An Evolutionary Approach to Discovering Execution Mode Boundaries for Adaptive Controllers

Anthony J. Clark*, Byron DeVries†, Jared M. Moore‡, Betty H. C. Cheng†, and Philip K. McKinley†
*Computer Science Department, Missouri State University, Springfield, MO, USA
†Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA
‡School of Computing and Information Systems, Grand Valley State University, Allendale, MI, USA
AnthonyClark@MissouriState.edu

***Abstract*—Adaptive controllers enable cyber-physical systems, such as autonomous robots, to manage uncertain conditions during execution. However, there is a limit to the range of conditions that can be handled by a given controller. When this limit is exceeded, a controller might fail to respond as expected, not only rendering it ineffective but possibly putting the entire system at risk. In this paper, we describe a method based on evolutionary search for automatically enhancing, and discovering the boundaries of, a given adaptive controller. Collectively, these boundaries define an *execution mode* for that controller. Explicit specification of mode boundaries facilitates the development of decision logic that determines, based on system state and sensed conditions, when to switch to a different execution mode and typically a different controller, such as one for providing fail-safe operation. To evaluate the proposed approach, we apply it to a robotic fish propelled by a flexible caudal fin that is governed by a model-free adaptive controller. Experimental results demonstrate that this approach is effective in characterizing a controller's ability to adapt to environmental dynamics, including physical damage to the robot itself.**

I. Introduction

Cyber-physical systems integrate computational elements with physical processes, and therefore involve combinations of discrete and continuous dynamics. Many such systems use adaptive controllers to help manage uncertain conditions during execution [1]. In the control theory domain, adaptation refers to a controller's ability to automatically adjust control parameters during operation in response to sensed feedback. A controller's environment includes any variables *not directly controlled by the system.* For example, from the perspective of a robot's control software, the environment includes not only the physical surroundings in which the robot operates, but also characteristics of the body (morphology) of the robot. As with the physical environment, the morphology is subject to change: the material properties and behavior of physical components can vary due to wear, changes in temperature, and physical damage. We use the term *scenario* to describe a specific set of environmental parameters and their respective values. Compared to a static (i.e., non-adaptive) controller, the ability to adapt ensures that a single controller will remain effective for many scenarios. An important task for the cyber-physical system designer, then, is to specify the limits of adaptation for the controller. That is, by how much can any parameter change before a controller fails and the system needs to switch to a different mode of execution driven by another controller?

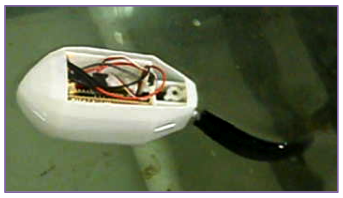
In this paper, we explore the role of evolutionary computation (EC) in both enhancing and discovering the limits of adaptive controllers. We employ a particular evolutionary algorithm—differential evolution [2]—and combine it with a dynamic simulation in order to evolve adaptive controller parameters. In the proposed method, over the course of evolution we expose controllers to different scenarios not only to enhance the adaptive capabilities of the controller, but also to identify conditions under which the controller will fail. This information can be used to define the boundaries of an *execution mode*, enabling the system designer to implement higher-order strategies for switching among execution modes, when the system detects that a change is necessary. The target platform for this study is a 3D-printed robotic fish, shown in Figure 1(a), propelled by a flexible caudal (tail) fin. The oscillating frequency of the fin is governed by a *model-free* adaptive controller (MFAC) [3], which "learns" how to control the system by continually updating link weights in an artificial neural network (ANN); additional details are provided in Sections II and III. Such a controller needs to adapt to dynamics in the external environment (e.g., water currents) as well as changes to the robot morphology.

The main contribution of this study is an approach that addresses uncertainty for autonomous robots at two levels. First, we evolve adaptive control to handle increasingly more adverse/diverse environmental conditions. And second, we automate the process of discovering the boundaries of adaptation.

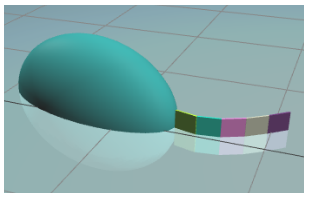
II. Background and Related Work

We begin by briefly reviewing areas of study that provide a foundation for this work: adaptive control, evolutionary algorithms, and their combination.

Adaptive Control. Control theory focuses on how to modify the behavior of dynamic systems using feedback. The control of a cyber-physical system typically involves



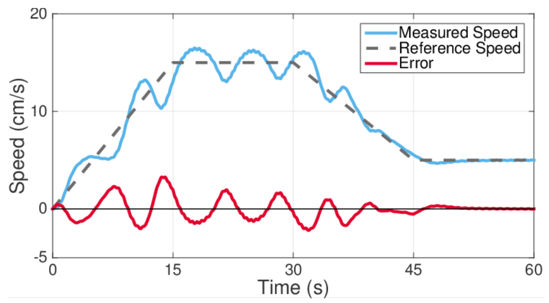
(a)



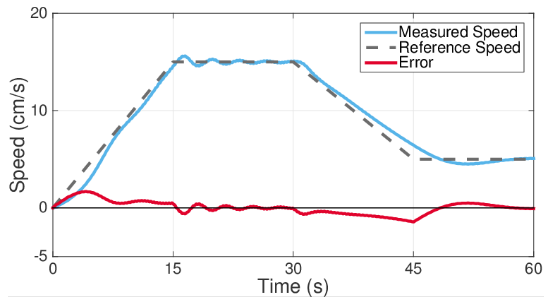
(b)

Fig. 1. (a) A prototype 3D-printed robotic fish with a flexible caudal fin (top cover removed for illustration). (b) Rendering of the simulated robotic fish; the fin appears to extend above the surface of water for visualization purposes only.

monitoring the output of a system (e.g., speed, temperature, flow-rate, etc.) and adjusting the system input accordingly. Often the goal of a control system can be referred to as *tracking*. The objective for a tracking controller is to generate a signal that drives the controlled system to behave in a similar manner to an input *reference signal*. Alternatively stated, the goal of a tracking controller is to minimize the error between a reference signal and the actual system output. Two examples of controlling the speed of a robotic fish are shown in Figure 2. The system input reference signal (the black, dashed line) and the system output (blue line) are the desired and measured speeds, respectively, of the robot. The error between these two values, which acts as an input to the controller, is shown in red. In Figure 2(a), the controller exhibits poor tracking, that is, the system output does not closely track the reference signal (notice that the output speed oscillates around the desired speed). In contrast, Figure 2(b) shows a controller that is effectively tracking.



(a)



(b)

Fig. 2. Examples of a robotic fish controller tracking an input reference signal representing desired speed. (a) is an example of relatively poor tracking, whereas (b) demonstrates effective tracking.

Figure 3 illustrates the difference between a conventional feedback controller and an adaptive controller. Both use the error e between the reference signal r and measured output y in order to produce an input u to the target system (or *plant*). While a non-adaptive controller enables the system to respond to dynamic conditions by adjusting the value of u, the manner in which it does so is *fixed*. Specifically, once tuned for a particular system and expected set of conditions, the parameters to the equations defining u are constants. An example is the widely used PID controller, where u is the weighted sum of terms involving e (**P**roportional term), its integral (**I**ntegral term), and its derivative (**D**erivative term).

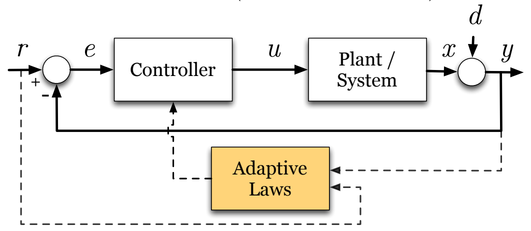


Fig. 3. A block diagram for a feedback control system. The shaded block shows the additional components necessary for adaptive control. The signals r, y, and e denote the reference signal, system output, and system error, respectively, while u and d are the control signal and disturbances, respectively. All of these signals vary as a function of time.

In contrast, an adaptive controller automatically adjusts control parameters online according to adaptive laws (i.e., rules typically derived using Lyapunov stability that dictate how control parameters will be updated). One promising approach is called *model-free* adaptive control (MFAC), which is also referred to as data-driven control [3], [4]. Like other adaptive control approaches, an MFAC attempts to minimize the error between desired and actual outcomes. However, instead of updating process-specific control parameters, an MFAC controls the system by updating link weights in an ANN. MFACs are intended for "gray box" situations, where only partial and possibly inaccurate knowledge of the physical process is available. However, one difficulty in using model-free control is that several parameters of the MFAC, discussed later, must be specified at design time. The approach explored here is to *evolve* these MFAC parameters by integrating physics-based simulation into evolutionary algorithms.

Evolving Adaptability. In studying animal behavior, zoologists recognize the importance of the *evolved* relationships among morphology, perception, action, and environment. This concept is also relevant to cyber-physical systems, where embedded computer systems need to interpret sensed information and respond accordingly through actuators. Evolutionary computation (EC) methods codify the basic principles of genetic evolution in computer software and are particularly effective in addressing problems involving large, multidimensional search spaces [5].

In the realm of cyber-physical systems, EC has been applied extensively in robotics. In evolutionary robotics [6], a genome encodes a robot's control system and possibly aspects of its morphology. Researchers have investigated the open-ended evolution of both individual and collective behaviors [6], [7]. Other studies have investigated how EC can be used to *enhance* more traditional engineering methods. For example, Coelho et al. [8] used evolutionary methods to improve the performance of an adaptive controller by evolving a neural compensator. For this work, we applied differential evolution (DE) [2], a global optimization algorithm that operates in a similar manner to other evolutionary algorithms.

In an earlier study [9], we used DE to evolve the parameters of an MFAC in order to increase the controller's adaptability. Specifically, those experiments demonstrated that exposing the MFAC to different scenarios during evolution produced a more resilient controller, even capable of adapting to conditions beyond those to which the system was exposed during evolution. However, the focus of that work was only adaptability, and the scenarios were designed by hand. In addition, for some scenarios (such as extensive damage to the fin), the controller was simply unable to effectively adapt. In the study reported here, we are interested not only in enhancing adaptive controllers, but also in discovering the boundaries of adaptive behavior with respect to the environmental conditions. Moreover, we seek automated ways to select scenarios that, when integrated with an evolutionary algorithm, most effectively identify and characterize those boundaries.

III. Robotic Fish Platform

Robotic fish are a type of aquatic robot that swim by deforming fins or their entire body. In addition to enabling high maneuverability, this form of locomotion produces less noise and water disturbance than propeller-driven locomotion. Therefore, robotic fish are of particular interest for applications such as environmental monitoring [10].

Fabricated Robot. The robotic fish shown in Figure 1(a) was produced with the aid of a high-resolution Connex 350 3D printer. This system can print in multiple combinations of materials (from rigid to very soft plastic), enabling fabrication of complex robot components. An example is a fin that is relatively rigid at the base and increases in flexibility toward the posterior. The 3D-printed components include the main body, the caudal fin, and other mechanical parts such as gears and battery/motor housings. In this prototype, the caudal fin is detachable, facilitating evaluation of performance using different fins. Flexible fins have been shown to produce more forward thrust than rigid fins, but the optimal flexibility depends on multiple factors [9]

Controller. Figure 4 shows a diagram of the MFAC and its configuration in the robotic fish platform. The input to the system is a reference signal r, which can be any physical signal relating to the robotic fish. For this study, r refers to a *desired* speed, and the output of the robotic fish y is the *actual* (measured) speed. Generally, reference signals are generated by a higher-level software module directing the robot to adopt a particular speed and heading as part of a larger task (e.g., following a list of way-points and taking environmental samples). The controller's objective is to produce a control signal u (in this case, a servo motor control pattern) such that y closely tracks r. To do so, the MFAC must adapt to changes in fin dynamics due to, for example, water temperature, material decay, and accumulation of residue. The shape of the fin itself can change due to physical damage. Indeed, from the perspective of the controller, the fin can even appear to be "lengthened" if it becomes entangled with an object, such as plant material. The robot controller does not "know" what has happened to the fin; it simply experiences a different response to the control signal.

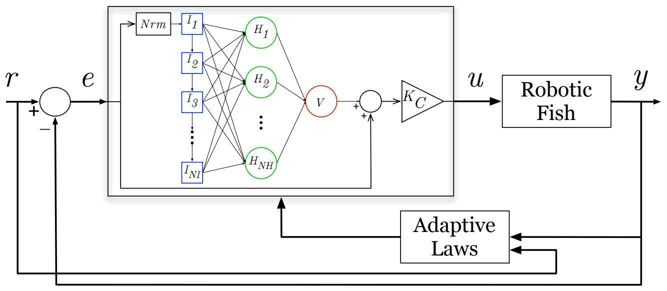
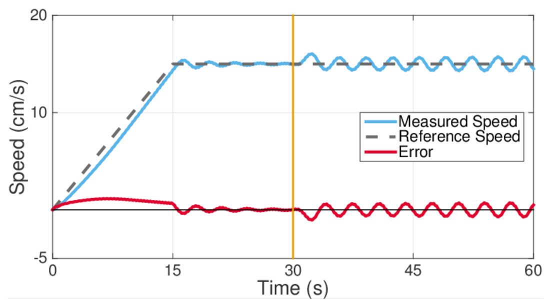


Fig. 4. A block diagram of the MFAC controller and the robotic fish. Signals r and y denote the reference and measured speeds, respectively, e is the difference between reference and measured speeds, and u is the sinusoidal controller output.

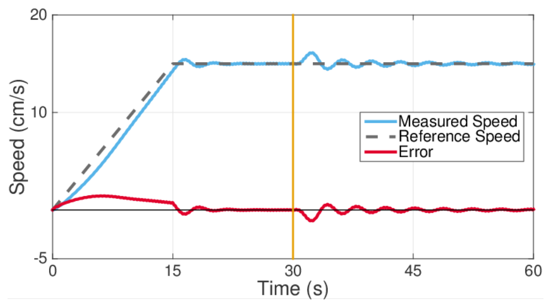
An MFAC relearns how to control the system by continually updating link weights in an adaptive ANN. As an input to the neural network, the MFAC takes a continuous error signal and discretizes it based on a configured sampling rate. By saving recent error signals and using them as additional inputs to the ANN (labeled I_1 to I_{NI} in Figure 4), the MFAC can take advantage of state information, or so-called neural network memory [11].

Figure 5 shows the difference between a lack of adaptive ability and a controller that is able to adapt. In both figures the fin suffers damage at the 30 second point (denoted by the vertical orange line); specifically, the fin is shortened from 8 to 5.2 cm and the flexibility is changed from 3.0 to 2.1 GPa (flexibility is described later in this section). Figure 5(a) shows the resulting behavior when the system cannot handle the change; the system starts oscillating around the target speed, but does not achieve good tracking. Figure 5(b) shows an adaptive controller that is able to regain tracking after damage.

Although an MFAC adapts by updating ANN link weights during execution, other parameters of the MFAC determine its *ability* to adapt. Table I lists the nine such parameters targeted in this study. Collectively, these parameters define the responsiveness and sensitivity of the controller, the structure and processing capabilities of the ANN, and the update periods for the ANN weights and



(a)



(b)

Fig. 5. In (a) the robotic fish is controlled by a non-adaptive controller, and in (b) the robot is controlled by an adaptive controller. In both figures, the fin suffers damage at t=30 s, and only the adaptive controller is able to successfully regain the ability to track.

the controller output. These parameters are configurable at design time, but typically do not change after deployment. Determining their optimal values is challenging and depends on the application domain. Traditionally, MFAC parameters are either chosen based on expert knowledge or tuned using proprietary software specific to the application [12]. Here we *evolve* these parameters.

TABLE I
Evolvable MFAC Parameters

K_c	controller gain to amplify/reduce output
N_I	number of ANN input nodes
N_H	number of ANN hidden nodes
E_B	value used to normalize MFAC error inputs
η	learning rate for ANN edge weights
T_{out}	update period for generating MFAC output
T_{wt}	update period for MFAC weights
α	sensitivity parameter, reactivity of MFAC
β	sensitivity parameter numerator

Simulation Model. To enable evaluation of MFAC performance within an evolutionary algorithm, we constructed a model of the robotic fish in Simulink [13], a graphics-based environment for modeling and simulating dynamical systems. An important part of the simulation is modeling of the dynamics of the flexible caudal fin. Here, we adopt an approach developed by Wang et al. [14], which represents the flexible fin as multiple rigid segments connected by springs and dampers. Figure 1(b) depicts the simulated robotic fish, where the caudal fin is modeled as 5 segments. The spring constant between two consecutive segments determines how stiff or flexible the caudal fin behaves. The forces acting on each segment are summed to produce the resulting thrust applied to the body. This model has been demonstrated to be both accurate, in capturing the dynamics of flexible materials in water, as well as computationally efficient [14].

Evolving a Base Morphology. Under this model, the morphology of the caudal fin is described by three parameters: flexibility, depth (height), and length. Flexibility is represented by a Young's modulus (YM) value, measured in Pascals (typically GPa). Rubber-like materials have YM values in the range of 0.1 to 1.0 GPa, and hard plastics have values in the range of 1.0 to 5.0 GPa. We begin by evolving the three fin parameters and a sinusoidal signal, which acts as the "controller," for a simple task (maximum average speed). The adaptive controllers evolved in the next sections will be required to respond to changes in this "base" morphology. We conducted 30 replicate DE experiments. The robotic fish achieved a maximum average speed of 22 cm/s, with a length, depth, and flexibility of 8.0 cm, 2.6 cm, and 3.0 GPa, respectively. In a prior study [15], we validated the results of this process by 3D-printing fins and testing them on physical robots.

IV. Methods

The primary goal of this study is to apply evolutionary search in order to discover and explicitly define the boundaries of adaptability for an MFAC that controls a robotic fish. Notably, this approach should also produce an MFAC that can operate effectively within those boundaries. Our basic approach is to expose the MFAC to multiple scenarios during the evolutionary process. Here, we focus only on changes to morphology, which typically occur due to uncertainty and unpredictable circumstances such as damage or when the device becomes entangled. In this section, we define precisely what constitutes a scenario and how we determine whether a scenario is feasible, followed by a description of the proposed algorithm.

Scenario Parameters. Each scenario includes the three morphological parameters discussed earlier: fin length, depth, and flexibility. Because the MFAC will be required to track a variety of reference signals (i.e., desired behaviors), each scenario also includes a reference signal whose parameters are generated randomly. For this study, reference signals describe only dynamics associated with the speed and acceleration of the robot, but the same approach could address other more complex behaviors and maneuvers. As depicted in Figure 6, each reference signal comprises an interval of acceleration from zero cm/s up to a constant speed S_1, followed by either another acceleration or a deceleration to a second constant speed S_2. The durations of the four intervals are defined by parameters t_1, t_2 and t_3. This approach enables generation of reference signals that contain a rich set of dynamics (i.e., different maximum and minimum speeds and accelerations/decelerations).

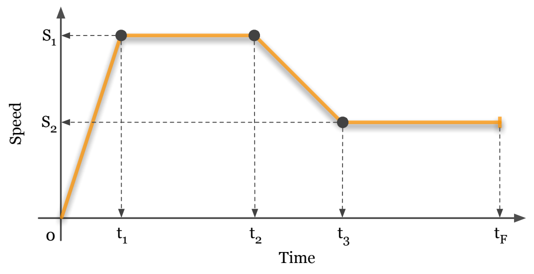


Fig. 6. Reference signal scenario parameters include values to describe four time segments. In the first segment (from t=0 to t=t_1) speed ramps from 0 to S_1, in the second segment the reference speed remains steady at S_1. In the third segment (from t=t_2 to t=t_3) the speed ramps (up or down) to the final speed, which is held steady during the final time segment.

Determining Scenario Feasibility. Some combinations of morphological parameters may produce a robot that simply cannot be controlled effectively. For example, if the fin is severely damaged or too flexible, then it might be impossible to generate sufficient thrust to reach a specified reference speed, much less maintain it. Such a scenario is deemed infeasible, or *invalid*. We implemented an automated procedure for identifying such scenarios, so that they can be excluded from the resulting execution mode (as well as from the evolutionary search process). The feasibility procedure also considers the fact that the MFAC used in this study cannot be applied to systems that switch between direct and reverse acting. Direct (or reverse) acting signifies that the system output, speed in this case, will always increase (or decrease) as the system input, frequency, increases (see Figure 7). For example, for very flexible fins, speed can increase at low frequencies, but start to decrease at higher frequencies [15].

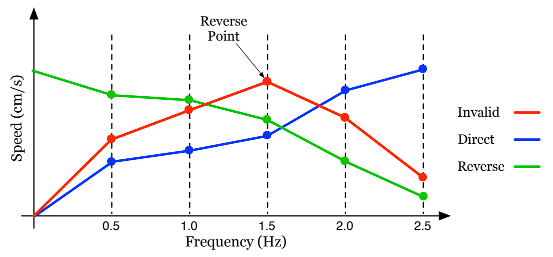


Fig. 7. Three behaviors are shown. The blue and green lines denote systems that are direct and reverse acting, respectively. The red line shows a system that switches acting modes and is deemed infeasible.

Before considering a randomly generated scenario for integration in the evolutionary algorithm, it is first tested for feasibility as follows. A robot with the specified fin characteristics is simulated for 15 seconds with fin frequencies of 0.5, 1.5, and 2.5 Hertz. For each frequency, the robot is first allowed to reach a steady-state speed, and then an average speed is sampled at 10 Hz over the final 5 seconds. If the results show that the behavior changes from direct to reverse acting, or if the robot fails to reach a speed of 15 cm/s, then the scenario is considered to be infeasible. We chose 15 cm/s because the reference signal values are allowed to include a maximum speed of 20 cm/s, which is just below the maximum speed of the base morphology. We note that no evolution (or feedback control) is involved in these evaluations; rather, they are intended only to test robot behavior at increasing frequencies.

Basic Algorithm. Figure 8 shows a flow chart of the algorithm developed for this study, which we refer to as the *Mode Discovery Algorithm*. As noted in Section III, we begin by evolving the base morphology. The base fin parameters are combined with a hand-designed reference signal to produce the *base scenario*, which is placed in the set of scenarios S applied in evolution. (The base reference signal starts at 0, ramps up to 15 cm/s, and then ramps down to 5 cm/s.) The algorithm then alternates between evolving the MFAC against scenarios in S for a fixed number of generations (by default, 10) and generating a new scenario to add to S. Here fitness is based on how closely the robot tracks the specified reference signal. Specifically, we evaluate the robot/MFAC for each scenario in S taking the mean-absolute-value of the sampled error signal. Smaller average error translate to higher fitness. The number of iterations through this basic loop is configurable (also 10 by default). The population size for the DE algorithm is 90. Evolving initially against only the base scenario is intended to bootstrap MFAC evolution, enabling the optimization process to start with an easier objective before adding a pressure for adaptability. We implemented and evaluated two methods for generating and selecting scenarios, termed *boundary selection* and *volume selection*. These methods, along with the corresponding results, are described in the next section.

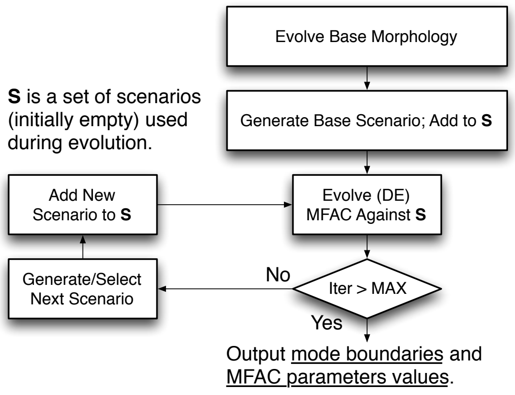

Evolve Base Morphology
S is a set of scenarios (initially empty) used during evolution.
Generate Base Scenario; Add to S
Add New Scenario to S
Evolve (DE) MFAC Against S
Generate/Select Next Scenario
No
Iter > MAX
Yes
Output mode boundaries and MFAC parameters values.

Fig. 8. Flowchart of the *Mode Discovery Algorithm* used to discover execution mode boundaries and produce a MFAC parameter values.

V. Results and Discussion

In this section, we describe results for the two scenario selection methods, and for a parameter sweep experiment that provides a "ground truth" for the execution mode. Both approaches aim to enhance adaptability while at the same time providing information regarding execution mode boundaries.

Boundary Scenario Selection. The method starts by identifying the limits of each morphological parameter (i.e., fin length, depth, and flexibility), using the feasibility checking procedure described above. Specifically, we start at the base value for each parameter and increase/decease the value until the simulated system becomes infeasible. During this process, we consider one parameter at a time, while the other parameters are fixed at their base values. The increment/decrement values were 1 mm for length and depth and 0.5 MPa for flexibility. Boundaries values (maximum and minimum for each parameter) are denoted in Table II. We note that the maximum value for flexibility is equal to the base value of 3.0 GPa, which is the maximum value we can 3D-print [15].

TABLE II
BOUNDARY SCENARIOS FOR FIN PARAMETERS
(* INDICATES BOUNDARY VALUE)

	Length	**Depth**	**YM**
$scenario_{Base}$	8.0 cm	2.6 cm	3.0 GPa*
$scenario_{length_{min}}$	6.0 cm*	2.0 cm	3.0 GPa
$scenario_{length_{max}}$	8.4 cm*	2.6 cm	3.0 GPa
$scenario_{depth_{min}}$	8.0 cm	1.0 cm*	3.0 GPa
$scenario_{depth_{max}}$	8.0 cm	2.7 cm*	3.0 GPa
$scenario_{YM_{min}}$	8.0 cm	2.6 cm	2.5 GPa*

Collectively, these feasibility tests produce the six boundary scenarios listed in Table II. These are integrated into the Mode Discovery Algorithm as follows. During the first 10 generations, the MFAC evolves using only the base scenario in fitness evaluation. At the start of each subsequent iteration, one of the remaining scenarios is randomly selected and added to set S. Adding one scenario at a time gradually increases the difficulty of the task. At the start of the fifth iteration (and the beginning of the 60th generation overall) all six scenarios are in use. Scenario insertion orderings will differ across replicate experiments, however, examination of the replicates shows that order has little if any effect.

Consistent with our earlier study [9], adaptive controllers resulting from this process exhibit enhanced adaptability when compared to those evolved against only the base scenario (results not shown due to space limitations). However, this approach has a drawback: it does not consider the interactions among morphological parameters. In earlier experiments [9], we observed combinations of parameters that were feasible, despite lying outside the feasible regions defined by the boundary values in Table II. For example, when a caudal fin is shortened it will generally continue to work well if it is sufficiently flexible. For instance, when we test the feasibility of a fin with a length of 6.4 cm and a Young's modulus of 2.1 GPa (below the discovered 2.5 GPa threshold) we find that it is, in fact, feasible.

These observations led us to explore the execution mode boundaries in two additional ways. The first is a brute force approach that tests for feasibility as described above, considering all combinations of the three fin parameters simultaneously. This method enables us to find the "ground truth" for the execution mode. While applicable to this relatively small problem, we emphasize that such an approach would likely be infeasible on a more complex system due to computational requirements. The second approach is to determine if we can find similar boundaries using the Mode Discovery Algorithm. Each approach is discussed in turn.

Simultaneous Parameter Sweep. We performed a parameter sweep across the three morphological parameters, using the same granularity as before. Results for these simulations, 62,068 in total, are shown in Figure 9. Each blue dot represents a feasible scenario (and which theoretically an MFAC can handle), while each gray dot represents infeasible scenario. Collectively, the regions of blue dots define a possible execution mode for an adaptive controller. The red boxes represent the areas found by the boundary selection method, and the diagonal line in Figure 9 is due to a constraint in the simulation model of the flexible fin, which requires that the length must be at least three times the depth [14]. As shown in Figure 9, these parameter sweeps reveal much larger and more complex feasible regions than the boundary experiments.

Volume-Based Scenario Selection. Given these results, we developed a second scenario selection method that takes into account interactions among parameters. Since it is woven into the evolutionary process, it produces both the execution mode boundaries (at lower cost than sweeps) as well as an adaptive controller for that mode. Table III describes the range from which scenario parameter values are randomly selected. Unlike the boundary selection method, where the reference signal was always the same base pattern, here we randomize the reference signal values for each scenario, which adds an additional challenge for the adaptive controllers (i.e., they must adapt to different control requirements in addition to different caudal fin dynamics). Each evaluation takes 60s of simulation time. The time values (t_1, t_2, and t_3) shown in Figure 6 were produced by generating three random numbers such that $t_1 = t_{rand_1}$, $t_2 = t_1 + t_{rand_2}$, and $t_3 = t_2 + t_{rand_3}$. Each of the first three time segments is in the range [5, 25] s, and if t_3 is less than 60 seconds then the final segment includes all time remaining.

TABLE III
VOLUME SCENARIO LIMITS

	Minimum	**Maximum**	**Base**
Fin Length	2.0 cm	20.0 cm	8.0 cm
Fin Depth	0.5 cm	4.0 cm	2.6 cm
Fin YM	0.1 GPa	3.0 GPa	3.0 GPa
S_i	0.0 cm/s	20.0 cm/s	15.0 cm/s
t_{rand_i}	5.0 s	25.0 s	15.0 s

As with the boundary approach, this method starts with the base scenario and adds one new scenario at the beginning of each iteration of the Mode Discovery Algorithm. However, since randomly generated scenarios

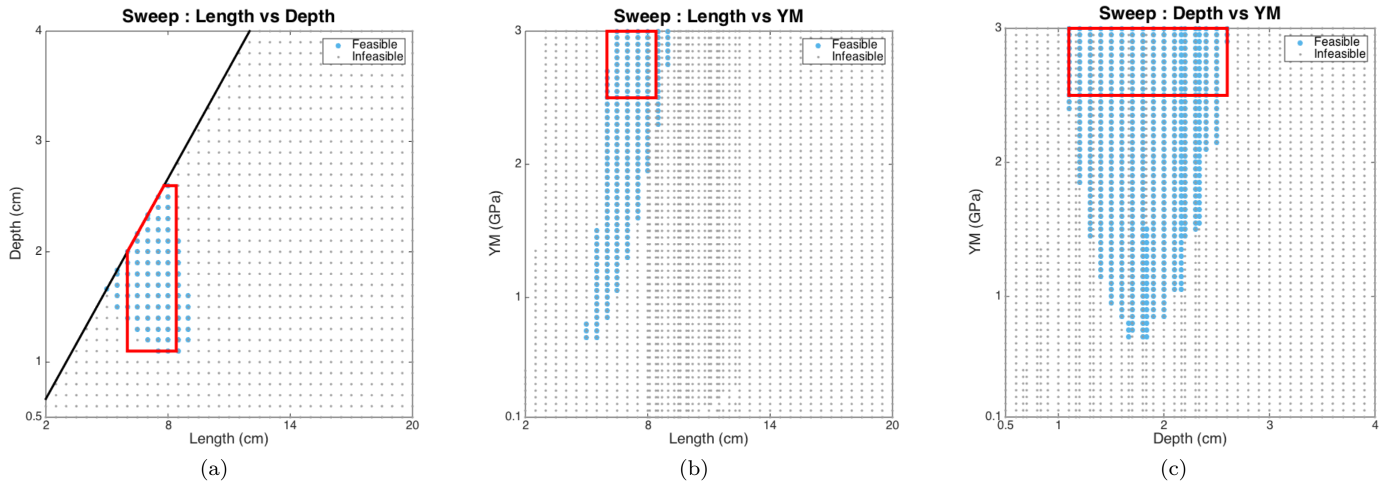


Fig. 9. Parameter sweep plots: (a) length vs. depth; (b) length vs. flexibility (c) depth vs. flexibility. The red boxes denote the limitations of adaptability found by the boundary selection method, and the black line in (a) relates to the length-depth limitation of the simulation model.

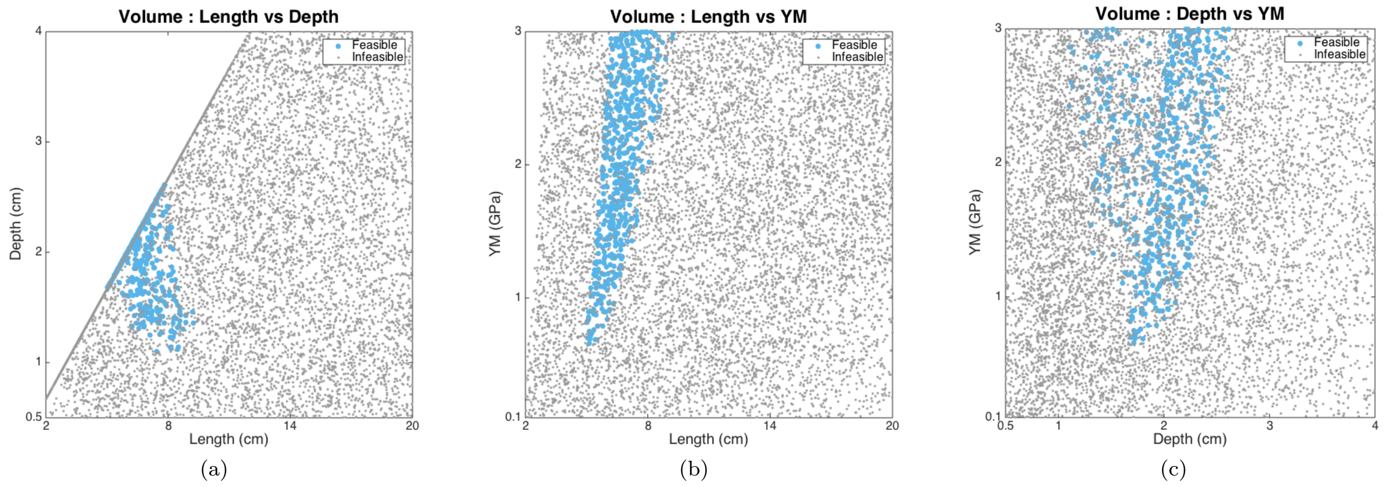


Fig. 10. Plots for mode discovery using volume selection: (a) length vs. depth; (b) length vs. flexibility; (c) depth vs. flexibility.

are not guaranteed to be feasible, we generate multiple (25) candidate scenarios per iteration. From these candidate scenarios, we choose the feasible scenario that produces the worst fitness score (largest average error) when evaluated with the current best MFAC controller. Essentially, we seek scenarios that are the most difficult for the adaptation process. At the end of the experiment, the set S contains 11 total scenarios. However, for comparison purposes, we cap the volume-based selection method to have a comparable number of evaluations as the boundary selection method. So, for each iteration, if $|S|$ is greater than five, only five scenarios are randomly selected for use in evolution. The 40 replicate experiments, 10 iterations, and testing of 25 random scenarios per iteration, yields 10,000 tested scenarios in total. Figure 10 shows that with far fewer simulations (10K vs approximately 62K), the volume scenario method defines the same boundaries as the full parameter sweeps shown in Figure 9. The advantage in efficiency is expected to increase with the number of scenario parameters, which will be the case for many cyber-physical systems. In fact, parameter sweeps may be completely infeasible under some circumstances.

The reader will notice that in Figure 10, some gray dots (representing infeasible scenarios) seem to fall within the discovered mode boundaries. This phenomenon is particularly evident in the depth vs. YM plot. We note that this appearance is simply a side effect of the pairwise plotting of the three parameters. Specifically those scenarios are infeasible because of the value of the third parameter (length in the case of the depth vs. YM plot). This relationship is also present in the data plotted in Figure 9, but does not appear in those plots because in the parameter sweeps, the values of parameters vary at regular intervals, and blue dots cover all the gray ones.

In addition to defining mode boundaries, the volume method generated effective adaptive controllers by sequentially integrating "difficult" scenarios into the evolutionary process. Figure 11 shows an example behavior of a simulated robotic fish that experiences damage halfway through operation. At 60 seconds, the fin length is reduced from 8.0 to 6.4 cm, depth is reduced from 2.6 to 2.1 cm, and flexibility is changed from 3.0 to 2.1 GPa. Despite these rather severe changes, the controller is able to quickly adapt and re-establish tracking.

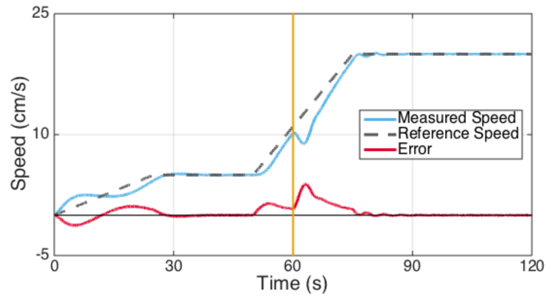


Fig. 11. An evolved MFAC adapting to sudden damage. At 60 seconds all fin morphology parameters are abruptly changed.

When comparing the two scenario selection methods, we found that both lead to similar algorithm convergence rates and similar MFAC behaviors. Table IV compares the fitness scores (the average mean-absolute-error recorded for each scenario used during fitness evaluation) for the overall best set of MFAC parameters from each experiment (lower values are better). The $rand_{boundary}$ and $rand_{volume}$ rows represents the mean fitness score for 100 randomly generated, feasible scenarios that are within the modes found by the boundary and volume selection methods, respectively. The table shows that the volume-based scenario generation approach does an equal or better job in every test case.

TABLE IV
MFAC Performance Comparison

	Boundary	**Volume**
$Base$	2.76%	2.60%
$length_{min}$	9.30%	7.63%
$length_{max}$	2.74%	2.73%
$depth_{min}$	6.23%	4.87%
$depth_{max}$	3.12%	2.92%
YM_{min}	2.98%	2.93%
$rand_{boundary}$	4.70%	4.54%
$rand_{volume}$	3.19%	3.14%

VI. Conclusions and Future Directions

In this paper, we proposed an approach to automatically discover the boundaries of adaptability for a cyber-physical system. Specifically, this approach characterizes the range of operation (i.e., *mode*) for a robotic fish and its adaptive control software. From the software's point-of-view, the morphological properties of the robot are aspects of an uncertain environment. The proposed approach to discovering mode boundaries involves generating scenarios within the range of the conditions that the system is expected to encounter and evolving an adaptive controller for those scenarios. Through a series of experiments, we found the volume-based scenario selection method to be both effective and computationally efficient. Although this approach accurately defined the mode in terms of the ground truth, for higher dimensional spaces more strategic methods of generating and selecting scenarios might be warranted. Additionally, once the boundaries of a mode are discovered, it will be useful to find control strategies for adjacent modes, as well as logic to switch among modes. In this way, the system can have predefined controllers that are effective for multiple scenarios, including fail-safe operation (e.g., when the system is damaged it can still be controlled to return to a base station). These topics are included in our ongoing investigations.

Acknowledgments

This work was supported in part by National Science Foundation grants CNS-1059373, DBI-0939454, and CNS-1305358, the Ford Motor Company, General Motors Research, and a grant from the Air Force Research Laboratory.

References

[1] P. Ioannou and J. Sun, *Robust Adaptive Control*. Dover Publications, Mineola, NY, 2012.

[2] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[3] G. S. Cheng, "Model-free adaptive (MFA) control," *Computing and Control Engineering*, vol. 15, no. 3, pp. 28–33, 2004.

[4] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, pp. 3 – 35, 2013.

[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992.

[6] D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary Robotics," in *Handbook of Robotics*. Berlin: Springer Verlag, 2008.

[7] H. Lipson, "Evolutionary robotics and open-ended design automation," in *Biomimetics*, B. Cohen, Ed. CRC Press, 2005, pp. 129–155.

[8] L. S. Coelho, M. W. Pessôa, R. Rodrigues Sumar, and A. Augusto Rodrigues Coelho, "Model-free adaptive control design using evolutionary-neural compensator," *Expert Systems with Applications*, vol. 37, no. 1, pp. 499–508, 2010.

[9] A. J. Clark, P. K. McKinley, and X. Tan, "Enhancing a model-free adaptive controller through evolutionary computation," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 137–144.

[10] X. Tan, "Autonomous robotic fish as mobile sensor platforms: Challenges and potential solutions," *Marine Technology Society Journal*, vol. 45, no. 4, pp. 31–40, 2011.

[11] S. Haykin, *Neural networks and learning machines*. Prentice-Hall, 2009.

[12] G. Cheng, *MFA in Control with CyboCon*. CyboSoft, General Cybernation Group, Inc, Rancho Cordova, CA, 2002.

[13] "Simulink: Dynamic system simulation for MATLAB, User's Guide," The MathWorks Inc., Natick, Massachusetts, USA, 1990-2013.

[14] J. Wang, P. K. McKinley, and X. Tan, "Dynamic modeling of robotic fish with a base-actuated flexible tail," *Journal of Dynamic Systems, Measurement and Control*, vol. 137, no. 1, August 2014.

[15] A. J. Clark, X. Tan, and P. K. McKinley, "Evolutionary multiobjective design of a flexible caudal fin for robotic fish," *Bioinspiration & Biomimetics, special issue on Bioinspired Soft Robotics*, vol. 10, no. 6, November 2015.