Hierarchical Decomposition Considered Inconvenient: Self-Adaptation Across Abstraction Layers

John C. Gallagher*
Dept. of Computer Science and Engineering, Wright State Univ., 3640 Colonel Glenn Hwy., Dayton, OH USA 45435-0001

ABSTRACT

Hierarchy may be difficult, if not impossible, to avoid either in natural or artificially engineered systems. Nevertheless, hierarchical decomposition can be considered inconvenient in a number of ways that interfere with the construction of adaptive systems capable of full exploitation of unexpected opportunities in an operational environment. This paper will briefly review some of those inconveniences and suggest principles by which to circumvent them while maintaining the obviously beneficial, and perhaps inevitable, paradigm of hierarchical decomposition of engineering designs. The paper will illustrate the introduced principles by appeal to an example auto-adaptive flight controller for an insect-scale flapping-wing micro air vehicle. The paper will conclude with discussion of possible future applications and methodological extensions.

Keywords: Adaptive Systems, Micro Air Vehicles, Hierarchical Abstraction

1. INTRODUCTION

Functional Decomposition is the process by which one decomposes an artifact or process into self-contained, modular parts whose interactions give rise to the artifact or process originally considered. *Hierarchical Decomposition* can be considered to be a version of functional decomposition in which one applies the process of functional decomposition inside each module in a recursive manner until grounding out in some number basic descriptions below which further exploratory or design activity is deemed unnecessary. One can make arguments that hierarchy of some nature is inevitable in natural systems[1]. Determining whether or not those arguments of necessity apply to human engineered systems would be an interesting exercise. That being said, however, the core paradigms by which we currently engineer systems are explicitly hierarchical for reasons of economy, managerial control, maintenance, and fault containment. It is safe to say that casting off these paradigms is unlikely in the short term because doing so would incur massive disadvantage. These quantifiably practical and in some sense arguably inevitable hierarchies, however, do stand in opposition to the full exploitation of adaptive self-design. Unless one were to posit that engineered systems could arise from nothing and create and recreate themselves as needed (an interesting philosophical prospect in itself) – any reasonably complex artifact will possess some pre-existing hierarchy to which self-adaptation will be subject. One always runs the risk that any self-adaptation that is subject to the limitations of a specific modularization will be unable to properly exploit opportunities or repair system damage because no solutions exist within the confines of the hierarchical decomposition chosen at design time. The existence of this difficulty is at the very least inconvenient.

There are quite a few attacks possible on the above-mentioned inconvenience. One can simply attempt to apply soft-computation methods or other optimization techniques to assemble whole-cloth engineering artifacts from sets of generic base parts. Since the assembly of components would only be subject to measured system performance and not design hierarchies imposed by humans, one could suppose that hierarchies, if they arose at all as Simon predicts[1], could be reworked by the same optimization methods used to create them. Unfortunately, tabula-rasa approaches are unlikely to scale beyond toy problems. One can imagine using such methods to build a control circuit that maintains stability of an inverted pendulum. It is unlikely one would be so successful using similar means to create a car that passes efficiency, safety, and emission standards and is guaranteed to sell well in all markets with a large margin of profit for the manufacturer. Writing an objective function that describes the above, much less having the computational horsepower to evaluate it over many candidate designs assembled from thousands of complex parts, some with internal complexity

*john.gallagher@wright.edu; phone 1 937 775-3929; fax 1 937 775-4133; www.cs.wright.edu

Evolutionary and Bio-Inspired Computation: Theory and Applications VI, edited by Olga Mendoza-Schrock, Mateen M. Rizki, Proc. of SPIE Vol. 8402, 840202 · © 2012 SPIE · CCC code: 0277-786X/12/$18 · doi: 10.1117/12.922786

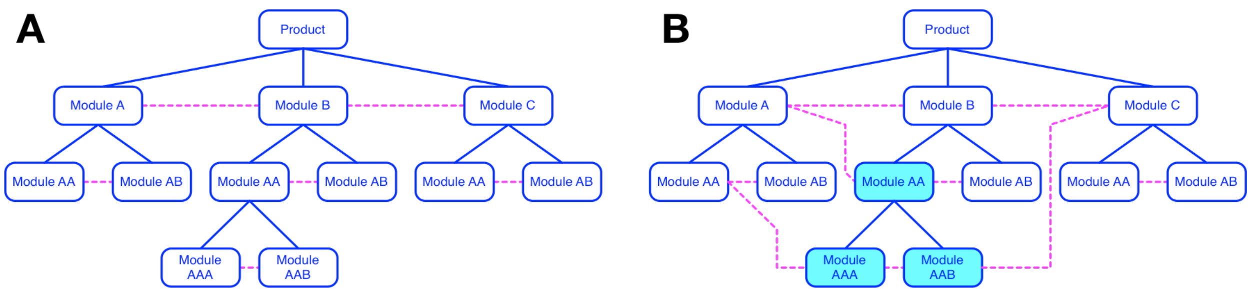

Figure 1. Design Hierarchy with and Without Plug-and-Learn Modules

rivaling that of the whole car, is unlikely. Alternatively, one could imagine a system of self-adaptation of designs that explicitly and simultaneously reworked design parameters at multiple levels of design abstraction simultaneously. In some sense, such a system would be able to melt and reform module definitions as needed. However, it would still be bound to work within pre-defined "layers" and subject to the same sorts of inconveniences as before – just at a slightly less annoying level. Both approaches are so obvious and foundational that there is no lack of either's representation in the literature. Any survey of the examples would as large and necessarily incomplete as a survey of examples of recursion used in computer science.

This paper will not propose either of the above extreme solutions. Rather, it will attempt an alternative approach that leverages, rather than fights, the nature of hierarchical decomposition to deliver enhanced adaptive capability that can be harnessed for effective system self-repair without over turning contemporary engineering methods. In this work, we will demonstrate how *plug-and-learn* components can exploit internal adaptation based on global performance metrics to provide benefit not by stepping outside a design hierarchy, but rather, by stepping into and through the cracks that the designers chose to push into the category of items not overtly represented in their decompositions. The paper will make appeal to an existing example of this method as applied to a controller for a flapping-wing micro air vehicle and will further discuss possible extensions to the idea and its applications.

2. PLUG-AND-LERAN: OPPORTUNITIES AND CHALLENGES

2.1 Plug-and-Learn vs. Plug-and-Play

Plug-and-Play components that could negotiate specific communication protocol settings with a host computer began to appear in the early 1980's. The idea was to decrease user frustration by allowing peripheral devices (modems, disk drives, etc.) to automatically negotiate interface protocols with host computers. Today we see that basic concept widely applied in a number of interface standards (USB, PCI, PCMCIA, etc.). *Plug-and-Play* represents a limited form of adaptability in which devices adapt their interfaces to one another in a highly prescribed manner. The salient observations are that adaptation occurs among peer modules that all exist at a single level of the design hierarchy; that it is the interfaces among modules that are adapted, and then only in a limited way; and that adaptation occurs only once, at the time of being "plugged in."

We define *plug-and-learn* components as devices that expand upon the concept of *plug-and-play* in the following ways: (1) upon attaching to the host system, they negotiate a performance metric defined at any level of design abstraction that is appropriately instrumented to report performance quality scores, and (2) during normal host system operation, they continuously modify their internal structures to improve the monitored performance metrics. *Plug-and-learn* represents a much broader form of component adaptability that in many ways paradigmatically subsumes any attempt to produce adaptive capabilities in an engineered device via encapsulation of adaptive capability into one or more modules of that device. The salient observations are that adaptation occurs in arbitrary groups of modules using information spanning multiple layers of the design hierarchy; that both the interfaces and the internal structures are adapted in perhaps profound ways; and that adaptation occurs on a continuous and decentralized basis over the operational life of the device containing the components. The *plug-and-learn* concept itself serves as a model of adaptive components existing in a hierarchy of structure.

2.2 How Plug-and-Learn Breaks Basic Hierarchical Analysis

Figure 1 represents the nature of design hierarchy and component interaction in both conventionally hierarchical designs and in designs containing *plug-and-learn* components. In Figure 1A, represents conventional design hierarchy. The product is broken into one or modules which are themselves further broken down until one reaches a layer of basic atomic components. The solid blue lines in Figure 1A represent design decisions about how to partition functionality into modules. Module decomposition designs are made for a combination of technical (e.g., component availability) and management reasons (e.g., corporate or design team structure). However a particular decomposition of a design comes to be, however, conventional design analysis assumes that intra-module communication and influence will mirror the structure of that decomposition. Communications occur among modules at the same level of hierarchical abstraction (dashed magenta lines) or to and from in-scope child or parent modules (solid blue lines). In such situations, it is possible to apply depth-first search and other hierarchical descent techniques to isolate faults, identify bottlenecks, and predict performance or reliability problems.

Figure 1B illustrates a more complex situation in which *plug-and-learn* modules (cyan shaded boxes) are present. By definition, *plug-and-learn* modules explicitly introduce cross hierarchy level communications and modify their internal structures to use or ignore, singly or in combination, any of those interactions to modify internal module structure to improve performances measured far outside of their own scope. A clean design hierarchy still exists, but that hierarchy is now only one of managerial utility, as the actual lines of module influence no longer mirror the lines of allocation of design responsibility. Hierarchical tools are no longer completely appropriate for isolating fault, identifying bottlenecks, or predicting future problems precisely because intra-module influence is no longer a matter of hierarchy. That some of the modules are adaptive and can manufacture lines of influence and communication while the device is in service makes matters even more complicated.

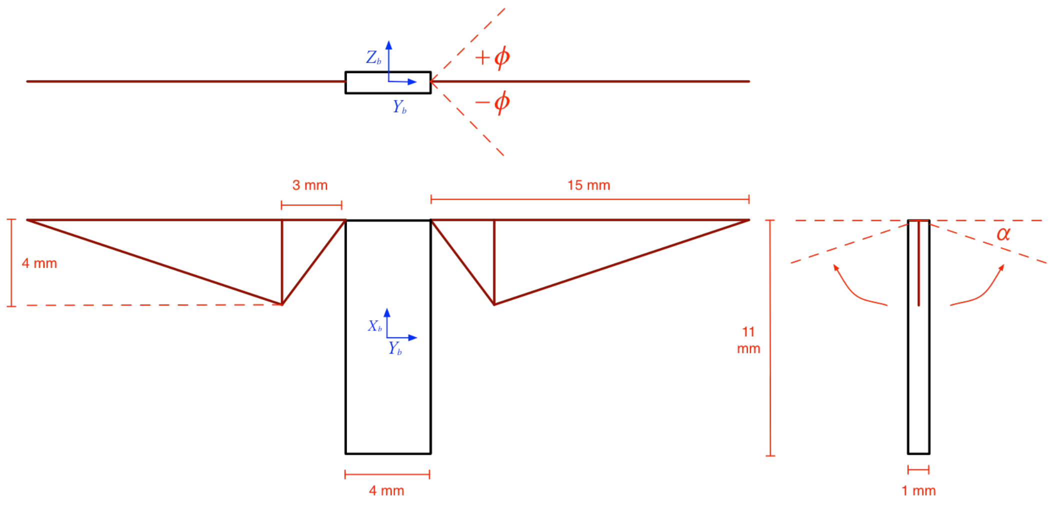

Figure 2. Orthographic View of Insect-Scale Flapping-Wing Vehicle

2.3 It Can Get Worse

In the context of a software-only system, the introduction of *plug-and-learn* components can introduce interactions in unexpected places and at arbitrary times. Those interactions are easily detectible, however, and one can map them by augmenting the original hierarchical decomposition to produce a graph structure. It is might be more complicated to isolate fault in an arbitrary graph, but at least the graph can be made visible and processed. *Plug-and-learn* modules combined with elecromechanical actuators can introduce lines of intra-module influence that are not easily represented by augmentation of any initial hierarchy. For example, consider a *plug-and-learn* oscillator inside a robot. Presume that the oscillator becomes damaged and, detecting a loss of performance at the whole system level, begins to reconfigure its internal structure to restore that whole-system performance measure. The oscillator autonomously discovers that by entraining to another oscillator elsewhere in the mechanism that performance can be restored. However, the adaptive module modifies itself to achieve entrainment via mechanical vibration. The *plug-and-learn* module completed its job, but in an unexpected manner that co-opted a physical substrate to communicate information via a channel not observable in or represented in the original design decomposition. Note that *plug-and-learn* modules adapt internal structure to increase a performance measure out of their local scope. There is no constraint to employ only particular interactions.

2.4 Plug-and-Learn Modules Break Hierarchy. Now What Do We Do?

Engineering practice will often require that adaptive capability be encapsulated in modules if for no other reason than to remain compatible with large-scale engineering practice. Project management concerns will still require hierarchy as a tool to assign design responsibility; but adaptation will quickly degrade the usefulness of that hierarchy for the purposes of practical design maintenance and fault detection. The basic problem exists in purely digital systems, but is much more difficult to deal with in systems where electronics are interfaced to sensors or effectors. The key question are, is such hierarchy breaking adaptation worth it, and can one recover some ability to analyze for potential faults after "hierarchy breaking" adaptation has occurred? These questions will be discussed in the context of a problem in flapping-wing vehicle control in which *plug-and-learn* components were used to enable the vehicle to recover flight precision after simulated in-flight damage.

3. PLUG-AND-LEARN IN AN INSECT-SCALE FLAPPING WING MICRO AIR VEHICLE

3.1 Vehicle Definition

We considered a variant of the Harvard RoboFly [1] with one piezoelectric actuator per wing instead of the one for both wings [2.3]. Figure 2 is an orthographic view of the vehicle. The basic wing motion can be best perceived in the top view (see the top view in the left top part of Figure 2, 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 (see the front view in lower left hand part of Figure 2 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 plan forms to an angle α under the plane of the spars (see the side view in the lower right hand part of Figure 2. Complete kinematics and dynamics of this modified two-effector vehicle can be found in previous works [2]. A related paper [4] shows 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. The specific wing frequencies and the scalar shape parameters specifying the form of the function for each wing can be computed from a simplified wing model and communicated to oscillator circuits that drive each wing. Unfortunately, subsequent simulation studies have demonstrated that even mild damage to the wings of the vehicle can cause any onboard controller to suffer significant losses in vehicle control accuracy and precision [5]. Due to the unpredictable nature of wing fabrication and in-service damage events, it is likely that on board controller adaptation will be necessary to maintain system efficacy in the field

3.2 Vehicle Simulation and Control

A MATLAB model for the vehicle has been developed by the Control Analysis and Design Branch of the Air Vehicle Directorate of the Air Force Research Laboratory (AFRL-RBCA). The author independently developed and verified against the MATLAB models altitude and roll mode simulation code in C that is appropriate for use on massively parallel computing clusters.

AFRL has also developed a simple Single Input Single Output (SISO) control structure for roll and altitude body translation with an allocation strategy that results in no axis cross-coupling [6]. Under this structure, each of two SISO controllers (one for roll and one for altitude) is based upon a cycle-averaged split-cycle concept. In split-cycle control, each wing is provided a wing beat 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 (Figure. 3) 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 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. Using blade element analysis, it is possible to relate wing beat cycle-averaged body forces and torques to wing frequency and shape parameter. Corrections to any aspect of body pose or position can be made by a

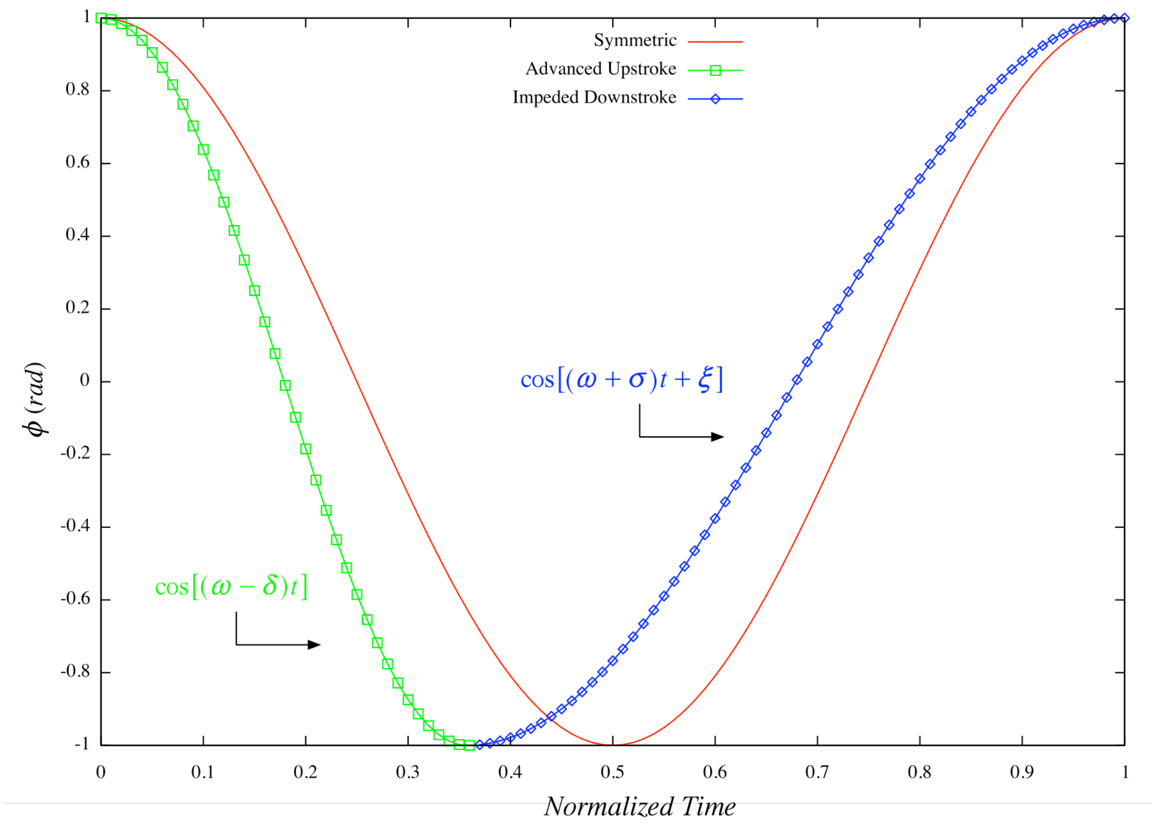

Figure 3. Split-Cycle Cosine

SISO control law that computes desired body force or torque followed by a mapping of the desired force or torque to wing shape parameters, which are given to the appropriate wing for the next wing beat. A collection of SISO controllers, plus an allocation strategy and a bank-to-turn waypoint tracker completes a six degree-of-freedom motion controller. The existing controller has been tested with the available aerodynamic model. Note that the controllers currently tested are based on idealized models; predictable, matched and balanced wings; and simplistic SISO controller allocation strategies. It is not clear as of yet that these are supportable assumptions, and in fact, simulation studies conducted by the proposing PIs suggest they are not. The relationship between this observation and the work proposed here will be discussed later in this document

3.3 An Evolvable Hardware Plug-and-Learn Component

Evolvable and Adaptive Hardware (EAH) [7] is an emerging subspecialty within Evolutionary Computation (EC) [8,9,10] in which one evolves designs for mechanical, computational, or electrical devices. EAH practitioners use search algorithms based on natural evolution to evolve mechanisms from a set of basic building blocks. The key difference between conventional automated controller tuning and EAH methodology for producing controllers is the amount of control the automated technique exercises over the final controller design. Conventional automated parameter tuning is constrained to tweaking the parameters of a human-designed device. 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. EAH methodology would make no assumption that the controller should be a linear proportional feedback controller. The final, EAH control device might operate like a linear proportional controller, but could just as easily have evolved to operate as a filter, as a linear quadratic regulator or even in ways that defy conventional description. It is this potential to step outside the box of convention that makes EAH attractive and in some sense, maximally adaptive. In this work, we will create an EAH *plug-and-learn* component that sits inside the predesigned SISO controller and modifies its internal structure to improve whole-system flight performance based on information it collects at a level of abstraction well above it.

The classical SISO controller relies heavily on the assumption that the wings produce predictable torques and forces on a cycle-averaged basis. Faults in wing manufacture and damage suffered in flight can cause these assumptions to be violated. However, in a vehicle of this size class, there is little room or time to do full system identification and attempt to rewrite control laws. Previously, we employed an adaptive EAH oscillator as a *plug-and-learn* component inside the design of several flight controllers for a simulated insect-scale Flapping Wing Micro Air Vehicle (FW-MAV) [5,6,11]. Each oscillator learned unique patterns of wing motion that restored the correct mappings between force and torque and wing shape parameter commands. These oscillators were designed to operate while the actual vehicle was in flight and use only local observations about the vehicle's deviation from its desired flight path to make wing motion table adjustments in the periods of time between when wing motion commands were issued once per wing beat. These oscillators have been tested in one degree of freedom flight (1-DOF) consisting of hover only and two degree of freedom flight (2-DOF) consisting of hover and roll (Figure 4). Over several million simulated test flights, we have found that randomly broken vehicles with up to 50% loss of wing lift and drag capacity can recover the same hover accuracy and precision of unbroken vehicles in an average of six minutes of vehicle flight time. With appropriate modifications to the basic

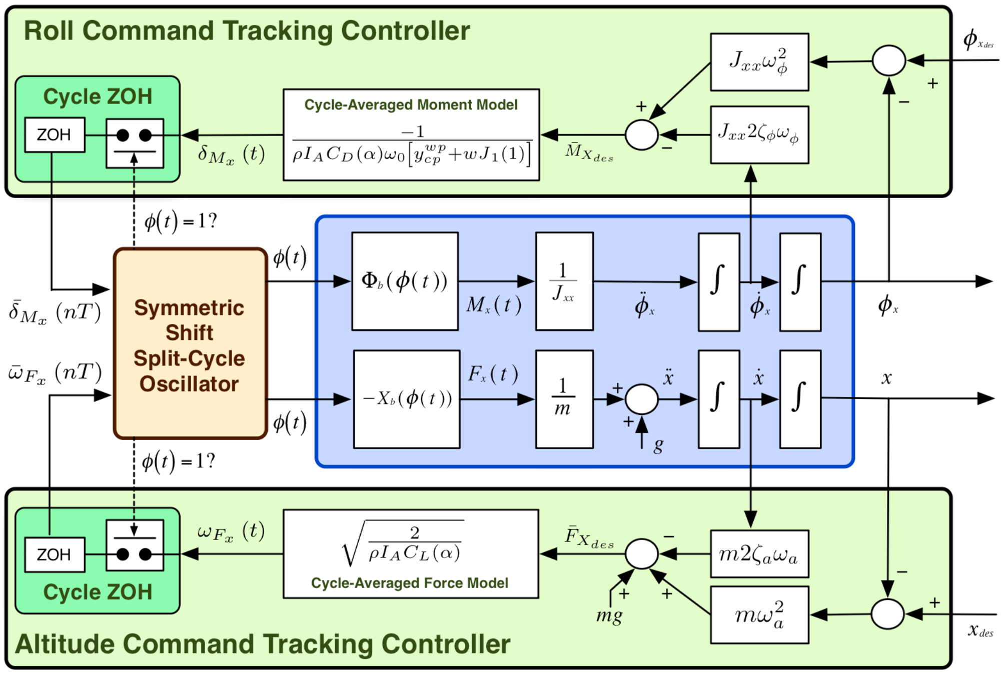

Figure 4. Roll and Altitude SISO Controller

oscillator circuitry [6], the same level of damage can be corrected in the 2-DOF case in, on average, sixteen minutes of vehicle flight time [6].

3.4 How Do the Oscillators Fix Flight Accuracy Problems?

Let's consider the case of altitude only correction. The most straightforward method of examining the extent to which a specific vehicle deviates from the assumed cycle- averaged model is via direct comparison of the generated forces and the controller predicted forces over the operational range of wing flap frequencies. The cycle-averaged upward force produced at a flapping frequency is:

$$F_X = \frac{\omega^2 \rho I_A C_L(\alpha)}{2} \tag{1}$$

where ω is the commanded flapping frequency of the wings, ρ is the air density, I_A is the wing area moment of inertia, $C_L(\alpha)$is the coefficient of lift for a wing attack angle of alpha, and F_X is the average upward force produced over the whole wing flap cycle. We will define the "force profile" of an oscillator to be the graph of F_X vs. ω over a range of possible wing flap frequencies. Figure 5A shows the force profile of several vehicles using the Altitude Command Tracking portion of the controller in Figure 4. The "perfect controller" is the profile of a controller using perfect oscillator with real valued time and output. The "true cosine" oscillators use 8-bit time increment intervals, but IEEE double precision cosine outputs for its split-cycle cosines. The "8-bit cosine" is identical to the "true cosine" case except that its cosine outputs are also quantized into 8-bit sized increments. The "failure A" and "failure B" profiles are vehicles using the "8-bit cosine" and with one of two different types of wing damage that reduce generated force and drag by 50% for one of the wings. Figure 5B shows the relative errors between each test case and the "perfect controller" for each oscillator/vehicle type. Note that the force profiles are close to ideal even under eight-bit quantization of time and output and that the profiles diverge significantly, as one would expect, with the damaged wings.

Each of the vehicles above could hover in a fixed position with the absolute errors shown in Table 1. Also shown are the mean squared errors (MSE) between the force profile of each simulated vehicle and the perfect profile as defined by equation (1).

Table 1. Average Absolute Error and Force Profile MSE for Various Oscillator and Vehicle Combinations

Experiment	**Absolute Error in Hover**	**Force Profile MSE**
perfect controller	+0.057 mm	4.115619e-18
reference oscillator with true cosine	-0.932 mm	1.708946e-14
reference oscillator with 8-bit cosine	+6.257 mm	2.338943e-09
reference oscillator with 8-bit cosine and wing failure A	-121.6 mm	3.717363e-07
reference oscillator with 8-bit cosine and wing failure B	-121.6 mm	3.717363e-07

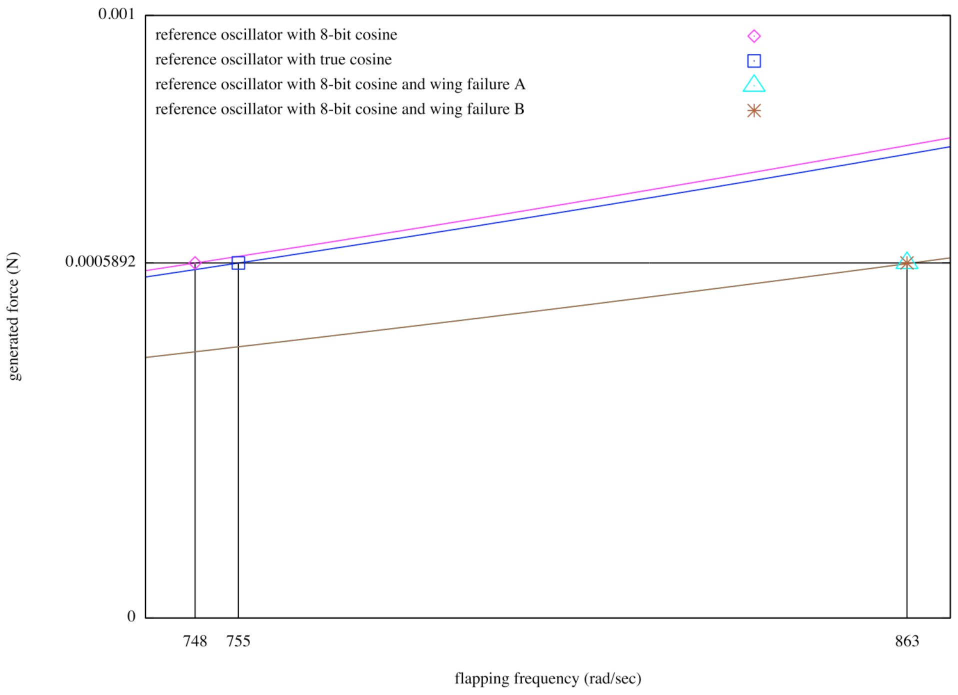

Figure 5. Magnified Force Profile Plot of Several Oscillator/Vehicle Combinations

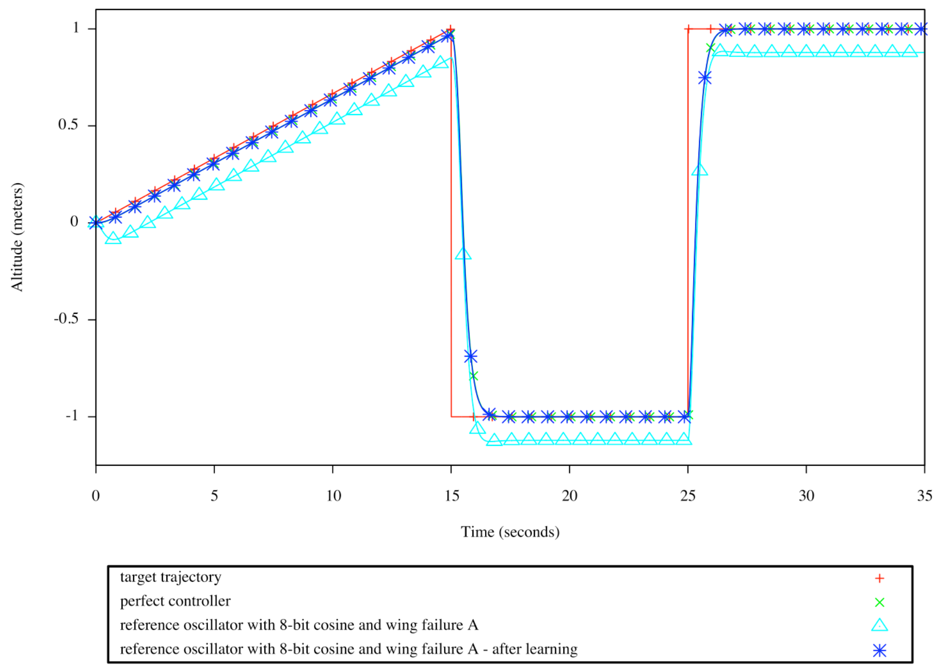

Figure 6. Flight Behavior of a Typical Failure Mode A Vehicle

The hover controller's failure to hover at the commanded altitude can be understood by examining the mismatch between forces commanded and forces generated. Figure 5 reproduces Figure 3A at higher magnification near the frequencies at which each reference oscillator controlled vehicle achieves hover. For the reference oscillator with true cosine 0.005892 N of force is produced at an ω of 755 radians/sec. Behaviorally, the hover controller in this vehicle commands an ω of 755 rad/sec for most wing beats, with an occasional command of 756 rad/sec to lift the slowly descending vehicle back to the desired hover point. The actual force profile of the damaged vehicles predicts that both wing hazard cases A and B should trim gravitational force at a frequency of 893 rad/sec. In simulation, both vehicle types exhibit exactly this flapping frequency at their false, erroneous hover points. Behaviorally, the wing failure cases do not generate enough upward force to track the desired altitude. When the altitude control becomes constant (a command to hover), the ACTC correctly commands ω values based on an incorrect vehicle model. In the currently discussed case, 893 rad/sec should cause the vehicle to rise. Because it approximately trims the vehicle, however, the vehicle does not rise above whatever false hover point it had achieved. The ACTC blindly enters into a behavioral deadlock condition in which it repeats approximately the same commands, but the desired whole-vehicle increase altitude action is not accomplished. The other absolute altitude errors can be via similar analyses.

Figure 6 shows the flight trajectory of a failure mode A vehicle controlled by a hover controller/adaptive oscillator combination before and after learning has completed. Figure 7 shows the before and after learning force profiles of the same vehicle. At hover, this vehicle can hold to within 0.02 mm of the commanded hover altitude and has a force profile MSE of $1.827417\ x\ 10^{-9}$. Although the flight performance and force profiles of Figure 6 and Figure 7 are typical of all learned oscillators, there is wide variation in the types of specific φ functions that produce more correct matching between vehicle behavior and the ACTC internal model. Qualitatively, one can see that the adaptive oscillator learns subtle modifications to wing motion to account for vehicle deficits at a level of abstraction below that at which cycle-averaged controllers function.

4. CONCLUSIONS AND FUTURE DIRECTIONS

Let's reconsider what was achieved and what are the implications. A model-based SISO controller was modified by removing normally non-adaptive components (the split-cycle oscillators) and replacing them with *plug-and-learn* versions. Those oscillators were able to modify their own patterns of oscillations to restore the frequency to force profile

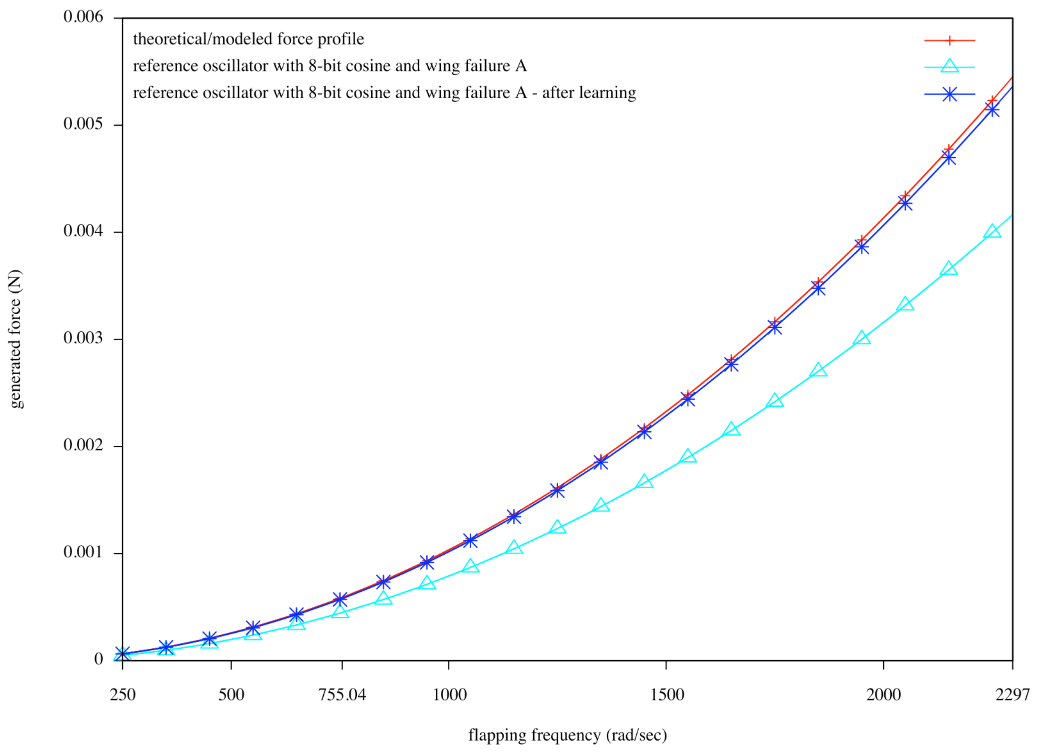

Figure 7. Force Profile of Typical Failure Mode A Vehicle After Learning

of vehicles with broken wings to that expected by the model-based SISO controller. That restoration was accomplished via adaptive action at a level of abstraction below that at which the controller was designed using information gathered from a higher level of the abstraction hierarchy. In this case, the oscillators learned to move the wings in a manner that would restore the force profile of the broken vehicle to a very good approximation of the force profile of the undamaged vehicle.

These experiments were conducted in simulation. There is, at this time, no way to know that the observed success could be repeated in an actual vehicle more susceptible to the possible problems outlined in section 2.2. Real vehicle studies are currently being designed, as are model-checking methods [12,13,14,15] for detecting the formation of new, non-hierarchical, interactions that might be introduced by *plug-and-learn* module adaptation. Although this work was presented in the context of modularized adaptation for control, the potential problems and the potential benefits are similar for any design effort that includes adaptive capability. Distributed sensor systems and/or sensor arrays that are exposed to potentially damaging conditions are both candidates for design methods like those proposed in this paper.

REFERENCES

[1] Wood, R.J., "The first takeoff of a biologically-inspired at-scale robotic insect", in *IEEE Transactions on Robotics*, 24 (11) 341-347 (2008).

[2] Doman, D.B., Oppenheimer, M.W., and Sigthorsson, D.O., "Dynamics and control of a minimally actuated biomimetic vehicle: part I -- aerodynamic model", In *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (2009).

[3] Doman, D.B., Oppenheimer, M.W., Bolender, M.A., and Sigthorsson, D.O., "Altitude control of a single degree of freedom flapping wing micro air vehicle", In *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (2009).

[4] Oppenheimer, M.W., Doman, D.B., and Sigthorsson, D.O., "Dynamics and control of a minimally actuated biomimetic vehicle: part II – control", In *Proceedings of the AIAA Guidance, Navigation, and Control Conference* (2009).

[5] Gallagher, J.C., Doman, D.B., and Oppenheimer, M.W., "Practical In-Flight Altitude Controller Learning in a Flapping-Wing Micro Air Vehicle", *IEEE Transactions on Systems* (submitted).

[6] Gallagher, J.C. and Oppenheimer, M.W., "An Improved Evolvable Oscillator for All Flight Mode Control of an Insect-Scale Flapping-Wing Micro Air Vehicle", In *Proceedings of the 2011 IEEE Congress on Evolutionary Computation* (2011).

[7] Greenwood, G and Tyrrell, A., [*Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems*], (2005).

[8] Goldberg, D.E. [*Genetic Algorithms in Search, Optimization, and Machine Learning*] Addison-Wesley, (1989).

[9] Fogel, D.B. [*System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press] (1991).

[10] Back, T., Harnmel, U., and Schwefel, H.P., "Evolutionary Computation: Comments on the History and Current State", in *IEEE Transactions on Evolutionary Computation*, 1(1), 3-17 (1997).

[11] Gallagher, J.C. and Oppenheimer, M.W., "Cross-layer learning in an evolvable oscillator for in-flight control adaptation of a flapping-wing micro air vehicle", In *45th Asilomar Conference on Signals, Systems, and Computers,* (2011).

[12] Emerson, A. and Clarke, E., "Characterizing correctness properties of parallel programs using fixpoints", in *Automata, Languages and Programming*. Springer, (1980).

[13] Clarke, E. and Emerson, E., "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic", In *Logic of Programs, Workshop*, (1981).

[14] Clarke, E.M. and Emerson, E.A., Sistla, A.P., "Automatic verification of finite-state concurrent systems using temporal logic specifications", in *ACM Transactions on Programming Languages and Systems*, 8(2) 244-263, (1986)

[15] Queille, J. and Sifakis, J., "Specification and verification of concurrent systems in CESAR", In *Lecture Notes in Computer Science - International Symposium on Programming*. (1982).