3(b), a one-fold symmetric constraint assumes symmetry across the wings, thus duplicating the eight indices for one wing over the other. This first symmetry is actually not only a search space restriction in the sense already discussed - it is also a restriction on phenotypes that requires two potentially physically different wings to produce a sum of forces that is both correct and generated by symmetrical wing motions.

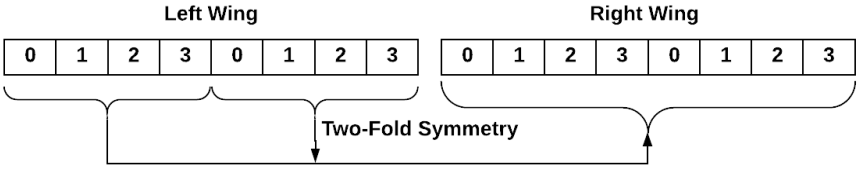
The two-fold symmetric constraint inherits the one-fold symmetric constraint and assumes one-fold symmetry within each wings individual genome segment at the 4th index, thus duplicating 4 times across the wings as shown Figure 3.3(c). As shown in Figure 3.3(d), a four-fold symmetric constraint inherits the two-fold symmetric constraint and as-



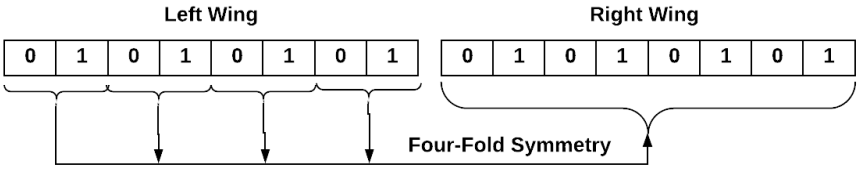
(a) Zero-Fold



(b) One-Fold



(c) Two-Fold



(d) Four-Fold

Figure 3.3: Symmetric Constraints to Reduce Search Space

sumes one-fold symmetry at 2nd index of the wing, thus duplicating 8 times across the wings.

Without the symmetric contraints, each of the 16 positions can pick one of 4096 basis functions, or 4096^{16} possibilities. On forcing symmetric motion across the wings (OSC), the possibilities are reduced to 4096^{8}, which is the same as only choosing 8 basis functions to average. TSC chooses 4 basis functions to average, with a total of 4096^{4} possibilities. The most restrictive symmetry, FSC, chooses only two basis functions to average, or 4096^{2} possibilities. In effect, the dimensionality of the search space is reduced with each type of constraint.

3.4 Simulated Vehicle Operating Conditions

To evaluate the effectiveness of the learning module in conjunction with the search space reduction techniques mentioned above, four vehicle operating conditions are simulated:

1. Wings with no damage: Experiments in this set employ unbroken wings. The purpose of these experiments is to see how long it takes for the learning system to acquire working wing motion functions when they are started randomly.

2. 75% lift one wing: Experiments in this set have ONE broken wing with up to a 25% lift force deficit uniformly selected from the range [0.0% .. 25.0%]. This means each trial has one unbroken wing and one wing with up to 25% loss is lift force capability.

3. 87.5% lift both wings: Experiments in this set have TWO broken wings with up to a 12.5% lift force deficit in EACH of the two wings. This means each trial has two broken wings with individual lift faults of up to 12.5% each. The can have up to the same overall deficit as the 75% one wing lift case (25%), but distributed over two wings.

4. 80% lift both wings: Experiments in this set have TWO broken wings with up to 20% lift fault per wings. This vehicle could have up to a 40% lift fault over two wings. At 60% lift capability, a vehicle would require very fast wing flapping rate and/or aggressive wing gaits just to remain hovering. This represents the most difficult type of fault considered in this thesis.

This thesis will introduce experiments to demonstrate the effectiveness of search space reductions in each of the four above categories.

3.5 Experimental Results

Over 250 experiments (actual N listed in Tables 3.1 - 3.4) were run for each type of vehicle operating condition and with the three proposed multi-fold symmetric constraints. As mentioned earlier, an acceptable solution was defined as one that maintains hover of the vehicle within 0.1 mm of the target height and is found in fewer than 8 hours of simulated flight time. While the algorithm could have terminated early due to convergence, it never did, likely due to aggressive hypermutation. All experiments terminated either after reaching the evaluation limit or once they found an acceptable solution. Further, all the experiments were run with a 12.5% immigration probability, a 12.5% hypermutation probability, an evaluation limit of 80,000, and 16 islands in ICGA. This work will address possible exploitation of the symmetric (multi-set) features of the encoding and hold all these other values to settings that have been empirically observed to produce good yield, if not stellar learning times. The set of experiments with vehicle operating with no wing damages and without any multi-fold symmetric constraint is chosen as the baseline experiments to compare the effect of multi-fold symmetric constraint based search space reduction. Further the two performance metrics employed to measure the improvement of the proposed search space reduction techniques are:

1. Yield: This is defined as the number of the total conducted evaluation runs that pro-

vided an acceptable solution (with criteria mentioned in the evaluation function).

2. Learning Time: This is defined as the time taken by each evaluation run to arrive at the acceptable solution (with criteria mentioned in the evaluation function).

Thus, the above two performance metrics are calculated for all the experiments conducted for each type of vehicle operating conditions with three proposed multi-fold symmetric constraints and tabulated in Tables 3.1 - 3.4. The columns in the tables represent the constraint performed, the 25th percentile of learning times, the 50th percentile of learning times, the 75th percentile of learning times, the minimum learning time, the maximum learning time, and the number of successful experiments run, respectively. All table entries except for N are in minutes of flight time. The N entries are the number of trials in each experiment. Yield of the expirements, or how long each experiment took to complete for each scenario, can be observed in Histograms 4.1 - 4.16. The list below decribes the acronyms in the tables.

- N - Number of trials in the set
- ZSC - Baseline scenario (no symmetry)
- OSC - One-fold symmetric constraint
- TSC - Two-fold symmetric constraint
- FSC - Four-fold symmetric constraint
- Q1: First quartile (25th percentile) of flight time to learn solution
- Q2: Second Quartile (50th percentile / median) of flight time to learn solution
- Q3: Third Quartile (75th percentile) of flight time to learn solution
- Min: minimum flight time to learn solution

- Max: maximum flight time to learn solution
- Mean: mean flight time to learn solution.

Table 3.1 lists metrics from the experiments where the vehicle is operating without any damages to wings. Table 3.2 lists metrics from the experiments where the vehicle is operating with randomized damage to one wing, limiting it to generating 75% of maximum expected lift. Table 3.3 lists metrics from the experiments where the vehicle is operating with randomized damage to one wing, limiting it to generating 75% of maximum expected net lift. Table 3.4 lists metrics from the experiments where the vehicle is operating with randomized asymmetric damage to both wings, limiting them to generating 60% of maximum expected net lift.

	Q1	Q2	Q3	Min	Max	Mean	N
ZSC	49.16	103.32	207.73	3.29	538.67	141.99	956
OSC	44.22	96.89	196.45	1.66	523.87	135.97	961
TSC	31.41	68.05	144.08	1.65	510.34	102.03	989
FSC	33.84	77.73	163.92	1.68	536.70	114.19	988

Table 3.1: Experimental Results of Wings with No Damage

	Q1	Q2	Q3	Min	Max	Mean	N
ZSC	75.25	168.96	312.92	3.13	538.11	202.72	709
OSC	58.83	131.44	257.99	1.63	538.10	175.41	836
TSC	38.00	95.18	184.12	1.62	521.64	129.12	957
FSC	32.62	80.76	169.52	1.65	539.17	119.97	983

Table 3.2: Experimental results of one wing up to 25% damaged: can produce 75% net lift force

	Q1	Q2	Q3	Min	Max	Mean	N
ZSC	79.34	169.35	313.97	1.63	538.75	206.04	713
OSC	64.98	151.53	290.20	1.62	537.93	186.77	845
TSC	34.44	81.98	168.74	1.63	694.15	120.55	978
FSC	30.27	76.90	158.45	1.65	528.09	110.75	989

Table 3.3: Experimental results of each wing up to 12.5% damaged: can produce 75% net lift force

	Q1	Q2	Q3	Min	Max	Mean	N
ZCS	109.46	205.39	342.00	3.27	538.99	230.92	535
OSC	69.10	161.79	285.78	1.59	536.82	191.25	719
TSC	45.76	109.02	213.48	1.65	539.25	144.54	943
FSC	38.29	83.78	167.35	1.61	532.73	120.69	975

Table 3.4: Experimental results of each wings up to 20% damaged: can produce 60% net lift force

Conclusion

In this thesis, the challenges of in-flight learning for the FW-MAV oscillator control problem have been outlined. Further, a multi-fold symmetric constraint based search space reduction technique has been proposed and implemented for the adaptive ICGA based learning module in a FW-MAV flight controller.

In the expiremental results Tables 3.1 - 3.4, Kruskal-Wallis ANOVA tests establish that the rows do not represent distributions with the same mean. Pairwise Mann-Whitney tests made between subsequent rows in each table allow rejection of the null hypothesis (the rows have the same means) with the levels of significance listed in Tables 4.1 - 4.4. Green cells represent that there is a significant difference in the means, and the null hypothesis can be rejected. Red cells represent that there is not a significant difference in the means, and the null hypothesis can not be rejected. Yellow cells represent that there is a weakly significant difference in the means, and the null hypothesis can be rejected.

There is a signicant difference in the means between not applying symmetric contraints (ZSC) and applying four-fold symmetric contraint(FSC). This implies that having the vehicle pick two basis functions provides working solutions for pendulum stable vertical flight. Due to this observation, it can be assumed that the solution space has large quantities of duplicate solutions. Anecdotally, it was observed that the hypermutation settings used by the algorithm are necessary to explore the search space efficiently.

	ZSC	OSC	TSC	FSC
ZSC	1.00	0.36751	0.00	5.6e-8
OSC	0.36751	1.00	4.3e-12	6.5e-6
TSC	0.00	4.3e-12	1.00	0.0094
FSC	5.6e-8	6.5e-6	0.0094	1.00

Table 4.1: Mann-Whitney Mean Comparison Tests for No Wing Damage

	ZSC	OSC	TSC	FSC
ZSC	1.00	0.00018	0.00	0.00
OSC	0.00018	1.00	5.0e-12	0.00
TSC	0.00	5.0e-12	1.00	0.03295
FSC	0.00	0.00	0.03295	1.00

Table 4.2: Mann-Whitney Mean Comparison Tests for One Wing Damaged up to 25%

	ZSC	OSC	TSC	FSC
ZSC	1.00	0.007	0.00	0.00
OSC	0.007	1.00	0.00	0.00
TSC	0.00	0.00	1.00	0.08656
FSC	0.00	0.00	0.08656	1.00

Table 4.3: Mann-Whitney Mean Comparison Tests for Boths Wings Damaged up to 12.5%

	ZSC	OSC	TSC	FSC
ZSC	1.00	7.03e-7	0.00	0.00
OSC	7.03e-7	1.00	2.44e-12	0.00
TSC	0.00	2.44e-12	1.00	0.00004
FSC	0.00	0.00	0.00004	1.00

Table 4.4: Mann-Whitney Mean Comparison Tests for Boths Wings Damaged up to 20%

Experimental evidence seems to indicate with statistical significance desirable effects on both solution yield and learning time. Completed trials versus time (in minutes) can be observed in Figures 4.1 - 4.16. These effects are presumably due to reductions in the volume of the search space and there existing workable solutions within the four-fold symmetric constraint (FSC) criteria. It was anecdotally observed that running search experiments with only one basis function (choosing from exactly one of the 4096 core basis functions) leads to terrible, near zero, yields. Although the nature of how basis functions combine to produce required frequency to lift force is not yet fully studied, at least two basis functions are required for consistent success.

It is relatively certain that combinations of two basis functions are sufficient to balance lift faults. It is not clear that merely two would be able to correct other motion faults like roll control deficits. Previous work learned oscillation functions that simultaneously correct for altitude and roll faults. Naturally these restriction experiments need to be updated for those and expanded situations. An interesting possibility, however, might be to impose a symmetry condition to more quickly regain appropriate control of altitude (presumably more important than precise control of roll) and then release the symmetry to allow additional degrees of freedom to better correct additional vehicle fault conditions.

This idea of space restriction via symmetry is independent of the specific EA used. Improvements to ICGA via other means or constructing an EA more suited to this specific problem, should find that this form of symmetry restriction remains compatible and continues to provide the benefit of producing an error correcting oscillator configuration rapidly, online, while the vehicle is in normal service.

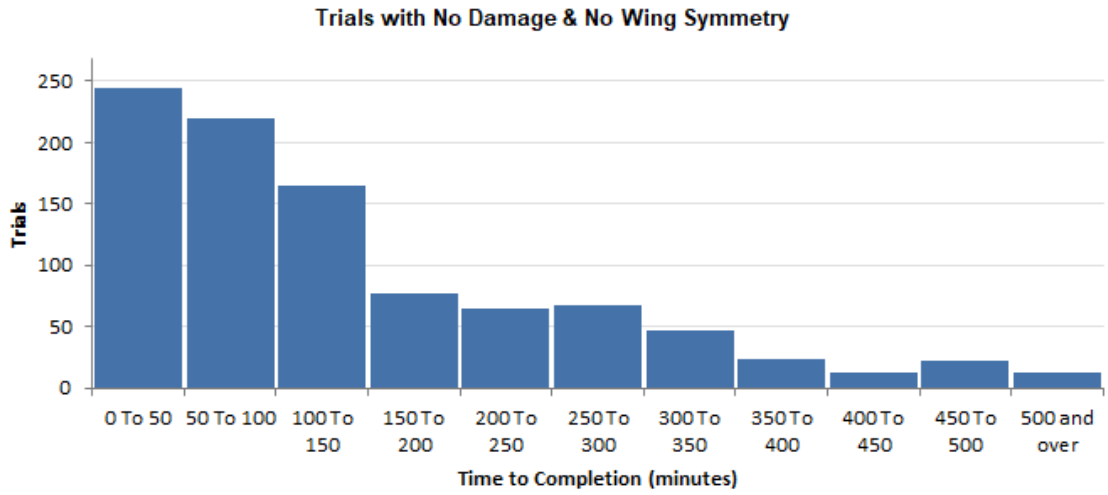


Figure 4.1: Number of Trials vs. Time to Completion without Wing Symmetry - No Damage

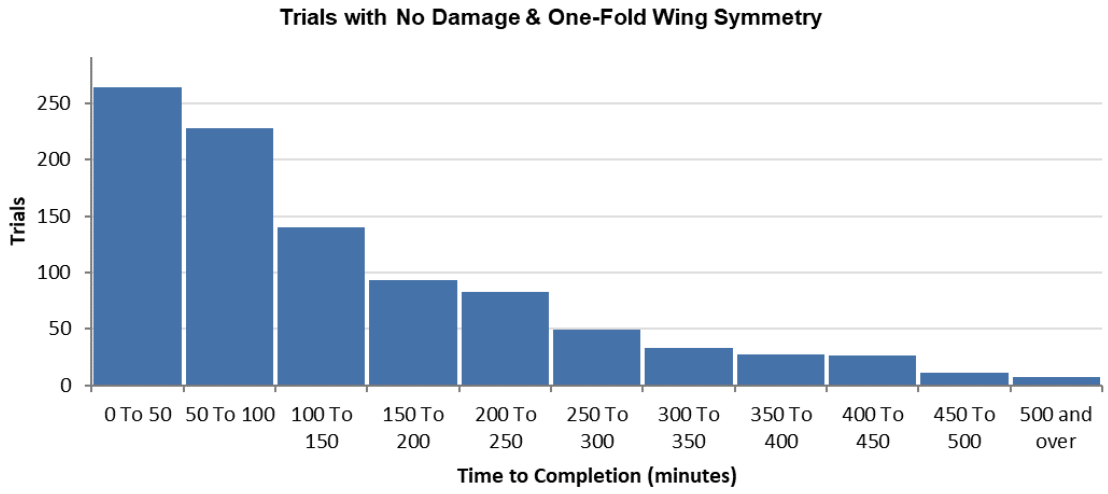


Figure 4.2: Number of Trials vs. Time to Completion with One-Fold Symmetry - No Damage

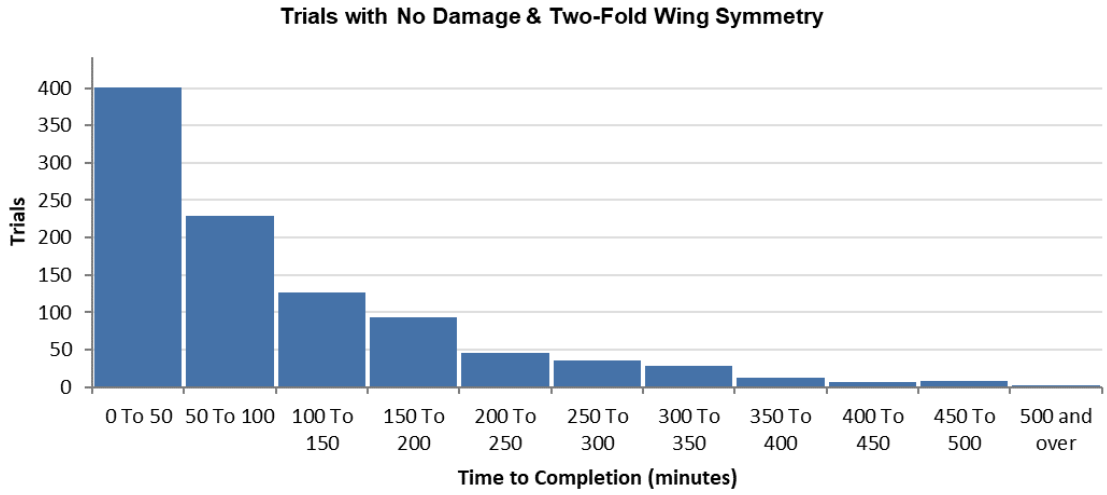


Figure 4.3: Number of Trials vs. Time to Completion with Two-Fold Wing Symmetry - No Damage

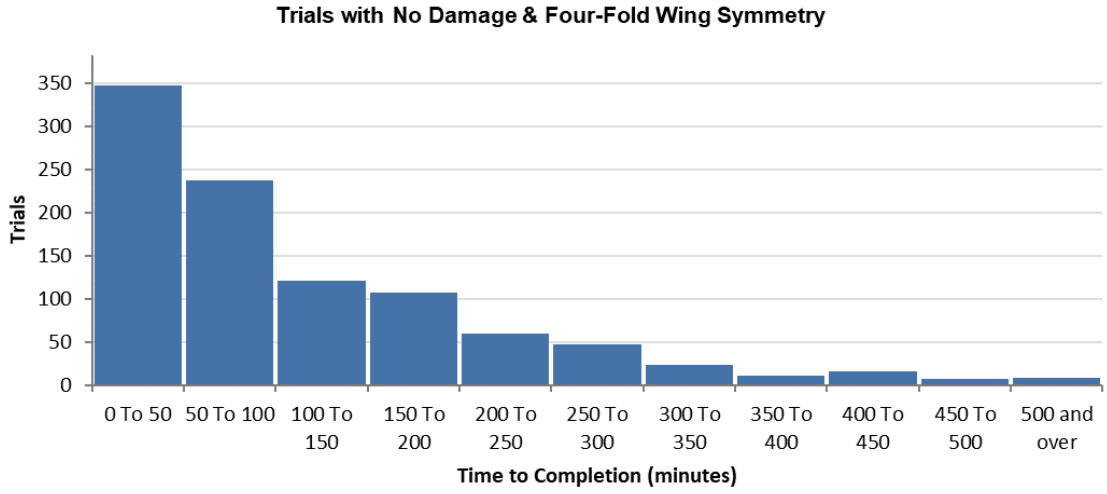


Figure 4.4: Number of Trials vs. Time to Completion with Four-Fold Wing Symmetry - No Damage

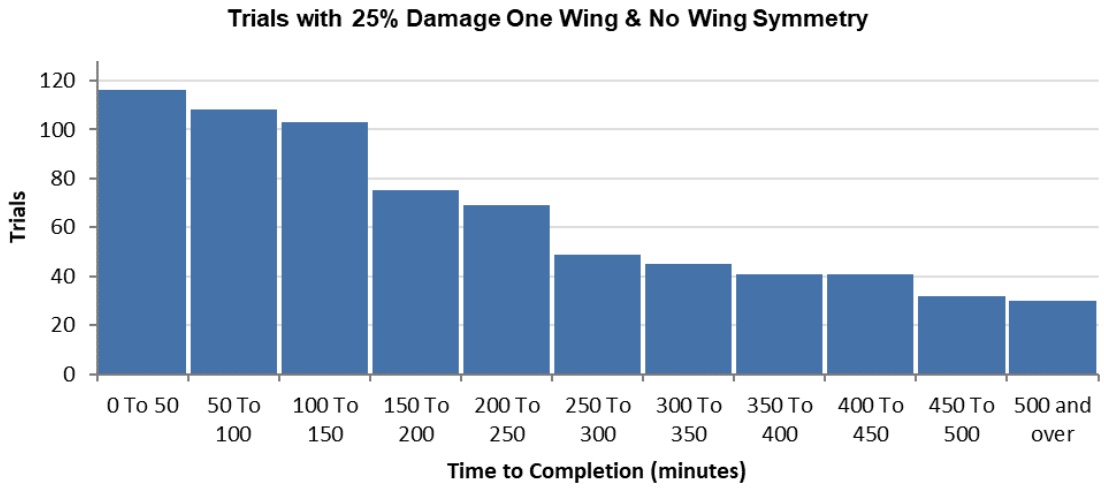


Figure 4.5: Number of Trials vs. Time to Completion without Wing Symmetry - Up to 25% Damage to One Wing

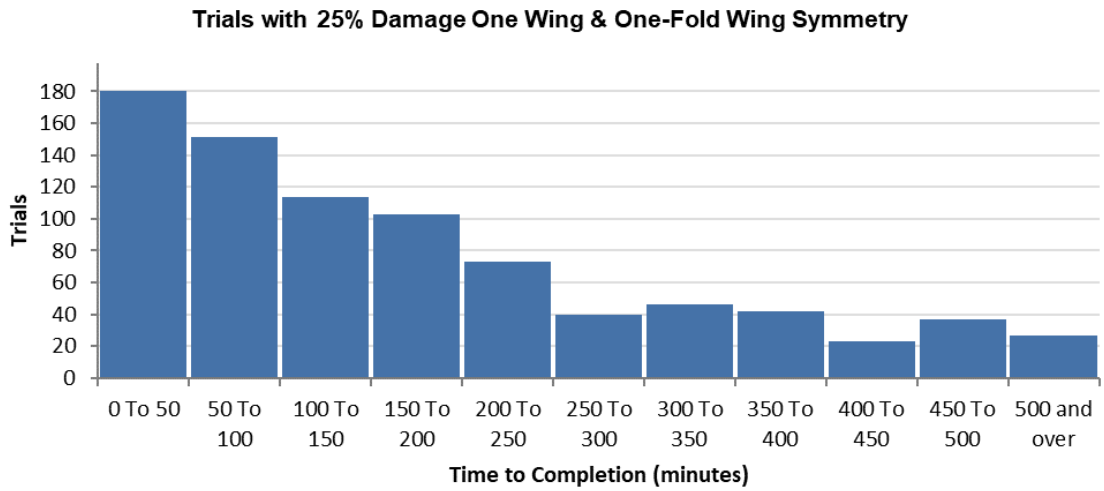


Figure 4.6: Number of Trials vs. Time to Completion with One-Fold Wing Symmetry - Up to 25% Damage to One Wing

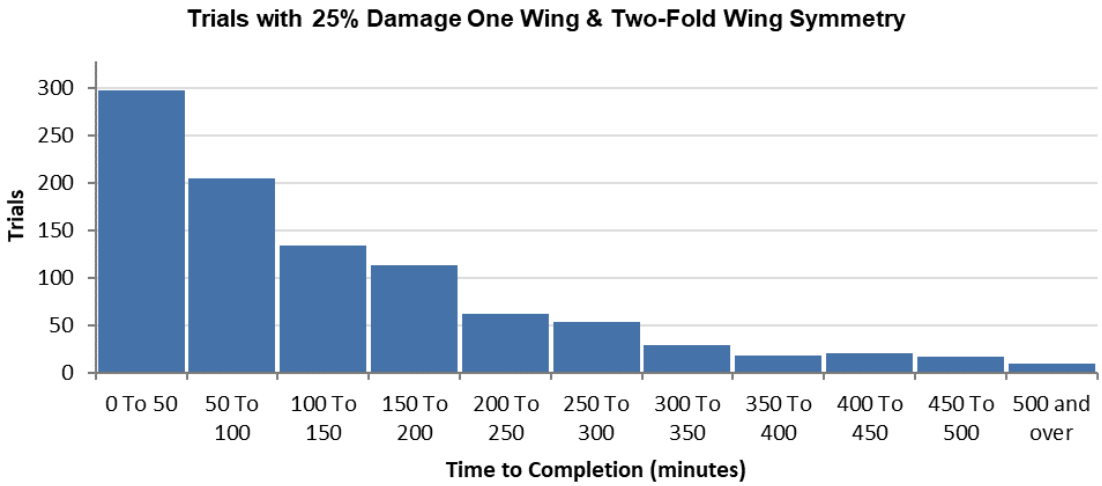


Figure 4.7: Number of Trials vs. Time to Completion with Two-Fold Wing Symmetry - Up to 25% Damage to One Wing

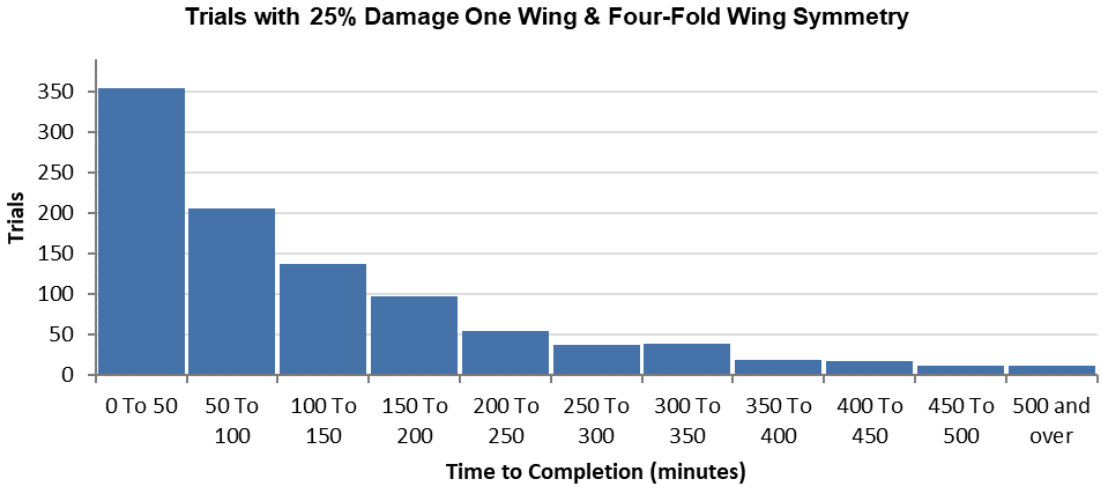


Figure 4.8: Number of Trials vs. Time to Completion with Four-Fold Wing Symmetry - Up to 25% Damage to One Wing

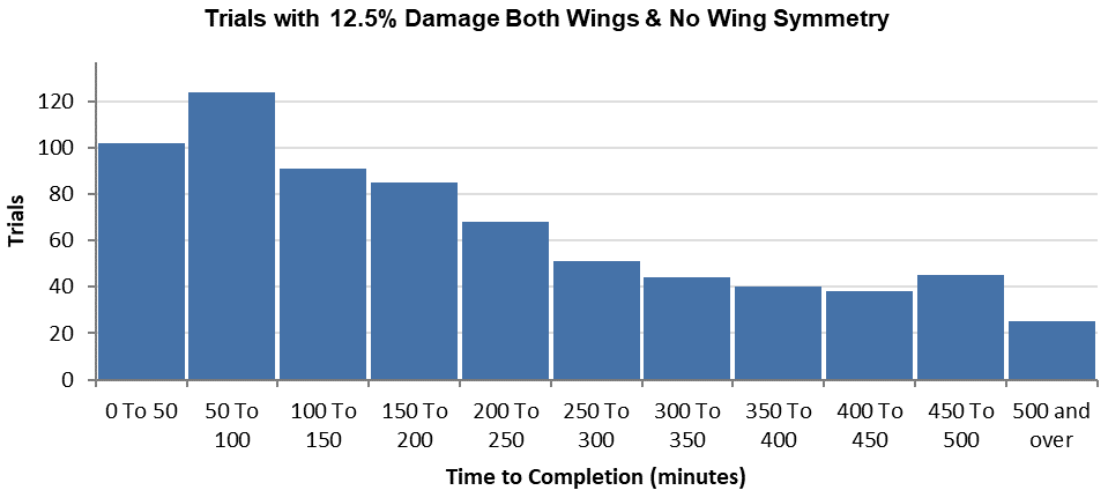


Figure 4.9: Number of Trials vs. Time to Completion without Wing Symmetry - Up to 12.5% Damage to Both Wings

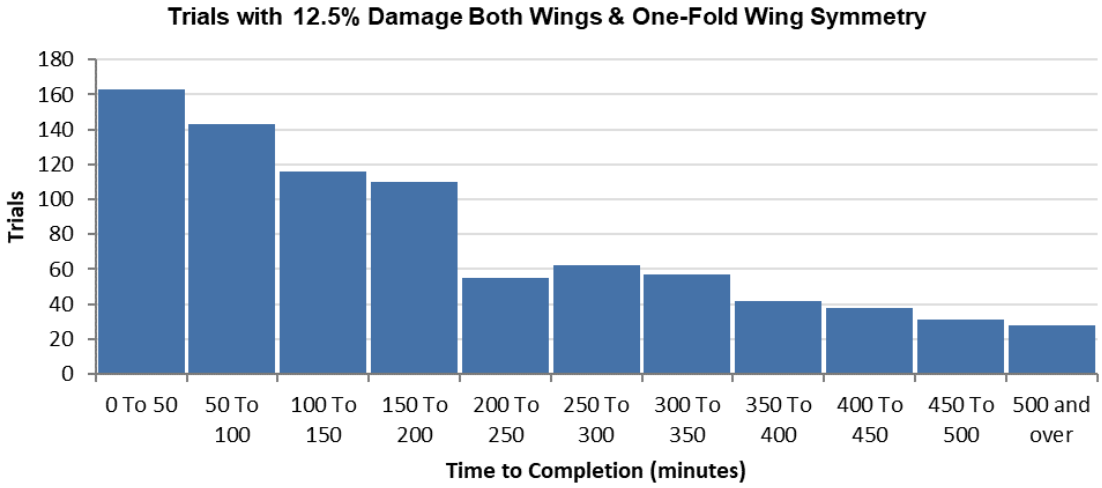


Figure 4.10: Number of Trials vs. Time to Completion with One-Fold Wing Symmetry - Up to 12.5% Damage to Both Wings

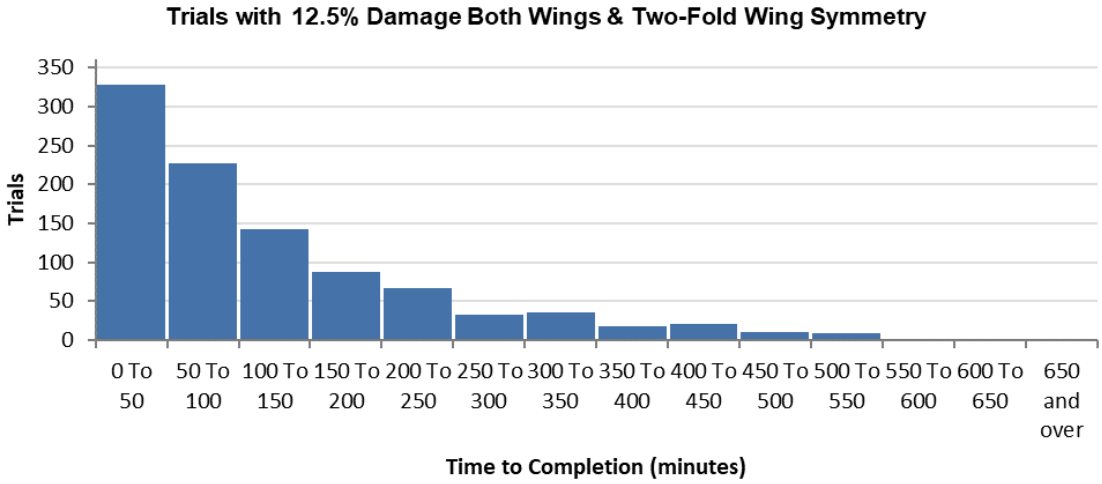


Figure 4.11: Number of Trials vs. Time to Completion with Two-Fold Wing Symmetry - Up to 12.5% Damage to Both Wings

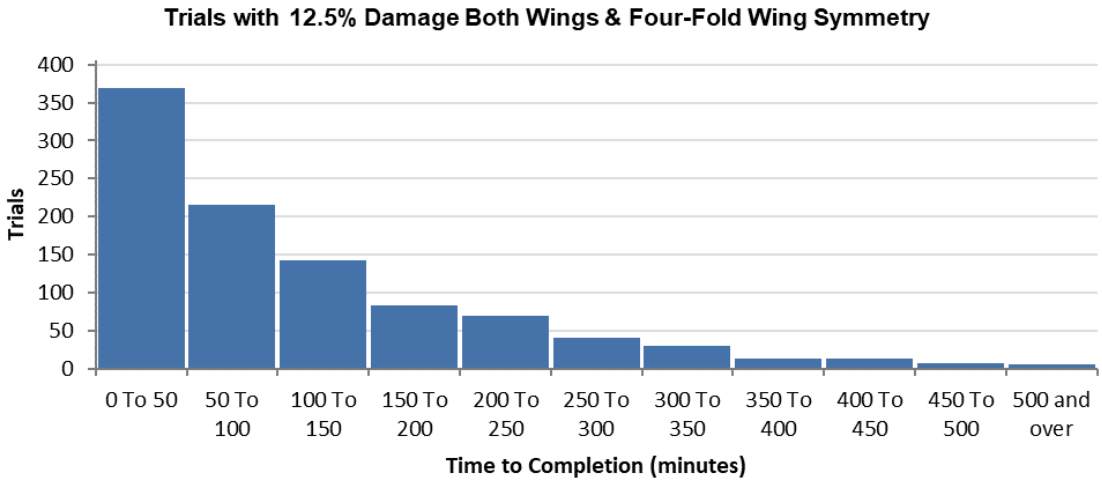


Figure 4.12: Number of Trials vs. Time to Completion with Four-Fold Wing Symmetry - Up to 12.5% Damage to Both Wings

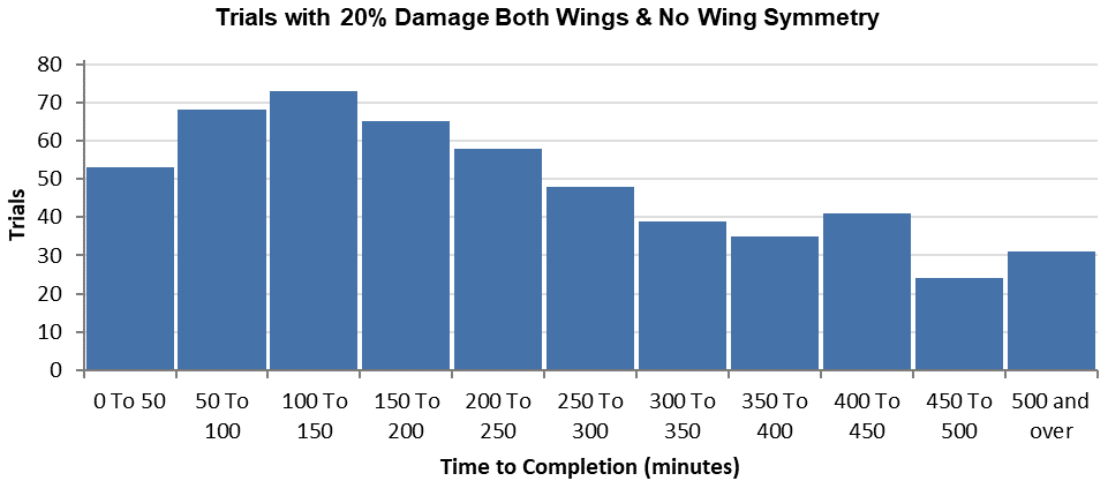


Figure 4.13: Number of Trials vs. Time to Completion without Wing Symmetry - Up to 20% Damage to Both Wings

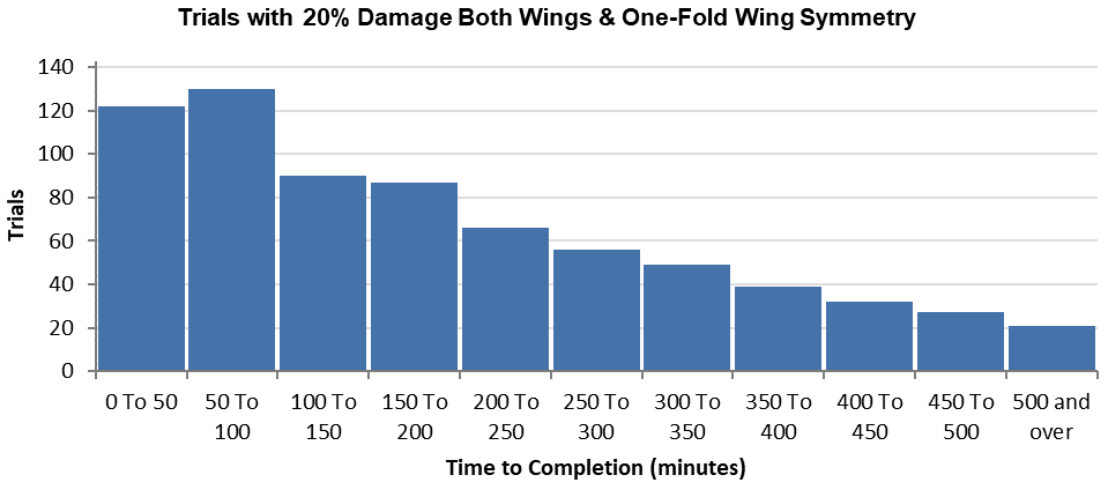


Figure 4.14: Number of Trials vs. Time to Completion with One-Fold Wing Symmetry - Up to 20% Damage to Both Wings

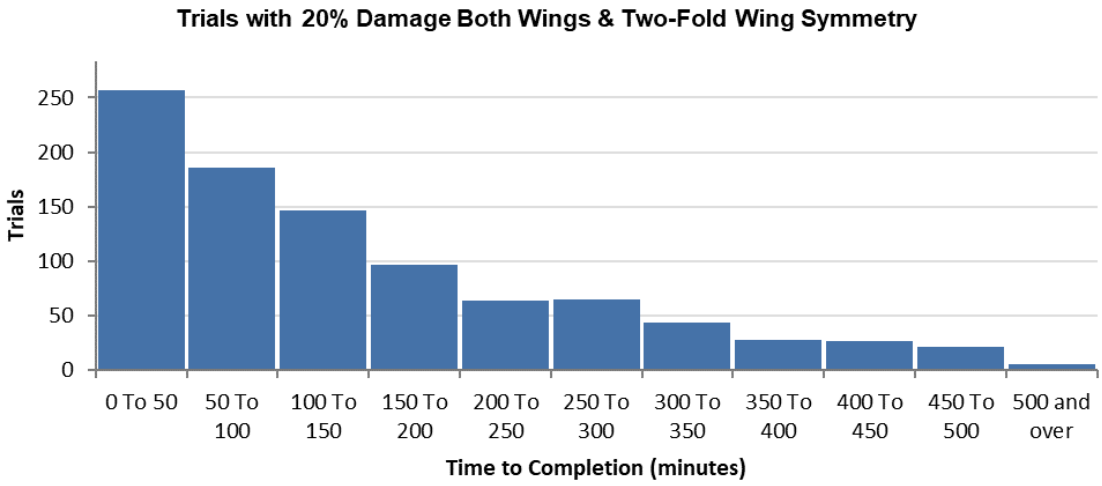


Figure 4.15: Number of Trials vs. Time to Completion with Two-Fold Wing Symmetry - Up to 20% Damage to Both Wings

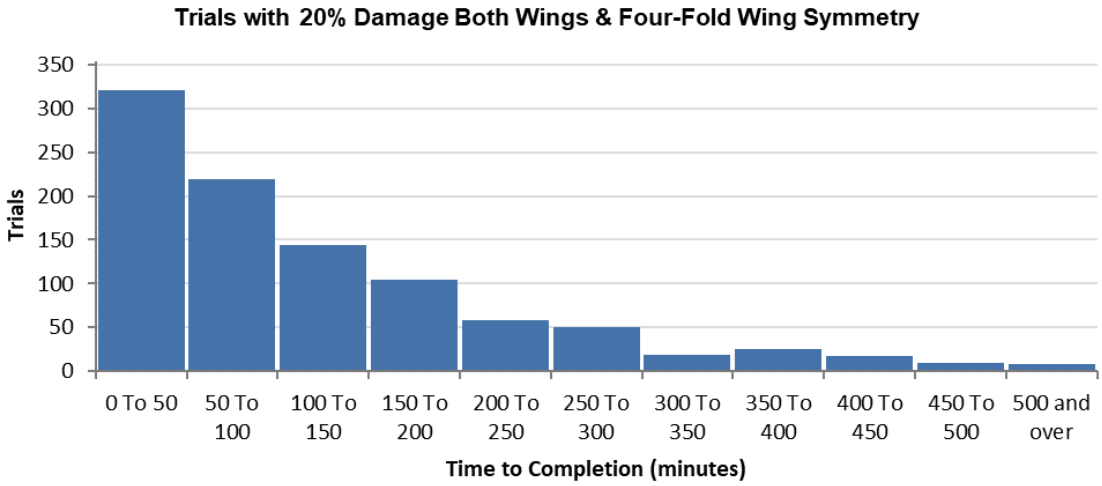


Figure 4.16: Number of Trials vs. Time to Completion with Four-Fold Wing Symmetry - Up to 20% Damage to Both Wings

Bibliography

[1] Bharath Venugopal Chengappa. A study of evolvable hardware adaptive oscillators for augmentation of flapping-wing micro air vehicle altitude control. Master's thesis, Wright State University, 2010.

[2] D.Doman, M.Oppenheimer, M.Bolender, and D.Sigthorsson. Altitude control of a single degree of freedom flapping wing micro air vehicle. *AIAA Guidance, Navigation and Control Conference*, 2009.

[3] D.J. Foster and R.V. Cartar. What causes wing wear in foraging bumble bees? *The Journal of Experimental Biology*, pages 1896–1901, 2011.

[4] Georges R. Harik, Fernando G. Lobo, and David E. Goldberg. The compact genetic algorithm. *IEEE Trans. on Evolutionary Computation*, 3:287–297, November 1999.

[5] H.P.Schwefel. Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of organic evolution. *Annals of Operations Research*, 1:165, 1984.

[6] J.Gallagher. An island of fitness compact genetic algorithm approach to improving learning times in swarms of flapping-wing micro air vehicles. *Proceedings of Robot Intelligence Technology and Applications*, 2012.

[7] J.Gallagher, D.Doman, and M.Oppenheimer. The technology of the gaps: an evolvable hardware synthesized oscillator for the control of a flapping-wing micro air vehicle. *IEEE Trans. on Evolutionary Computation*, 16:753–768, February 2012.

[8] J.Gallagher and M.Oppenheimer. An improved evolvable oscillator for all flight mode control of an insect-scale flapping-wing micro air vehicle. *IEEE Congress on Evolutionary Computation*, pages 417–425, June 2011.

[9] J.Gallagher and M.Oppenheimer. An improved evolvable oscillator and basis function set for control of an insect-scale flapping-wing micro air vehicle. *Journal of Computer Science and Technology*, 27:966–978, 2012.

[10] K.Timmerman. A hardware compact genetic algorithm for hover improvement in an insect-scale flapping-wing micro air vehicle. Master's thesis, Wright State University, 2012.

[11] Max F. Platzer, Kevin D. Jones, John Young, and J. C. S. Lai. Flapping wing aerodynamics: progress and challenges. *AIAA Journal 46*, 9:2136–2149, 2008.

[12] R.J.Wood. The first takeoff of a biologically inspired at-scale robotic insect. *IEEE Transactions on Robotics*, 24:341–347, April 2008.

[13] Sanjay P. Sane and Michael H. Dickinson. The control of flight force by a flapping wing: Lift and drag production. *Journal of Experimental Biology*, 204:2607–2626, April 2001.

[14] Sanjay P. Sane and Michael H. Dickinson. *The Aerodynamic Effects of Wing Rotation and a Revised Quasi-steady Model of Flapping Flight*, volume 205. April 2002.

[15] Sanjay P. Sane and Michael H. Dickinson. The aerodynamic effects of wing rotation and a revised quasi-steady model of flapping flight. *Journal of Experimental Biology*, 205:1087–1096, April 2002.