Gallagher JC, Oppenheimer MW. 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(5): 966–978 Sept. 2012. DOI 10.1007/s11390-012-1277-1

An Improved Evolvable Oscillator and Basis Function Set for Control of an Insect-Scale Flapping-Wing Micro Air Vehicle

John C. Gallagher[1], *Senior Member, IEEE*, and Michael W. Oppenheimer[2], *Senior Member, IEEE*

[1]*Department of Computer Science and Engineering, Wright State University, Dayton, Ohio, 45435-0000, U.S.A.*

[2]*Control Sciences Branch, Air Force Research Laboratory, WPAFB, Ohio, 45433-7531, U.S.A.*

E-mail: john.gallagher@wright.edu; michael.oppenheimer@wpafb.af.mil

Received August 31, 2011; revised July 17, 2012.

Abstract This paper introduces an improved evolvable and adaptive hardware oscillator design capable of supporting adaptation intended to restore control precision in damaged or imperfectly manufactured insect-scale flapping-wing micro air vehicles. It will also present preliminary experimental results demonstrating that previously used basis function sets may have been too large and that significantly improved learning times may be achieved by judiciously culling the oscillator search space. The paper will conclude with a discussion of the application of this adaptive, evolvable oscillator to full vehicle control as well as the consideration of longer term goals and requirements.

Keywords evolvable and adaptive hardware, micro air vehicle, evolutionary algorithm

1 Introduction

The technical challenges inherent in building insect-scale Flapping-Wing Micro Air Vehicles (FW-MAVs) are daunting. Not among the least of those challenges is recovery of flight control precision in vehicles with damaged air frames. Previous work examined an adaptive altitude controller for a simple insect-sized FW-MAV that hybridized mathematically rigorous control theory with a compact evolvable and adaptive hardware (EAH) oscillator[1-3]. The goal of that hybridization was to provide the target vehicle with the ability to correct for vehicle anomalies via flight control adjustment in-flight and while conducting normal missions. Such an adaptive learning capability is considered vital due to the potentially significant manufacturing variability in the vehicles as well as the possibility that the delicate insect-scale structures might be subjected to physical damage and wear while in service.

The adaptive hardware oscillator originally presented[2] was intended as a proof-of-concept and supported altitude control only. This paper presents an improved oscillator design that will be capable of supporting all anticipated vehicle flight modes (e.g., roll control, pitch control, forward translation) This new oscillator is tested, in simulation, on a two degree-of-freedom (2-DOF) control task that requires simultaneous regulation of vehicle altitude and roll and online learning to restore appropriate flight behavior in the face of vehicle damage. It has also been recently discovered that the set of basis functions employed by the adaptive engine inside the oscillator may have been too large for the actual needs of the control problem. This is significant because judiciously restricting the search space may provide significant gains in learning time and circuit resources. This paper will provide experimental evidence demonstrating that smaller basis function sets deliver improved learning performance. It will begin with a description of the vehicle and a brief synopsis of previous work that focuses primarily on a conceptual view of the operation of the older oscillator. It will then discuss the changes necessary to that oscillator needed to enable control adaptation for all flight behaviors of which the vehicle is capable. Following, the paper will provide an example of the new oscillator being used to correct wing-motion behavior in a 2-DOF altitude and roll controller applied to vehicles with simulated damage as well as with judicious reductions in search space in place. The paper will conclude with discussion of intended future work and outstanding challenges.

2 Background and Previous Work

2.1 Evolvable and Adaptive Hardware

Evolvable and adaptive hardware (EAH)[4] is an

Regular Paper

This material was assigned a clearance of CLEARED on 16 Dec. 2011 with case number 88ABW-2011-6488.

emerging subspecialty within evolutionary computation (EC)[5-7] in which one evolves designs for mechanical, computational, or electrical devices. EAH practitioners use search algorithms based on natural evolution to assemble mechanisms from sets of basic building blocks. In the context of controller design, the key difference between conventional automated controller tuning and EAH methodology is the amount of control the automated technique exercises over the final controller design. Conventional automated parameter tuning is limited to adjusting parameters of a human designed device. EAH is allowed significantly more design freedom and may combine solution building blocks in ways that defy any specific design paradigm. For example, a human-designed, computer-tuned, linear proportional feedback controller would remain a linear proportional feedback controller no matter how much its parameters were adjusted. Contrast this with an EAH control device that might come to evolve to operate like a linear proportional controller, but could just as easily have evolved to operate as a linear quadratic regulator or even in ways that defy conventional description. In this work, an EAH oscillator will replace a traditional oscillator inside a conventionally defined flight controller with the goal of restoring whole-system control efficacy in the face of damage to the vehicle. This replacement of a component, rather than the whole controller, with EAH hardware also allows us to limit the amount of risk to the system. Discussion on risk limitation specifically can be found in [2].

2.2 The Vehicle

The Micro Air Vehicle (MAV) considered here is a variant of the Harvard RoboFly[8]. The original RoboFly uses a single piezoelectric actuator to drive identical forward and backward stroking of two wings. The vehicle considered here (Fig.1) employs two independent piezoelectric actuators, each of which independently drives the motion of one wing. The basic wing motion can be best perceived in the top view (left top portion of Fig.1), which represents the wings as 15 mm lines extending from either side of the body. These lines represent wing span spars that can be independently moved to angles ϕ_L and ϕ_R for the left and right wings respectively. The triangular wing planforms (wing shapes) shown in the front view in Fig.1 hang down from the support spars, to which they are passively hinged. As the wing spars stroke forward and backward, dynamic air pressure lifts the triangular planforms to an angle α under the plane of the spars (see the side view in the lower right hand part of Fig.1). Complete kinematics and dynamics of this modified two-effector vehicle can be found in [9] and [10]. Those same sources show how idealized versions of this specific vehicle can be controlled at up to five degrees of spatial and rotational freedom employing a cycle-averaged approach that supplies specially shaped periodic waveforms to each wing. Previous and current work explicitly presumes that the vehicle's center of mass corresponds to the geometric center of the body and that accurate vehicle altitude and roll measurements are available.

2.3 Cycle-Averaged Split-Cycle Cosine Control

All control devices considered in both previous and

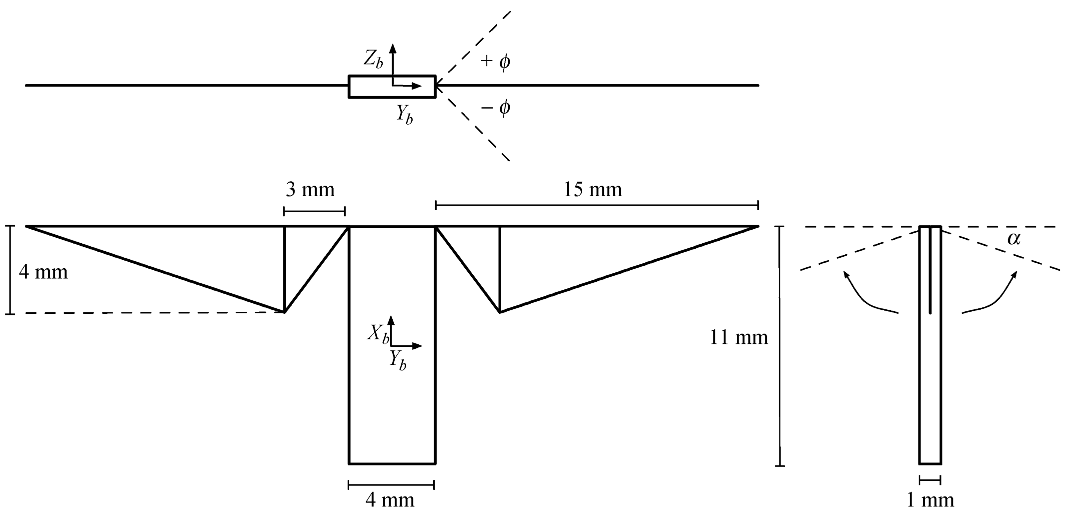

Fig.1. Orthographic view of flapping wing MAV. Both wing spars are restricted to rotational motion about their joints with the body and in the Y_b/Z_b plane. The range of those rotations is $[-1..1]$ radians. As the spars rotate, dynamic air pressure lifts the triangular wing segments up to an angle of α radians under a base vector embedded in the Y_b/Z_b plane.

current work are *cycle-averaged*. Cycle-averaged controllers close the vehicle control loop over complete wingbeat cycles of the vehicle. Rigorous derivations of the cycle-averaged (whole wingbeat) wing-applied vehicle forces and torques can be used to create a model relating specific sinusoidal and modified sinusoidal wingbeat patterns to vehicle forces and torques. With those relationships, it becomes fairly straightforward to close a feedback control loop around a desired vehicle attitude and position by actuating torques and forces applied to the body. Cycle-averaged control is desirable because the computation of control efforts on a once-per-wingbeat basis is far less resource intensive than equivalent computations at a finer time scale. Updates of wing flap frequencies would be, therefore, very infrequent even from the perspective of a simple control computer operating at a few MHz. Cycle-averaged control, however, does come with some serious challenges that could render it impractical on a real vehicle. Three of these are limited precision of onboard Digital-to-Analog and Analog-to-Digital converters, fidelity of vehicle models underlying controller derivations, and the accumulation of vehicle damage and wear that could introduce further model/vehicle mismatches. These issues were considered more completely in [2] and [11], but ultimately are variants of the same problem — mismatches between the actual and the modeled relationships between body torques and forces and wingbeat patterns. Evolvable Hardware adaptation is intended to tune those wingbeats to restore the expected relationships between control efforts and force/torque generation which allows the original cycle-averaged flight controllers to be used on damaged or otherwise compromised vehicles.

The specific form of cycle-averaged control considered here is *split-cycle control*. In split-cycle control, each wing is provided a wingbeat frequency and a waveform shape parameter at the beginning of its wing stroke. In the current generation of controllers, the wing motion envelopes are 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/s, and whose downstroke phase (motion from −1 radians back to 1 radian) 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. Fig.2 illustrates the nature of a split-cycle cosine. The upstroke and downstroke wing motions indicated by that split-cycle cosine are defined as:

$$\phi_U = \cos\big((\omega - \delta)t\big), \tag{1}$$

$$\phi_D = \cos\big((\omega + \sigma)t + \xi\big), \tag{2}$$

where δ is the frequency offset that defines how much the upstroke phase is impeded or advanced and σ and ξ, defined below, characterize a downstroke consistent with the previously described split-cycle philosophy.

$$\sigma = \frac{\delta\omega}{\omega - 2\delta}, \tag{3}$$

$$\xi = \frac{-2\pi\delta}{\omega - 2\delta}. \tag{4}$$

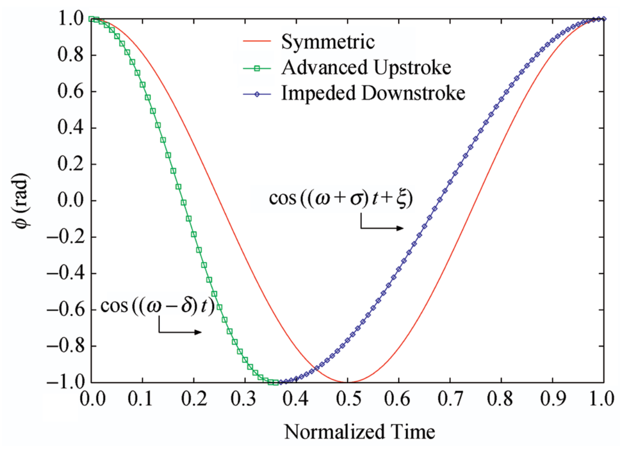

Fig.2. Split cycle cosine.

With split-cycle control as defined here, it is possible to define cycle-averaged control laws to govern motion of up to five degrees of whole vehicle motion. The special case of altitude-only control requires only modulation of wing beat frequency of a pure cosine (zero split-cycle shift). Control of more than just hover requires active control of both frequency and split-cycle delta shift.

2.4 EAH Augmented Control

The work in [2] introduces an evolvable and adaptive hardware (EAH) oscillator that corrects altitude control precision problems by learning wingbeat motion schedules that allow a specific damaged vehicle to match the force-to-frequency mapping implicit in the altitude command tracking controller (ACTC). The rationale is that, although convenient for mathematical analysis and correct for idealized vehicles, pure cosines might not produce correct mappings between vehicle force and torque and wing motion when the vehicle has suffered physical damage by any means. The split-cycle cosine oscillator driving each wing, therefore, is replaced with an adaptive evolvable hardware oscillator that learns wing beat motion functions for each wing that improve the vehicle's ability to match flight profiles that arise during normal flight. Although these patterns are evolved based only on samples of measured error between desired and actual vehicle attitude and position in flight, there is strong experimental evidence that the system learns wingbeat motions that restore the mappings between controller commands and the forces and torques generated in response[11].

The EAH oscillator maintains a library of sampled, pre-computed basis functions. Each wing's runtime motion waveform is the arithmetic average of eight pre-computed and sampled basis functions selected from a set of 4 096 bases. An on-board evolutionary algorithm learns the eight basis function indices for each wing, resulting in a genome of 16 integers in the range of [0..4 095]. One can view this genome as the concatenation of two eight-element multisets, resulting in approximately 4×10^{24} unique wing oscillation patterns. A full explanation of the rationale behind the basis functions is included in [11]. Here it is sufficient to note that the basis functions were specifically chosen to promote three goals: minimization of the chances of crashing; avoidance of complicating the analysis of any augmented controllers that evolve; and avoidance of a need for complicated floating-point computations. A simple digital circuit to produce oscillations subject to those constraints is provided in [2]. We will first focus on the operation of the old style signal generator as a prelude to discussing the modifications necessary to support a full palette of flight control modes.

2.5 Old Style Oscillator

A simplified Register Transfer Level (RTL) data path description of the signal generator is given in Fig.3. In its original form, the circuit takes in a single eight-bit wing flap frequency (ω) that is used for both wings and two eight-bit split-cycle shift values (δ_L and δ_R). These values are stored in registers in circuit areas A and B as indicated in Fig.3. The register DC is a free-counter tied to a system clock that advances once per clock tick. The register is sized in tandem with that clock so that the upper eight bits of DC increment once every base time step of the oscillator. The period of wingbeat position updates in terms of the base time step is computed from OF by a wired reciprocal operation, and when the running counter is equal to this value, it increments the TCC register, an eight-bit counter. The end effect is that TTC increments at evenly spaced time intervals 256 times during the actual period of a vehicle wing flap. This portion of the address to the Wave Table ROM (basis function library) is intended to "step through" the waveform so that individual wing position updates can be computed and presented to the wings.

One will note that the Wave Table ROM, which stores the precomputed basis functions as described in [11], derives part of its address from TCC and part from a component titled the "Shuffle LUT RAM" (Shuffle LookUp Table RAM). The Shuffle LUT maintains a file of 16-element basis function identifiers, eight for each wing. These are the identities of the precomputed basis functions that will be combined to produce the driving function for each wing. In this original version of the oscillator circuit, the intent was to evolve and store a shuffle vector for each ω, δ_L and δ_R. This is the explanation of circuit block B, which merely uses controller provided delta shift values to select one of a set of learnable compositions of basis functions. Subsequent sensitivity analyses of the controllers providing wing ω, δ_L, and δ_R rendered this approach infeasible. The granularity and range of adjustments necessary to the δ_L, and δ_R, values for practical vehicle control preclude storage in a lookup table one could easily accommodate on-board the vehicle.

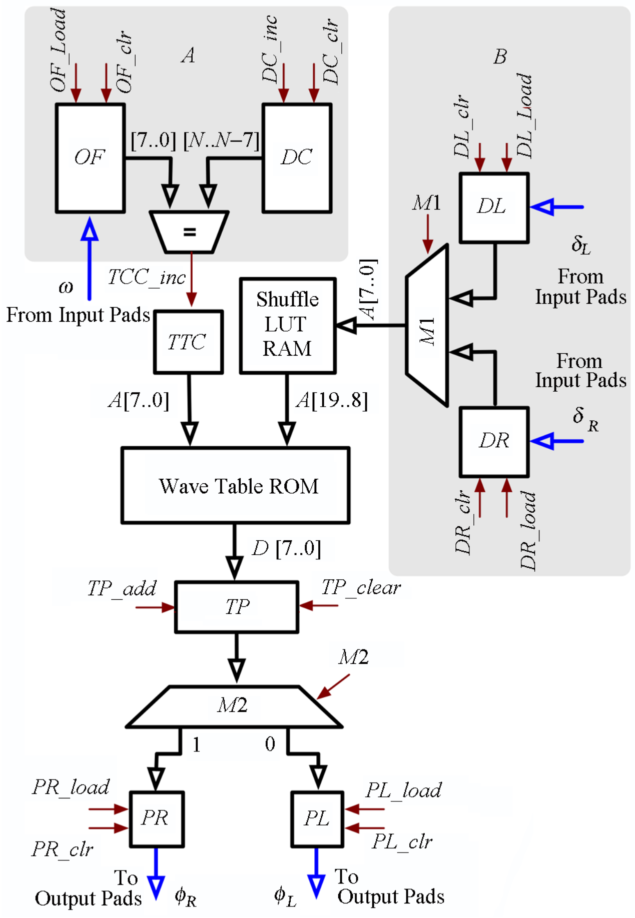

Fig.3. Old style wing motion signal generator.

The remainder of the circuit consists of a simple adding register and a multiplexer. As a new wing position is needed, eight wave table ROM lookups are made per wing, these are added in TP subsequent to a wired logical shift to affect a division by eight. These wing position values are multiplexed into the PR and PL registers to drive wing position directly. Since the vehicle wings and the *TCC* count events operate in the range of KHz, even a slow digital system clock of 1 MHz will not produce significant skew in wing position outputs.

Due to both granularity and combinatoric issues in addition to the need for an ω_L and ω_R unique to each wing, circuit areas *A* and *B* require significant redesign to support non hover-only flight.

3 New Evolvable Oscillator

3.1 Theory of Operation

The most direct means of accomplishing split-cycling of arbitrary, table-defined, wing position functions ($\phi_L(t)$, and $\phi_R(t)$) is to employ different update intervals between wing position updates in the upstroke and downstroke phases of the wingbeat. Fig.4(a) illustrates wingbeat motion generation as accomplished in the old circuit which used a fixed interval size between individual wing position updates. The original circuit employed fixed duration time steps between subsequent updates of the cosine. One can accomplish split-cycling by employing different update intervals for the up and down strokes of the wing as illustrated in Fig.4(b). As opposed to the conceptual operation illustrated in Fig.4(b) that uses two groups of five intervals each, the real circuit uses two ranges of 127 intervals each. One would compute and use one interval for the first 127 wing motion updates and another interval for those updates remaining after. The sizes of the update intervals required to achieve a specific split-cycle shift can be computed arithmetically with a purely combinational digital circuit.

Fig.5 is an updated datapath that would be dropped into the circuit in [2] to create the new EAH oscillator. Area *B* in the new circuit has the same conceptual function as it did in the earlier circuit, except this time it, via multiplexer, routes stored δ_L and δ_R values through a combinational circuit (*CONV_B*) that outputs the upstroke and downstroke intervals respectively. Area *A* of Fig.5, as with Area *A* of Fig.3, is responsible for timing of individual generations of wingbeat motion function points. It is bilaterally symmetric and duplicates the same timing functionality for the left and right wings respectively. As needed and under the control of a microcontroller (not shown), the left and right *FS_Inc* and *BS_Inc* blocks would be loaded with their appropriate "front side" and "back side" intervals. For the first 127 wing position updates, the *FS* interval would be routed through the multiplexer into the left and/or right *D_Count* registers as needed. The *D_Count* registers are downcounters, and when they reach zero, the corresponding left or right *TCC* register is incremented. The left and right *TCC* registers serve the same function they do in the old circuit, and drive an identical waveform generation block to create the actual wing position updates. In this new circuit, the Shuffle LUT RAM becomes the working memory of the evolutionary algorithm's population of candidate wing motion functions as opposed to a working store of split-cycle parameter specific compositions.

This basic circuit template provides asynchronous updating of left and right wing time intervals and independent setting of left and right wing split-cycle parameters. These are necessary capabilities for advanced flight controllers. Also, by choosing the sizes of the various timing registers and driving clocks appropriately, one can arbitrarily tune the amount of precision in which one may specify split-cycle parameters.

4 Experimental Verification

4.1 A 2-DOF Roll and Altitude Controller

The operation of the new adaptive oscillator is verified by placing it inside a 2-DOF controller derived specifically to test the oscillator's ability to learn wingbeat motion functions for a multi-degree-of-freedom

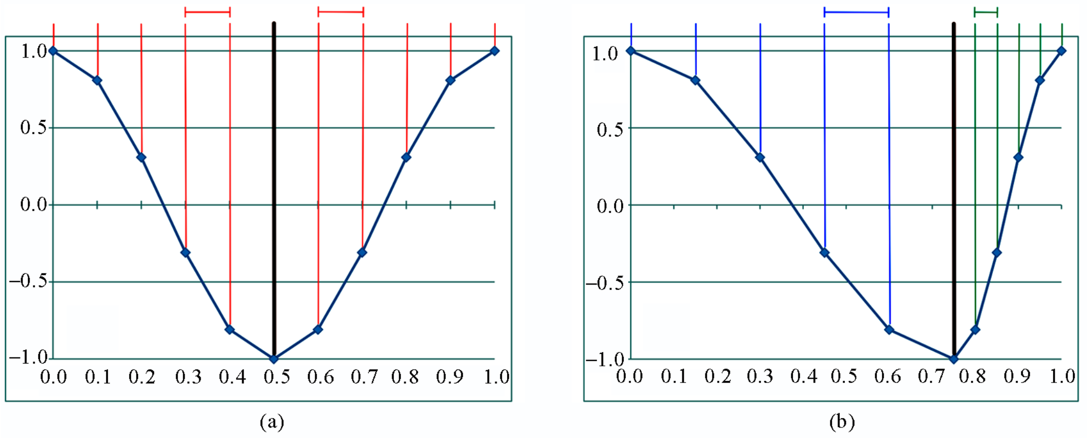

Fig.4. Conceptual old and new style wing position generation.

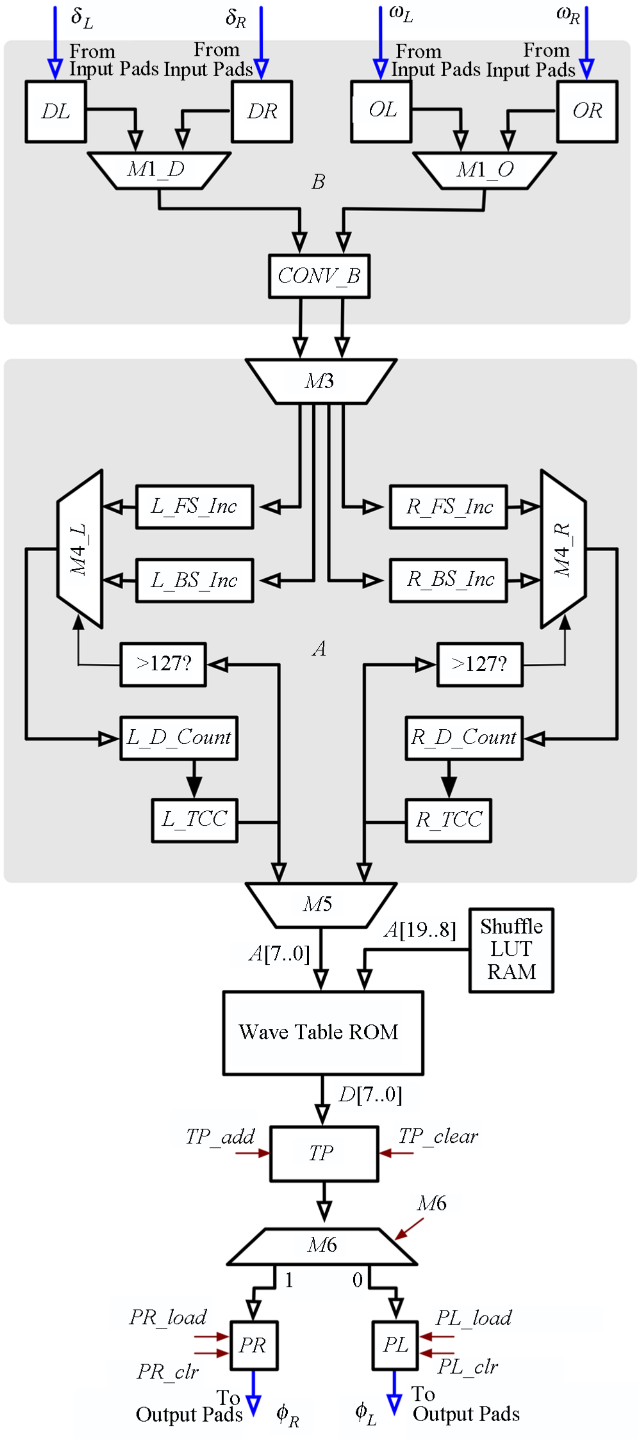

Fig.5. New style wing motion signal generator.

control adaptation task. In this work, the FW-MAV is constrained to move up and down a single guide wire. This means the vehicle can translate up and down the wire (altitude) and it can rotate around the wire (roll). Fig.6 presents a block diagram of the plant and controller models that includes calculation of body forces and moments as well as independent altitude and roll feedback controllers. Note that the Roll Command Tracking Controller (RCTC) at the top of Fig.6 is nearly identical in function to the previously discussed Altitude Command Tracking Controller (ACTC)[12] at the bottom of Fig.6. The RCTC uses an explicit model of the relationship between desired cycle-averaged body roll moment ($\overline{M}_{x_{\rm des}}$) and split-cycle delta shift ($\delta_{M_x}(t)$) to provide the oscillator with an appropriate delta shift at the beginning of each wing beat. Fig.6 replaces a simple cosine oscillator with a *Symmetric Shift Split-Cycle Oscillator*, which simply produces two wing beat signals, one with the provided delta shift, and the other with a delta shift equal to the negative value of that provided. In effect, this produces a "push/pull" relationship between the wings and produces rotational moments around the body's central axis. In the case of combined roll and altitude control, the RCTC uniquely determines delta shift and the ACTC uniquely determines wing flap frequency. At the level of ACTC/RCTC control, beat frequency and delta shift are decoupled, so there is no need to conduct controller allocation operations and the two controllers can operate in parallel. Note that the symmetric shift split-cycle oscillator is a special case of the circuit given earlier in this paper and can be achieved by wiring both frequency inputs to the same source and by appropriately constraining the split-cycle shift inputs.

4.2 Experimental Setup

The vehicle and oscillator simulations were custom coded in *C*. The vehicle model was verified against a MATLAB implementation of the original vehicle models prior to use. In each experiment, a faulty vehicle is simulated by randomly modifying the drag and lift produced by each wing by a random factor of 50% to 150%. This results in a wide variety of mismatches between expected and actual generated upward force and roll torques. It also produces these faults at a level of abstraction below that of cycle-averaged modeling or the actions of the ACTC or RCTC. This form of the experiment also represents a far more aggressive test than those previously tested against fixed fault conditions.

The learning engine (evolutionary algorithm (EA)) used in this work is a variant of the mini-population (MiniPop) genetic algorithm[13]. This learning implements a small, but user selectable, number of stochastic hill climbers. Each hill is centered around a bitstring that is the best candidate seen on that hill to date. Search progresses by running a tournament for each hill that pits the current champion against a mutant version of itself with the winner taking the champion slot of the island. At the end of each round of tournaments

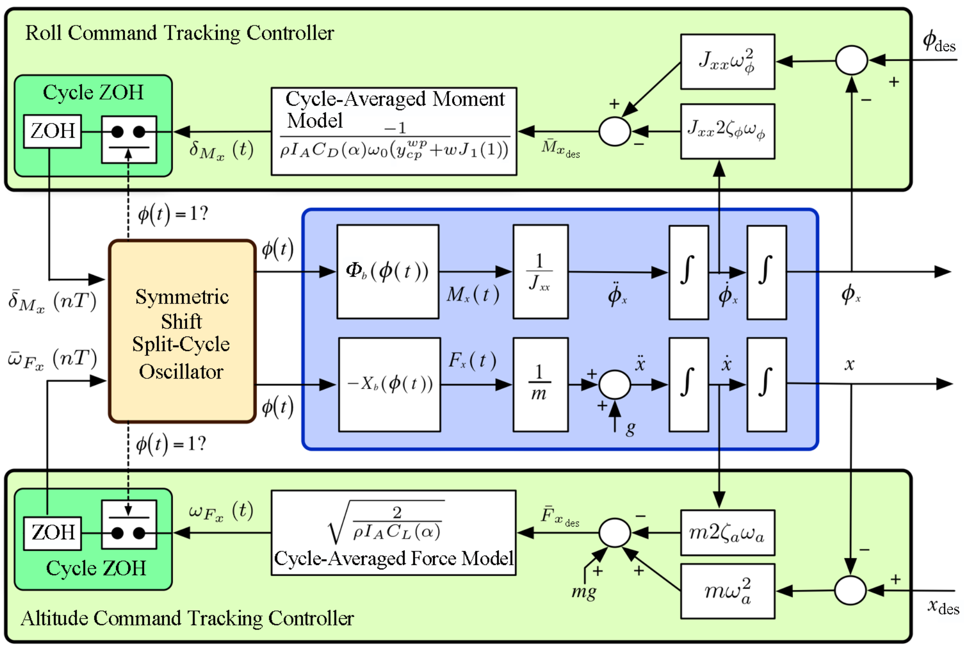

Fig.6. 2-DOF control: Altitude and roll command tracking controllers.

on all islands, a hypermuation may occur with a probability selected by the user. If a hypermutation occurs, the island containing the worst performing genome is randomized. This algorithm is purely mutation driven. Although it is possible for all islands to occasionally contain the same bitstring, the continuous mutation-based tournaments will end any such "convergence events" unless that condition actually represents a local or global error minimum.

This learning algorithm was selected because it requires very little chip area and no expensive floating-point operations[14]. For use onboard an FW-MAV, the following modifications have been made:

1) The bitstring initialization in the original is changed from random bits to LUT encodings for the standard split-cycle cosine generation.

2) The hypermutation tournament is removed and replaced with a best-to-worst drift operation that, with a user-selected probability, overwrites the worst LUT encoding in the population with the encoding for the best LUT in the population.

3) A wing-beats per evaluation parameter is added to allow evaluation scores to be taken after a user-selected number of wing beats. This provides both more accurate estimates of the quality of a candidate LUT and a possible ability to correct for phenomena seen only after multiple wing beats.

4) Mutation rates are expressed as an expected number of bits to mutate per genome.

5) The learning loop never terminates, as learning is conducted inline with actual control. A user might add a flag variable that turns off updates to the LUT after a certain number of evaluations or upon satisfaction of some other condition.

6) Dummy evaluations of the best genome seen to the current time in an ongoing search are inserted prior to any evaluation of a candidate that is scored and used for search. These spare evaluations compensate for deceptively low scores that could result by evaluating a good candidate right after a very bad candidate. This heuristic is less expensive in time than increasing the evaluation period.

A MAV_MINIPOP hardware implementation would consist of a state machine in which all operations are indexed to a state counter controlled by the wing position generation circuitry. The data path required to implement MAV_MINIPOP would be identical to that presented in [14]. An appropriate microcontroller can be trivially derived from a description of the algorithm by conversion into an algorithmic state machine and subsequent expression as a microcoded sequence controller. The new microcontroller would be approximately the same physical size as that used for standard MINIPOP. These learning components, combined with the LUT-based wing position generator constitute a complete EAH augmented synthesized oscillator.

Evolutionary runs were conducted online, while the vehicles were in simulated flight, and wing functions were evaluated based on the mean squared error between actual and desired roll angle and altitude over a user definable period of flight measured in wing beats. MAV_MINIPOP learning parameters were set as follows: population size = 8, wing flaps per evaluation = 50, DRIFT = 100%, and MRATE = 16 expected bit flips per mutation event. These settings correspond to the optimal learning settings determined for altitude control only learning[11]. The genome consists of 16 LUT (lookup table) indices (eight for each wing) encoded each in 12 bits. The resulting genome is 192 bits in length. Learning was run online, while the simulated vehicle was in flight, with a termination condition of acquisition of acceptable as measured by absolute position and attitude error. The error score considered "minimally acceptable" was set to correspond to altitude control precision of 1 mm or less and roll angle control precision of 0.5 degrees or less.

4.3 Experimental Results

Using a C language simulation of the vehicle and the underlying adaptive circuitry, 44 973 independent learning trials were conducted using the experimental setup defined previously. Over all 44 973 runs, 86.6% resulted in at least a minimally acceptable controller. The rest had failed to find minimally acceptable controllers prior to the experiments' hard-coded time limit of approximately six hours of vehicle flight time. Table 1 presents experiment success percentages broken down by levels of maximum wing fault. Note that the table rows are cumulative and reflect all runs with less than or equal to the level of maximum error indicated (e.g., the 30% entry also contains the data for the 20% and 10% levels of error).

Table 1. Percent Acceptable Solutions Evolved

Maximum Drag/Lift Error (%)	Trials	Percent Acceptable	Percent Unacceptable
10	1 824	99.7	0.3
20	7 276	99.0	1.0
30	15 734	97.0	3.0
40	28 911	93.0	7.0
50	44 973	86.6	13.4

Table 1 provides percentage yields for different magnitudes of wing fault. In that table, "percent acceptable" is the percentage of controllers that adapted to have performance at least equal to the level defined as acceptable previously in this paper. "Percent unacceptable" is the percentage of controllers that did not learn to perform to that standard, though it should be noted that even unacceptable controllers were capable of flight and did not crash. From Table 1, one can see that yields are very good for moderate to high levels of wing fault damage (up to 30%). They are less impressive, though still encouraging even when near-catastrophic (40% and higher) wing faults are admitted. Preliminary qualitative analysis of controller learning at very high levels of wing fault reveal that the largest difficulty the evolutionary algorithm (EA) encounters is that so little (or so much) upward force is generated that the internal oscillators would need to operate outside their designed frequency limits to produce correct forces. One could argue that most failures at very high levels of wing damage are due to insurmountable physical limitations of the vehicle or a lack of appropriate basis functions in the basis function library. On the other hand, the failures to learn at low to moderate levels of wing faults are very likely due to shortcomings in the learning algorithm. This is an issue to be addressed in future work.

Table 2 shows the average and 75th percentile (Q3) times required for the vehicle to achieve minimally acceptable performance as defined earlier. For this table, non-acceptable controllers were culled to give a picture of how long MAV_MINIPOP requires to achieve acceptable controller performance in those cases when it does function acceptably.

Table 2. Time to Evolve Acceptable Solutions

Maximum Drag/ Lift Error (%)	Acceptable Trials	Average Time (min)	Q3 Time (min)
10	1 818	49.36	67.08
20	7 204	55.68	73.98
30	15 734	62.32	82.61
40	26 895	69.00	92.27
50	38 937	74.72	102.52

Table 2 indicates what might be unacceptable amounts of vehicle flight time required to correct for vehicle faults. At moderate levels of wing damage (30%), one, on average, has to wait for the vehicle to be in flight for about one hour before minimally acceptable control (as defined earlier) is achieved. These vehicle would most likely make relatively small flights punctuated by periods of passive station-keeping for energy recharge or to serve mission objectives. Therefore, one would desire to keep learning times as short as possible. The above results are encouraging, but more sophisticated learning will be needed in the future.

4.4 Experimental Results: Typical Flight Trajectories

Figs. 7 and 8 show the flight trajectories for a broken vehicle (left wing has 50% reduced lift and drag) and the same vehicle after adaptive oscillator learning had concluded. The vehicle was commanded to simultaneously follow the target altitude and roll trajectories shown in Figs. 7 and 8. Note that the trajectories represented in the figures are different from that used during learning, suggesting that the adaptive oscillator is not over learning to a specific flight profile. The performance shown is typical of that of all minimally acceptable controllers under all wing fault conditions tested.

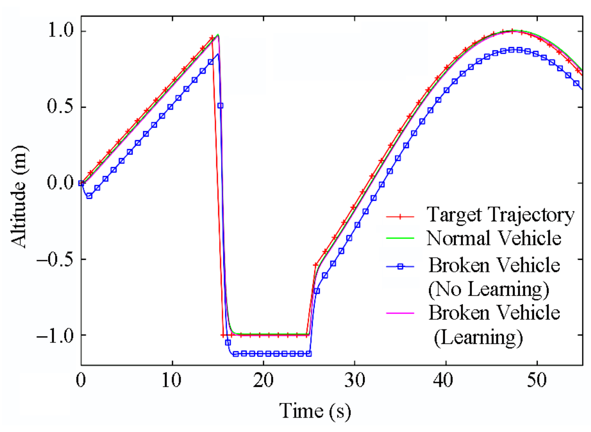

Fig.7. Typical altitude control.

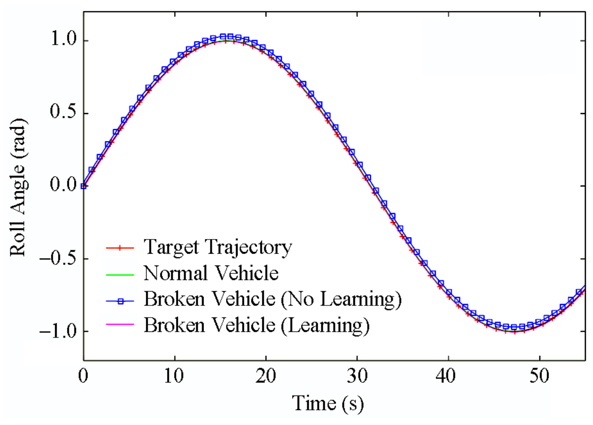

Fig.8. Typical roll control.

5 Minimization of Basis Functions and Structured Search

The main contribution of this paper is that an EAH oscillator is feasible in this application both in terms of implementability as a simple digital circuit and in terms of efficacy in solution yield and learning time. That being said, it must be noted that learning times on the order of hours, though encouraging from the perspective of feasibility, may not be sufficient for practical operational systems. It must also be said that the EA used here is little more complicated than a community of stochastic hill climbers. Again, that a simplistic approach does so well is interesting from a feasibility standpoint, but one would be remiss in not observing that there is room to improve. There are two obvious means of improving learning times. First, one may cull the size of the basis function set (currently 4 096 functions) in an attempt to shrink the size of the search space that must be considered during in-flight learning. Presuming that there are operational redundancies in the basis function set, one could imagine this could be done with no harm. Second, one can improve the EA algorithm itself. There are obvious EA techniques that have not yet been brought to the table. Both are currently under consideration. Here we will present some interesting observations with respect to culling of the basis function library that have bearing on both approaches and may inspire future improvements.

First, consider the structure of the basis function set. The basis function set contains 4 096 precomputed wing motion schedules themselves partitioned into 16 distinct shape families. Each shape family is defined as a combination of two members of a smaller core of four basic core shapes. The four core basis functions (bases A though D) are given below:

$$A(x) = \cos(x), \tag{5}$$

$$B(x) = \frac{\cos(x) + \cos(3x)}{2}, \tag{6}$$

$$C(x) = \frac{2\cos(x) + \cos(3x)}{3}, \tag{7}$$

$$D(x) = \frac{4\cos(x) + \cos(3x)}{5}. \tag{8}$$

These four basis functions were chosen for the following reasons: 1) They all satisfy the condition that the wings be fully forward ($\phi(t) = 1.0$ rad) at the beginning and end of each wingbeat; 2) They all represent a mostly normal cosine wingbeat with more or less aggressive superimposed mini-wing beats; 3) They encapsulate non power-of-two divides and multiplies into pre-computed basis function tables.

One is free to combine the upstroke and downstroke split-cycles of more than one core basis function. Fig.9 illustrates the 16 possible pairwise combinations of the four core basis functions. In addition, one may compute an arbitrary number of δ shifted versions. In this paper, each of the 16 mixed core basis functions comes in one of 256 shifted varieties, with the shifts being evenly spaced between $\delta = [-1.5..0.38]$ rad/s. This results in a set of basis functions of cardinality $16 \times 256 = 4\,096$. In the reference circuit implementation, 256 elements of each of the 4096 functions are computed at eight bits of precision and concatenated into an ROM, resulting in a 1 MB ROM lookup table.

The computation required to average these functions at run time is trivial so long as one chooses to sum and subsequently divide a number of values equal to a power of two. In this work, the actual functions driving wing motions are the average of eight of the basis functions.

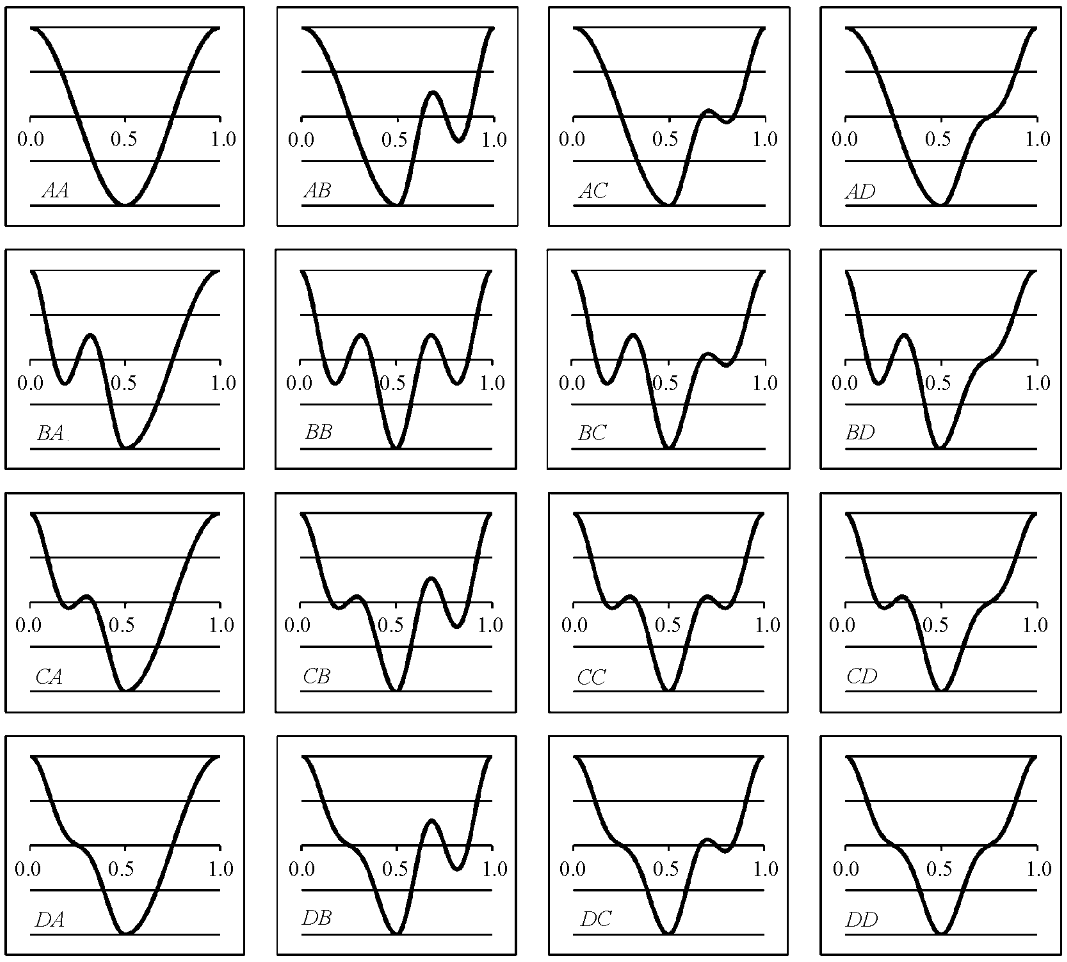

Fig.9. 16 pairwise combinations of core basis functions.

A single serial adder circuit and an inexpensive shift register are all that are required to compute runtime wing positions. In this paper, the left and right wings are driven by different oscillatory signals which are each stored as eight indexes into the 1 MB table of pre-computed functions. The oscillator, rather than playing a simple cosine, reads eight basis function outputs per wing, averages these values, then commands the resulting wing positions. Circuitry to accomplish this is given in [2]. Note that with this circuit, the $\phi(t)$ function driving each wing is represented by a lookup table index multiset of size eight selected from a pool of size 4 096. This results in approximately 1.978×10^{24} unique possible $\phi(t)$ functions. For a two-winged vehicle, therefore, there are approximately 4×10^{24} possible composed split-cycle oscillators under this scheme. This is quite obviously a huge search space, and it is legitimate to ask if a restricted, easier to search, space would still contain sufficiently many operationally fit flight solutions.

As an initial exploration, we ran five hundred randomized learning trials of a modification of the FW-MAV algorithm used elsewhere in this paper with a randomized wing torque/force error of 30% applied to each wing. The modification was to replace the replacement of the worst individual in the given algorithm with a randomly selected worst-with-best replacement or a hypermutation. In these new runs, the adaptive oscillator was only allowed to construct solutions using one of the 16 basis function families. Instead of having a pallet of 4 096 basis functions, each of these 16, 500 trial, sets had a different pallet of 256 basis functions. It is not surprising that hypermutation applied so liberally would be disruptive to search and we would see increases in learning time. Indeed, in a run of this modified algorithm using the whole set of 4 096 basis functions, the average flight time required to find an acceptable solution increased to an average of 5.56 hours. On a positive note, yield increased from the 97% reported earlier in this paper to 100%. The surprising results were seen in the reduced scope runs. One basis function family (AC from Fig.9) produced acceptable solutions with 100% yield and an average learning time of about 40 minutes, even when using the same highly disruptive hypermutation operator that slowed whole basis set learning times to 5.56 hours. The full results of the 16 runs are given in Table 3. The row and column labels correspond to basis function grid given in Fig.9. Kruskal-Wallis ANOVA testing[15] on all 16 experiment sets represented in Table 3 allows us to reject the null hypothesis that the average learning times are drawn from the same underlying distribution (99% confidence level).

Table 3. Average Minutes to Solution for Restricted Basis Function Sets

	A	*B*	*C*	*D*
A	75.9	65.8	40.0	89.7
B	163.5	368.3	152.4	138.5
C	64.3	80.9	62.0	88.1
D	252.1	46.3	77.7	253.2

Note that for this problem, we apparently did not need the vast majority of the basis function library to find quality solutions. At this juncture, of course, we have no way to know that one will not need the full set when we attack control of more than two degrees of the vehicle's spatial freedom. We do know, however, that experimentation and tuning of the basis function set will likely lead to gains in search performance and should be studied as more complex control schemes are tested.

Two other sets of experiments are worth mentioning in that they shed light on the structure of solution space for this problem. Restricting search to one of the 16 basis function families is one end of a continuum of restriction options. One could just as easily choose any subset of the 16 of any cardinality between 1 and 16. Although testing all of these subsets is cost prohibitive at this time, two were tried. We ran 500 random vehicle damage trials for basis function sets composed of (*AB* and *AC*) and (*AA*, *AB*, *AC*, and *AD*). We ran each restricted set under two conditions. In the first, both hypermutation and mutation varied over the whole set. In the second, although hypermutation varied over the whole set, regular mutation was restricted to the core basis function set that an allele was already in. This created a choice between a "full-range" mutation and a "tiered full-range" mutation on hypermutation events and a localized in-family mutation on tournament mutations. The results here too were telling. The tiered mutation was consistently better in terms of learning time. Table 4 summarizes these values.

There are two salient observations. The first is that tiered mutation seems superior, as it allows global search via hypermutation and localized search during regular tournament mutation. Also note that the *AB*, *AC* tiered mutation set appears to have done slightly better than just AC alone. Kruskal-Wallis ANOVA testing[15] on the four experimental sets represented in Table 4 allows us to reject the null hypothesis that the average learning times are drawn from the same underlying distribution at a 99% confidence level. Also, a Mann-Whitney test[16] between the 69.0 and 39.6 minute datasets allows us to reject the hypothesis that those sets are drawn from the same underlying distribution at a 99% confidence level.

Table 4. Average Time to Solution for Dual and Quad Basis Function Sets

Algorithm Type	Average Time to Solution (min)
AA, *AB*, *AC*, *AD* Full Mutation	176.7
AA, *AB*, *AC*, *AD* Restricted Mutation	69.0
AB, *AC* Full Mutation	122.3
AB, *AC* Restricted Mutation	39.6

6 Related Work

Control of flapping-wing vehicles has attracted considerable interest. Some representative, alternative, conventional approaches to FW-MAV control are described in [17-19]. It is presumed that these traditional efforts could also be augmented with EAH methods as they were in this work, as they are not inherently incompatible with the techniques described in this paper. Various non-conventional, soft-computation approaches to the problem of flapping-wing flight control are in the literature. Representative of these efforts are [20-26]. All of these appear to apply learning at the level of flight-mode control laws and/or controller and vehicle co-evolution, as opposed to our application of learning at a lower level of abstraction underneath a traditionally derived control law. One of these efforts[21] is somewhat similar to the work reported here. The primary differences between that work and our work are: 1) We represented wing motion schedules as compositions of pre-computed basis functions and they represented wing motion schedules as Bezier curves; 2) We employed a custom EA designed to be implementable in small chip area and they used a standard elitism genetic algorithm; and 3) Our work is designed for inflight use on a real, pre-designed, vehicle with attention given to not crashing the vehicle and theirs as focused on coevolving controllers and bodies in simulation with the goal of later construction.

7 Discussion and Conclusions

This paper has demonstrated that EAH methods previously introduced to augment FW-MAV altitude controllers do, with some modification, scale for use in 2-DOF control consisting of simultaneous command of vehicle roll and altitude. At the outset, it was unknown if wing motion shapes that simultaneously matched damaged vehicle behavior to the cycle-averaged models implicit in the ACTC and RCTC were even learnable using the basis functions adopted. The work presented here represents the first evidence that such shapes indeed do exist and can be learned using very simple EA methods during normal vehicle flight. Previous work did not explicitly employ split-cycle manipulations, as such is not needed for pure altitude control. The method for generating split-cycle signals of arbitrary piecewise linear curve approximations is novel, if perhaps somewhat obvious. Previous work was conducted with respect to two specific wing failure modes. This work extended the set to over 40 thousands randomized wing failures and provides, for the first time, evidence that learning successes are not limited to only two special cases of wing failure types. It has been demonstrated that learning yields are very high (97% or better for moderate wing damage) and learning times (one hour flight time on average for moderate wing damage) are reasonable, if not yet as fast as mission planners might desire. We have also demonstrated that practicality issues like the physical size of the digital circuitry required to store the basis set and the actual time required to find acceptable solutions can be effectively attacked via better EA search (including tiered mutation as demonstrated here) and judicious restriction of the basis function set. There is much work to be done on those issues yet, but there is even at this early juncture significant reason to believe that those efforts will be fruitful.

The adaptive oscillator circuit of [2] could be trivially constructed using 1980's or earlier technology and requires a very slow 1 MHz clock. The expanded circuit presaged in this paper would require only slightly more circuitry and a 250 MHz clock to accommodate the finer timings needed for roll control. Once one is willing to presume a 250 MHz clock inside the adaptive oscillator unit, and we must now presume that because of physical necessity, there is less reason to continue avoidance of extensive processing of EA populations and potentially complex evaluation methods. The primary bottle neck involved in learning is that every candidate oscillator must be, under the current methods, evaluated for about 0.5 seconds of flight time. Increasing or decreasing that evaluation period severely degrades learning performance[11]. A more sophisticated evaluation process could attempt to take shorter samples of performance and employ predictive models to generate estimates of performance over longer segments of time.

It is expected that although MAV_MINIPOP was sufficient for us to develop a workable basis function set and demonstrate feasibility, it is unlikely to fly in the actual vehicle. Whatever does fly, however, would still have to function under rather severe resource and error feedback limitations. Further development of the ultimate learning engine will represent a significant challenge to be attacked over the next few years. This work has demonstrated the feasibility of in-flight controller tuning using an adaptive oscillator and has provided guidance on what issues should be studied in the pursuit of the next generation learning algorithm.

References

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

[2] Gallagher J, Doman D, Oppenheimer M. The technology of the gaps: An evolvable hardware synthesized oscillator for the control of a flapping-wing micro air vehicle. *IEEE Transactions on Evolutionary Computation*, in press.

[3] Gallagher J, Oppenheimer M. Cross-layer learning in an evolvable oscillator for in-flight controller adaptation in a flapping-wing micro air vehicle. In *Proc. the 45th Asillomar Conference on Signals, Systems, and Computers*, Nov. 2011, pp.1547-1551.

[4] Greenwood G, Tyrrell A. Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems. Wiley-IEEE Press, 2006.

[5] Goldberg D. Genetic Algorithms in Search, Optimization, and Machine Learning. Boston, MA, USA: Addison-Wesley, 1989.

[6] Fogel D. System Identification through Simulated Evolution: A Machine Learning Approach to Modeling. Ginn Press, 1991.

[7] Bäck T, Hammel U, Schwefel H. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1997, 1(1): 3-17.

[8] Wood R. The first takeoff of a biologically-inspired at-scale robotic insect. *IEEE Transactions on Robotics*, 2008, 24(2): 341-347.

[9] Doman D, Oppenheimer M, Sigthorsson D. Dynamics and control of a minimally actuated biomimetic vehicle: Part i—Aerodynamic model. In *Proc. the AIAA Guidance, Navigation, and Control Conference*, August 2009.

[10] Doman D, Oppenheimer M, Sigthorsson D. Dynamics and control of a minimally actuated biomimetic vehicle: Part ii—Control. In *Proc. the AIAA Guidance, Navigation, and Control Conference*, August 2009.

[11] Gallagher J, Doman D, Oppenheimer M. Practical in-flight altitude controller learning in a flapping-wing micro air vehicle. Technical Report 12-01, Department of Computer Science and Engineering, Wright State University, 2012.

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

[13] Kramer G, Gallagher J. Ananalysis of the search performance of a mini-population evolutionary algorithm for a robot locomotion control problem. In *Proc. the 2005 IEEE Congress on Evolutionary Computation*, Sept. 2005, pp.2768-2775.

[14] Vigraham S, Gallagher J. A space saving digital VLSI evolutionary engine for CTRNN-EH devices. In *Proc. the 2005 CEC*, Sept. 2005, pp.2483-2490.

[15] Kruskal W, Wallis A. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 1952, 47(260): 583-621.

[16] Mann H, Whitney D. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 1947, 18(1): 50-60.

[17] Schenato L, Wu W, Sastry S. Attitude control for a micromechanical flying insect via sensor output feedback. *IEEE Transactions on Robotics and Automation*, 2004, 20(1): 93-106.

[18] Deng X, Schenato L, Sastry S. Flapping flight for biomimetic robotic insects: Part ii — Flight control design. *IEEE Transactions of Robotics*, 2006, 22(4): 789-803.

[19] Epstein M, Waydo S, Fuller S, Dickson W, Straw A, Dickinson M, Murray R. Biologically inspired feedback design for drosophila flight. In *Proc. the 26th American Control Conference* (*ACC*), July 2007, pp.3395-3401.

[20] Augustsson P, Wolff K, Nordin P. Creation of a learning, flying robot by means of evolution. In *Proc. of the 2002 Conference on Genetic and Evolutionary Computation* (*GECCO2002*), July 2002, pp.1279-1285.

[21] van Breugel F, Lipson H. Evolving buildable flapping ornithropters. In *Proc. the 2005 Conference on Genetic and Evolutionary Computation*, June 2005.

[22] Hunt R, Hornby G, Lohn J. Toward evolved flight. In *Proc. the 2005 Conference on Genetic and Evolutionary Computation* (*GECCO2005*), June 2005, pp.957-964.

[23] Mouret J B, Doncieux S, Meyer J A. Incremental evolution of target-following neuro-controllers for flapping-wing animats. In *Lecture Notes in Computer Science 4095*, Nolfi S, Baldassarre G, Calabretta R, Hallam J, Marocco D, Meyer J A, Miglino O, Meyer J, Parisi D (eds.), Springer Berlin/Heidelberg, 2006, pp.606-618.

[24] Weng L, Cai W, Zhang M, Liao X, Song D. Neural-memory based control of micro air vehicles (MAVS) with flapping wings. In *Lecture Notes in Computer Science 4491*, Liu D, Fei S, Hou Z G, Zhang H, Sun C (eds.), Springer Berlin/Heidelberg, 2007, pp.70-80.

[25] Guo Q, Hu M, Wei R, Xu J, Song H. Hovering control based on fuzzy neural networks for biomimetic flying robotic. In *Proc. the IEEE Int. Conf. Information and Automation 2008* (*ICIA2008*), June 2008, pp.504-508.

[26] Boddhu S, Gallagher J. Evolved neuromorphic flight control for a flapping-wing mechanical insect. In *Proc. the 2008 IEEE Congress on Evolutionary Computation*, June 2008, pp.1744-1751.

John C. Gallagher is an associate professor of computer science and engineering at Wright State University in Dayton, USA. He holds a Ph.D. degree in computer engineering from Case Western Reserve University. He is a senior member of the IEEE and a recipient of an NSF Young Investigator's Award (CAREER). Dr. Gallagher's research interests are in evolvable and adaptive hardware, soft computation, and cyber-physical systems.

Michael W. Oppenheimer is a senior electronics engineer at the Control Design and Analysis Branch at the Air Force Research Laboratory, Wright Patterson Air Force Base, USA. He is the author or coauthor of more than 70 publications including refereed conference papers, journal articles, and technical reports. He holds a Ph. D. degree in electrical engineering from the Air Force Institute of Technology and is an Associate Fellow of AIAA and a senior member of IEEE. Dr. Oppenheimer's research interests are in the areas of reconfigurable flight control, control allocation, and control of flapping wing micro air vehicles.