Proceedings of the 6th International Workshop on
Evolutionary and Reinforcement Learning for
Autonomous Robot Systems, in conjunction with the

Just Keep Swimming:
Accounting for Uncertainty in Self-Modeling Aquatic Robots

Matthew J. Rose, Anthony J. Clark, Jared M. Moore and Philip K. McKinley[1]

Abstract.
A robust robotic system should be able to overcome unforeseen conditions, including physical damage and component failure occurring after deployment. A self-modeling system maintains an internal image of itself, which can be updated to reflect incurred damage. The robot can use this model to derive (or evolve) new behaviors such as gaits that account for the damage. In this paper we describe an approach to self-modeling for aquatic robots. The aquatic environment presents unique challenges to the self-modeling process, including the inherent uncertainty in the robot's orientation and configuration. We propose and evaluate two approaches to automatically infer missing contextual information, which otherwise complicates the task of developing an accurate model. We demonstrate the effectiveness of these methods on a particular aquatic robot intended for remote sensing.

1 Introduction

Increasingly, robotic systems are required to be resilient to adverse conditions that occur after deployment. A particularly challenging problem is how to overcome situations where the structure or functionality of the robot is fundamentally changed due to physical damage or mechanical/electronic failure. While pre-programmed recovery modes offer a computationally efficient means of addressing common problems during system operation, they are highly dependent on the ability of an expert designer to anticipate failure cases. On the other hand, if the robot has a means to automatically discover changes to itself, it might be able to generate a new compensatory behavior even when faced with unanticipated scenarios.

One approach to achieving such run-time adaptability is *self-modeling*, where the system maintains an internal "mental image" of itself [3]. Current science suggests that animals maintain a mental predictive model of themselves and use this model to plan appropriate behaviors [29]. In the case of a robot, if the system has an accurate simulation model of its own morphology, including sensors and actuators, it can derive new behaviors dynamically using the model rather than the physical system. For example, consider a situation with an aquatic robot, where the motor actuating a pectoral fin fails and stops working entirely. Executing pre-programmed control sequences for specific gaits is unlikely to produce expected movements. On the other hand, if the system could dynamically produce a new model of itself, then it might discover that one of its flippers is seized and derive a new gait to compensate for this limitation.

The search capability of computational evolution has been shown to be effective in discovering both models and corresponding behaviors for robots. Bongard and Lipson [20] introduced the Estimation-Exploration Algorithm (EEA), a general purpose algorithm for reverse engineering complex, non-linear systems. At its core, the EEA is a co-evolutionary process that alternates between gaining information about the target system (exploration) and integrating that information into its model hypothesis (estimation). By evolving multiple model hypotheses in parallel and selecting the most appropriate hypothesis for testing, this approach greatly reduces the number of physical tests needed, relative to a traditional "generate and test" methodology [22]. Bongard and Lipson demonstrated the use of EEA to automatically diagnose failures in terrestrial robots (quadrupeds and hexapods) and generate compensatory behaviors without human intervention [7].

In this paper, we adopt the general EEA approach but extend it to address uncertainties associated with a specific aquatic robot, shown in Figure 1. The robot has a caudal fin and two pectoral flippers. Although we have previously fabricated this robot and evaluated it in an aquatic testbed, in the study described here we investigate the self-modeling process using only a simulated robot modeled after the physical counterpart. Applying the proposed process to online evolution in physical robots is part of our future research.

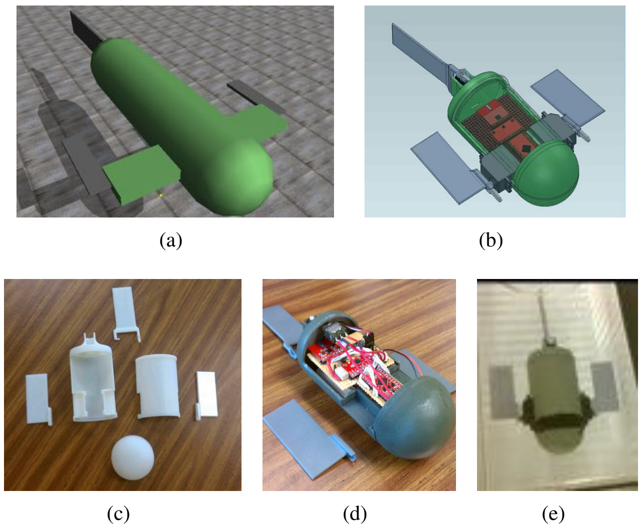


Figure 1. Modeling and fabrication of an aquatic robot [24]: (a) simulation model in Open Dynamics Engine (ODE); (b) corresponding SolidWorks model for fabricating prototype; (c) 3D-printed passive components of prototype; (d) integration of electronic components and battery into the prototype; (e) assembled, painted and waterproofed prototype in an elliptical flow tank. The main body of the physical prototype is 13cm long and 8cm in diameter, the pectoral flippers are 8cm long and 2cm wide, and the tail fin is 7 cm long, 3 cm tall, and 2 cm thick.

[1] Department of Computer Science and Engineering, Michigan State University, East Lansing, Michigan, USA, Contact email: mckinley@cse.msu.edu

A particularly difficult problem arises from the uncertainty surrounding initial orientations and configurations of the robot during the self-modeling process. Whereas with terrestrial robots studied in previous works the algorithm could exploit known starting positions in order to evolve accurate models, the dynamics of the aquatic environment complicates this task. For example, the response of our target robot to a given set of servo motor commands depends on the initial position of its flippers. However, due to hardware limitations specific to this device, the initial positions of the flippers cannot be determined from the robot. To address this issue, we propose and evaluate two inference mechanisms (In-Band Inference and Out-Of-Band Inference) that reduce the uncertainty regarding contextual information. Here, we apply these techniques to help determine the starting positions of the aquatic robot's flippers and fin, however, the approach is generally applicable to other uncertain factors that affect modeling performance. The effectiveness of this approach is demonstrated on a simulated target robot with a damaged pectoral fin. Specifically, the modified EEA produces a model approximating the damaged system, with which a new gait is evolved.

2 Background and Related Work

Although the focus of our research lies in the field of robotics, self-modeling is a general concept and has been applied to a wide variety of systems. Much of this research has been conducted in the context of autonomic, or self-*, systems [17, 23], which are designed to be capable of self-management and self-healing, among other functions. In particular, computational reflection [21] has provided a foundation for extensive work on self-modeling software, including adaptive middleware [4] and formal models for developing adaptive software [30]. More recently, the search capability of evolutionary computation has been wielded to help software find effective and safe paths to reconfiguration [26] and address uncertainty [11]. In software systems, self-modeling typically addresses software structure, whereas in robotics, it is often concerned with physical structure and mechanical functionality of the system. Such is also the case with many biological organisms.

Zoologists recognize the importance of studying the behaviors of an organism in the context of its *Umwelt* [13]. This concept refers to a *coevolved* linkage between perception and action, whereby the complexity of an animal's control systems is governed by the the capabilities of its physical body, the behavioral problems it faces, and the environment in which it operates. The Umwelt concept is also relevant to engineered robotic systems, where embedded computing systems need to interpret sensed information and respond accordingly through robot actuators. Indeed, the adaptability and robustness exhibited by natural organisms has led to many *bio-inspired* design approaches. In *biomimetic* methods, a structure or behavior found in nature is replicated in the artificial system. However, simply codifying animal behavior in a robotic controller does not account for, or exploit, the vast differences between engineered systems and natural organisms. A complementary bio-inspired approach is to harness the *process* that produced robust systems in nature: evolution. Evolutionary computation (EC) methods [12] codify the basic principles of genetic evolution in computer software. EC methods are particularly effective at addressing problems involving large, multidimensional search spaces. The most well-known EC method is the *genetic algorithm* (GA) [15], an iterative search technique in which the individuals of a population encode candidate solutions to an optimization problem. GAs and related EC methods have proven to be effective in a wide variety of science and engineering domains, rivaling and even surpassing human designers [1].

In addition to solving optimization problems, some EC approaches produce software for controlling physical devices such as robots. For example, neuroevolution [28] is a machine learning method in which GAs are used to train artificial neural networks (ANNs), which in turn can control physical systems such as robots. ANNs are particularly attractive for controllers because they have low computational cost, and their inputs and outputs can correspond directly to sensors and actuators, respectively. In *evolutionary robotics* [19], an artificial genome encodes a robot's control system and possibly its morphology. The control program is downloaded into a real or simulated robot, which is then let "loose" in an environment. The fitness of the system is evaluated with respect to performing tasks. The most fit individuals in the population are allowed to reproduce, with random mutations and recombination, to form the next generation on which this cycle is repeated until a satisfactory solution evolves [14]. Over the past 15 years, extensive progress has been made in several aspects of evolutionary robotics, including evolution of neurocontrollers for walking robots, finless rockets, and behaviors such as foraging and game playing.

An issue for a simulation-developed solution is how well it transfers into a physical robot. The so-called "reality-gap" arises when solutions that work well in a simulated environment face issues in a physical environment that were either unforeseen or incorrectly modeled [10, 16]. Approaches to this problem include evolving the simulator in conjunction with a robot [9] and directly rewarding solutions for performing similarly in reality and simulation [18]. However, even if the deployed robot initially behaves as in simulation, over its lifetime it is likely to experience hardware decay, software bugs, and even physical damage. Self-modeling approaches provide an avenue for robots to continuously adapt to the ever changing operational state and environmental conditions faced in the real world.

Bongard and Lipson proposed the EEA in part to enable remote robots to automatically diagnose failures and generate compensatory behaviors without intervention [7]. The algorithm's ability to automatically derive a damage hypothesis (diagnosis) and then evolve and effective behavior that maintains most of the intended functionality was demonstrated for simulated quadrupedal and hexapod robots. The extensibility of this algorithm was later demonstrated with the automated inference of gene regulatory networks [8]. In this paper, we further extend the EEA method to deal with operational uncertainty. Specifically, we alter the EEA to include an inference component from which the system extrapolates its own starting orientation. This capability is particularly important in aquatic environments, where the robot's orientation may be highly dynamic.

When performing a damage diagnosis on a real-world system, the algorithm must also account for its environment, a factor that is beyond the system's control. Fundamental work was completed in this area by Bongard and Lipson in [5], an extension to the work in [7]. In this approach, the algorithm is extended to not only derive a damage diagnosis for itself, but also to make basic estimates of its environment. This method, however, applies only to static environments. The approach presented in this paper can infer unknown attributes about the system being modeled or its environment on a per-trial basis, removing the requirement that the environment be static.

3 Methods

In this section, we describe the simulation environment, evolutionary computation methods, basic self-modeling algorithm, and the two proposed methods of accounting for uncertainty.

3.1 The Aquatic Robot

As shown in Figure 1, the aquatic robot targeted in this study comprises a capsule-shaped main body, two pectoral flippers, and a single caudal fin. The main body of the physical robot contains an Arduino microcontroller board, two lithium polymer batteries, three servo motors, and a 6-axis inertial measurement unit (IMU). The IMU includes an integrated gyroscope and accelerometer, enabling the robot to compute its current position, orientation and velocity in three-dimensional space. The robot is designed to be configurable, in that plastic sleeves enable 3D-printed flippers and fins with different characteristics (size, shape, flexibility) to be swapped in and out for different experiments. The servo motors powering the flippers are a continuous rotation type that offer only a variable speed in either direction with no position feedback, a feature typical of servo motors of this size. The lack of position information complicates the self-modeling process, since the behavior of the robot is highly coupled to its initial configuration. This inherent uncertainty in the robot's initial configuration is the basis of and motivation for this study.

In the future we plan to integrate and test self-modeling algorithms using the physical robot. However, in this study we explore self-modeling algorithms using only a simulated version of the robot. This approach enables us to efficiently explore novel approaches to dealing with uncertainty as part of the self-modeling process. Promising techniques can later be evaluated through experiments in Evolution Park, a testbed for research in autonomous systems and evolutionary robotics located at our institution; see Figure 2.

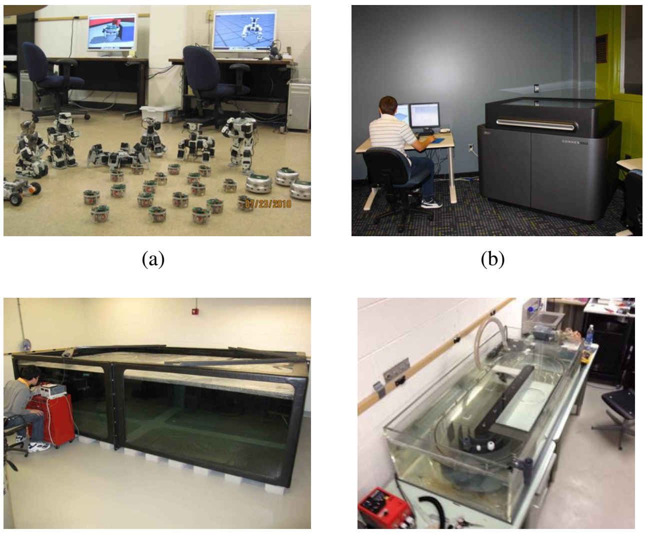


Figure 2. Components of the Evolution Park experimental environment: (a) swarm of commercial terrestrial microrobots and interactive simulation cluster; (b) Connex 350 multi-material 3D printer for fabricating robotic components; (c) 4500-gallon custom-built tank for aquatic robot experiments; (d) elliptical flow tank for studying behaviors such as station keeping. Additional details are available at http://www.cse.msu.edu/evopark.

We simulated the robot using the Open Dynamics Engine (ODE) [25], an open-source software package for simulating rigid-body dynamics. We have previously developed a computationally efficient fluid dynamics model for ODE, enabling us to apply this platform to the investigation of aquatic robots [24]. The simulated robot approximates its physical counterpart in form and also offers similar sensing modalities. Specifically, a simulated IMU provides inertial data for use by the robot controller and the self-modeling algorithm. In this study, the controller is an evolved artificial neural network (ANN), described later, but other types of controllers could be used. As with the physical robot, the position of the flippers is not directly available and must be inferred. Although the physical robot is designed to accommodate flippers of varying material composition and flexibility [24], in this initial study the flippers of the simulated robot are rigid.

For the purpose of the self-modeling algorithm, we have defined a standard way of controlling the simulated aquatic robot. An atomic control command consists of a three-tuple of servo motor speeds, one for each of the two pectoral flippers and one for the caudal fin. The pectoral flippers are allowed to operate with an angular velocity of $[-0.5,\ 0.5]$ revolutions per second, and the caudal fin has a range of [0.0, 1.0] Hz oscillation within a $[-30, 30]$ degree sweep. When given a command, the simulation sets its servo motors to the commanded angular velocities and simulates itself for a fixed (but configurable) amount of time. IMU sensor readings for pitch, roll, and yaw are recorded at each step of simulation and placed into a log of samples. Once the simulation completes, the list of sensor samples is aggregated into a *behavior summary*, which is returned as output of the simulation. In preliminary studies we experimented with different methods of aggregating these data, in an attempt to differentiate responses of the robot. Methods we explored included the mean angular velocity in each axis and the root mean squared angular velocity in each axis. However, we found the Discrete Fourier Transform (DFT) to be most effective in capturing differences among robot movements, and we used it as the behavior summary for all the experiments presented in this paper.

In addition to the control scheme, the simulation is also parameterized to allow for changes in the robot morphology. Specifically, the dimensions of each of the three flippers can be altered. Since the ODE engine is unitless, only the relative dimensions are of interest. The pectoral flippers can be sized in the range $[0, 1]$, $[0, 2]$, $[0, .5]$ for (x, y, z), respectively, where the x dimension is width of the robot, y is the length along the robot's capsule, and z is the vertical dimension. The caudal fin can be sized in the range $[0, .5]$, $[0, 2]$, $[0, 1]$ for (x, y, z), respectively. The range of potential flipper sizes is intentionally larger than the physical robot's flippers to allow the algorithm to accommodate changes in morphology such as damage, wear, or the entaglement of a flipper with foreign objects.

3.2 Self-Modeling Algorithm

As noted earlier, we adopt the EEA [6] to produce candidate models of the *target system* (in this case a simulated robot). The EEA is a general purpose algorithm for reverse engineering complex, nonlinear systems. As depicted in Figure 3, the EEA is a co-evolutionary process that alternates between gaining information about the target system (*exploration*) and integrating that information into its model hypothesis (*estimation*). The combination of an exploration phase and an estimation phase is referred to as a *round*. These two phases are iterated until a maximum number of interactions with the target system have occurred, or it is determined that the current model hypothesis accurately represents the target system. Determination of a good stopping criteria is a complicated subject in its own right. For this study we have simply allowed the modeling process to run until a computation time limit has been exceeded.

Exploration Phase. The exploration phase is responsible for collecting behavioral information from the target system for use in the estimation phase. Specifically, the algorithm attempts to identify an

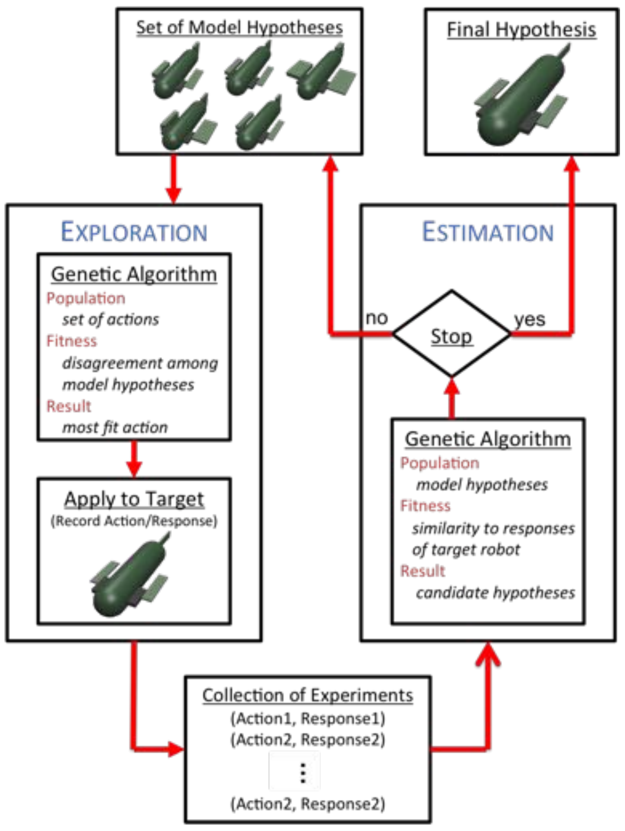


Figure 3. Basic operation of the Estimation-Exploration Algorithm [6].

action (in our case a particular movement of the robot's flippers) that provides maximal information regarding the current hypotheses of the robot morphology. A genetic algorithm is used to search for such an action; a population of individuals, each an encoding of a candidate action, executes for a predetermined number of generations. The fitness of a candidate action is based on the amount of disagreement it generates among the current model hypotheses, as discussed below. The action with the highest fitness in the final generation is selected and applied to the target system (in our case a simulated robot, but in the field a physical robot). The response from the target is collected from the system's sensors and recorded, along with the action that generated it (collectively known as an *experiment*), for use in the estimation phase. This guided selection of actions, that is, applying evolution to simulated robots until the final step, has been shown to reduce the number of physical tests on the target system by several orders of magnitude [7].

For our study, an action is encoded as a binary string of 18 bits: 6 bits to encode each of the pectoral servo speeds, and the remaining 6 bits for the caudal fin frequency. The fitness of a potential action is based on the diversity of behaviors it elicits from the set of candidate morphology models. That is, an action that produces a similar response in most candidate models receives a low fitness score, while one that produces markedly different responses receives a higher fitness. Formally, the fitness of action a is defined as

$$ACT\_FIT(a) = \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} [1 - sim(B_{i,a}, B_{j,a})] \div \binom{M}{2}, \quad (1)$$

where M is the number of candidate models, $B_{i,a}$ is the behavior when action a is applied to model i, and sim is the similarity measure between two behaviors. As noted earlier, after exploring alternatives we implemented a DFT-based definition of $sim()$, specifically, the similarity of two behaviors B and B' is defined as:

$$sim(B, B') = \frac{1}{f} \sum_{k=1}^{f} 1 - \frac{|c_k(B) - c_k(B')|}{MAX}, \quad (2)$$

where f is the total number of discrete frequencies in the series, $c_k(B)$ is the kth coefficient for behaviour B, and MAX is a predetermined maximum coefficient value (in our system, $MAX = 100$).

Estimation Phase. The estimation phase is responsible for refining the hypothesis of the robot's morphology. To this end, it generates a set of candidate hypotheses that best explain the experiments performed in prior exploration phases. As in the exploration phase, a genetic algorithm is used for this search process, with candidate models encoded as binary strings. In this study, each model encoding comprises two sets of pectoral fin dimensions (15 bits each) and one set of caudal fin dimensions (15 bits), for a total of 45 bits. These parameters produce a discretized set of potential flipper/fin dimensions, as shown in Figure 4. For example, the width, length and thickness of each pectoral flipper are represented by 5 bits in the x dimension, 6 bits in the y dimension, and 4 bits in the z dimension. For each experiment found during the exploration phase, the corresponding action is applied to the candidate model, and the response is compared to that of the target system. The fitness of a candidate model is based on how well it reproduces the response of the target. Formally, the fitness of model M_i is defined as:

$$MOD\_FIT(M_i) = \frac{1}{E} \sum_{e=1}^{E} sim(B_{i,a_e}, T_{a_e}) \quad (3)$$

where E is the number of currently recorded experiments, $B_{i,a}$ is the behavior of candidate model i for action a, a_e is the action associated with experiment e, and T_{a_e} is the recorded behavior of the target system from experiment e. The similarity measure, sim, is the same as for the exploration phase. The model optimization genetic algorithm is also configured with a fixed population size and allowed to evolve for a pre-determined number of generations. The final model hypothesis is selected from the final generation as the candidate with the highest fitness.

The main difficulty in applying these methods directly is that the response of the robot to a given set of commands is highly dependent on the initial positions of the flippers. Without direct knowledge of this information, we need to infer the positions as part of the evolutionary process.

3.3 In-Band Inference

In our first inference method, referred to as *In-Band Inference*, we attempt to infer the unknown initial flipper positions in conjunction with model optimization in the estimation phase. To do so, we augmented the definition of an experiment on the target system to include meta-information about the context in which it was performed, specifically, a range of angles for each flipper/fin and a probability indicating the confidence that the true position of the flipper lay in that range. Let us first describe how this information is used. We modified our simulator's PERFORM() method, which commands the simulation to run an experiment, to incorporate this information. Pseudocode is shown below. The inferred initial position of each flipper is stored within the experiment as a range from some min_angle to

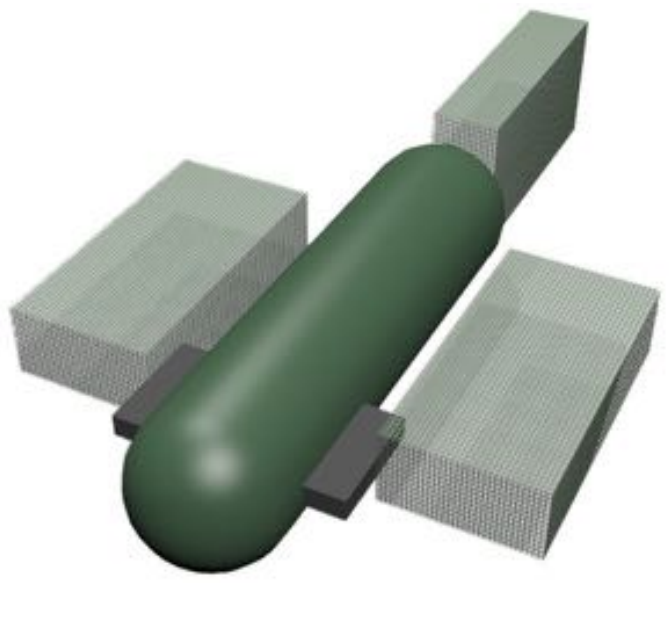


Figure 4. The simulated aquatic robot is designed to have configurable flipper sizes. Since the underlying representation of the model is a binary string, dimensions are discretized as shown.

some max_angle, and returned by the $experiment.getAngles()$ method. The $setFlippers()$ function samples uniformly from the range $[min\_angle, max\_angle]$ for each flipper.

```
function PERFORM(experiment)
    $confidence \leftarrow experiment.getConfidence()$
    if $rand() \leq confidence$ then
        $angles \leftarrow experiment.getAngles()$
        $setFlippers(angles)$
    else
        $randomizeFlipperPositions()$
    end if
    $simulate(experiment)$
end function
```

The In-Band algorithm is based on the assumption that models with higher fitness levels are likely to have good approximations of the initial configuration. Pseudocode for the modified estimation phase is given below. After evaluating each of the model hypotheses, the estimation phase records the initial flipper positions of the k models with the highest fitness using $getKBestPos(models)$. It then computes the smallest inclusive angles between these positions on a per-flipper basis, using $smallestInclusive(bestPos)$, and stores those inclusive angles in the experiment using $setAngles()$. At the end of each EEA round, the inferred values for the resulting experiment are "locked in" by setting the corresponding confidence level to 1.0. Doing so ensures that as part of a given round N, the algorithm explores initial conditions only for the most recent experiment; the confidence levels of experiments from prior rounds, that is, the first $N-1$ experiments, are all 1.0. In contrast, the confidence level of the most recent experiment is set to 0.5. This setting effectively means that half the population is used to explore previously identified areas of high fitness, while the other half of the population randomly searches for new high fitness areas.

```
function ESTIMATE(experiments, models)
    $experiments.last.setConfidence(0.5)$
    for $generation = 1 \rightarrow maxGenerations$ do
        ... pre-existing estimation code ...
        $bestPos \leftarrow getKBestPos(models)$
        $angles \leftarrow smallestInclusive(bestPos)$
        $experiments.last.setAngles(angles)$
    end for
    $experiments.last.setConfidence(1.0)$
end function
```

A potential drawback of this approach is that it can lead to instability in the optimization process. A model with potentially high fitness (in terms of flipper morphology) might be lost due to receiving a bad set of initial flipper positions, significantly lowering its apparent fitness. Additionally, a model's fitness can vary from generation to generation, since it is initialized with different flipper positions in each. We also note that, while this method is conceptually similar to Evolutionary Mutliobjective Optimization (EMO) [2], no information about the flipper positions is encoded in the model's genome. Rather, the inferred information is stored in the experiment's meta-information and affects the model's fitness only implicitly. The reason it cannot be directly added as a fitness component is that the target position is unknown, so a fitness value for the flipper positions cannot be determined.

3.4 Out-of-Band Inference

Our second inference method attempts to decouple the tasks of model optimization and resolution of unknown contextual information. In this approach, the model optimization process is paused at regular intervals, in order to perform flipper position inference. The period is configurable, and in our tests was 35 generations. This setting caused the model optimization process to be paused for flipper position inference three times during its 100 generations, at generations 0, 35 and 70. The PERFORM() simulation method presented with the *In-Band Inference* algorithm was used without change. Pseudocode for the modified estimation phase is given below. In this algorithm the $optAng()$ function initializes a genetic algorithm to find an optimal set of initial flipper positions for the most recent experiment found during exploration. The fitness of candidate initial configurations reflects their ability to help the candidate models reproduce the target behavior. That is, a set of initial flipper positions that enable the candidate models to accurately reproduce the current experiment will receive a high fitness. In addition, we note that in contrast to *In-Band Inference*, this algorithm immediately sets the confidence of new experiments to 1.0. This approach effectively disables the flipper position randomization mechanism and implies that all models will use the inferred initial conditions.

```
function ESTIMATE(experiments, models)
    $experiments.last.setConfidence(1.0)$
    for $generation = 1 \rightarrow maxGenerations$ do
        ... pre-existing estimation code ...
        if $generation\%periodicity = 0$ then
            $angles = optAng(models, experiments.last)$
            $experiments.last.setAngles(angles)$
        end if
    end for
end function
```

Next, we describe a set of experiments intended to evaluate the effectiveness of EEA when modified to incorporate the two inference algorithms.

4 Results

We began our experiments by establishing two baselines: (1) performance of the self-modeling algorithm with known initial flipper/fin positions and (2) performance of the unmodified EEA in the presence of unknown starting positions. Next, we evaluated the effectiveness of the two inference methods at resolving this unknown contextual information.

Within each set of experiments, the performance of the algorithm is measured as its objective fitness versus EEA round. The objective fitness of a model is defined as the average dimensional error over each of the flipper dimensions. More formally, the objective fitness is defined as:

$$OBJ\_FIT = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} \frac{|model_i - target_i|}{max_i - min_i} \right), \qquad (4)$$

where n is the number of morphological variables (in our case, 9), $model_i$ and $target_i$ are the i^{th} flipper dimensions of the candidate and target models respectively, and min_i and max_i are the minimum and maximum possible values of the i^{th} dimension. We note that the objective fitness function is not a measure that is used in the evolutionary optimization process, but rather as an oracle measurement we monitor from the outside to gauge how the optimization process is proceeding.

Each test was executed as a set of 25 replicate runs, each of which was seeded with a different random number seed. The individual runs were executed as a single threaded process on late-model, commodity hardware and given 48 hours of CPU time to complete 25 rounds of the EEA process before being terminated. The orientation of the target robot was randomized between every action. This was true of every series reported in the paper and reflects what would likely be the case in a real-world situation. The genetic algorithm in the exploration phase used a population of 25 actions and executed for 25 generations. The genetic algorithm in the estimation phase used a population of 25 model hypotheses and executed for 100 generations. Both genetic algorithms used tournament selection with a tournament size of two, a crossover rate of 0.75 and a mean mutation rate of one gene per genome. Simulations consisted of a configurable number of discrete time steps, where each time step was 0.01s of simulation time, with the total number of steps being a configurable power of two.

4.1 Ideal Baseline

In order to establish a ceiling on performance, we first executed a baseline with known flipper positions. In this configuration, the target aquatic robot has random flipper orientations at the start of each EEA experiment, but those initial flipper positions are noted and passed back to the modeling process for use by candidate models. During the modeling process, each model is able to read the flipper position information and properly initialize itself before executing the experiment. This case represents what the modeling process could achieve if the flipper positions were a measurable quantity rather than a contextual uncertainty.

Four data series were collected for this configuration, representing experiment execution times of 1.28s, 2.56s, 5.12s and 10.24s. One

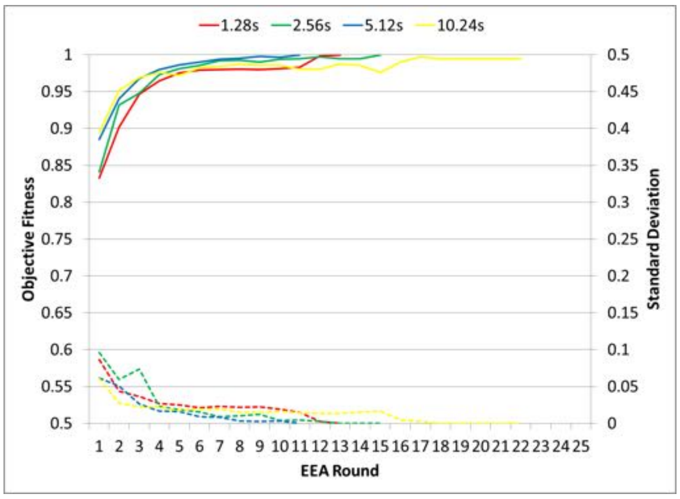


Figure 5. Performance of the modeling process versus EEA round number when the initial positions of the robot's flippers are known. Solid lines represent objective fitness, and dashed lines represent standard deviation.

would expect longer experiment execution times to produce more accurate models. As can be seen in Figure 5, the performance of this algorithm is excellent, even for very short experiments of only a second or two. Objective fitness approaches 1.0 after only a few experiments (EEA rounds) on the target robot, with longer action time yielding slightly higher fitnesses earlier. In addition, the standard deviation between replicate runs is quickly reduced to near zero, indicating that the replicate runs are consistently performing well. This performance is not surprising, as the effectiveness of the EEA modeling process in quickly finding good model hypotheses, when the initial conditions are known, has been previously demonstrated [5].

4.2 No-Inference Baseline

In our second study, we removed the assumption of known initial flipper positions, introducing uncertainty into the modeling process. The target aquatic robot is randomized before each EEA experiment, but this time the initial orientations of its flippers are not stored as meta-information in the experiment. During the modeling process, each model has no contextual information to reference, so it initializes its flippers to a zero state (both pectoral flippers in an ”arms-back” position, and caudal flipper at zero degrees). This case represents the performance of the unmodified EEA modeling process when faced with uncertainty.

Again four data series were collected, representing 1.28s, 2.56s, 5.12s and 10.24s of experiment execution time. As can be seen in Figure 6, the performance of the EEA modeling process degrades considerably relative to the ideal baseline. Objective fitness for all simulation lengths is poor (in most cases no better than 75%) and does not show improvement over time. In fact, most of the series actually lose objective fitness over time. The standard deviation between replicate runs is much higher than the ideal baseline, and is not reduced over time. This baseline affirms our initial assumption that the behavior of models, and therefore the modeling performance, is tightly coupled to the initial configuration of the aquatic robot.

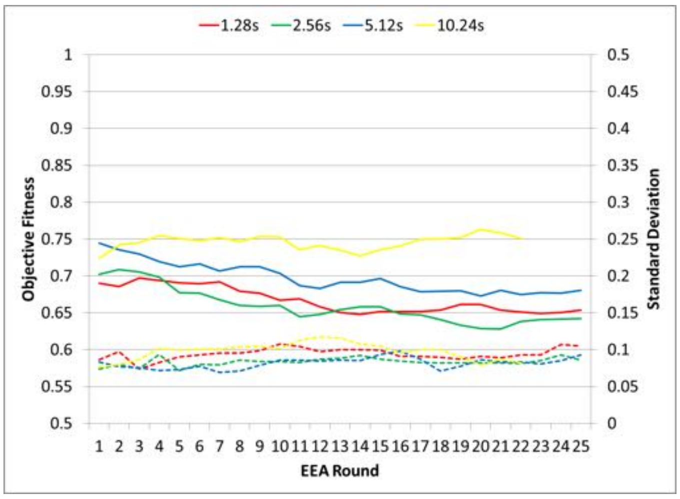


Figure 6. Performance of the modeling process versus EEA round number when the initial positions of the robot's flippers are unknown and models are initialized to a zero state.

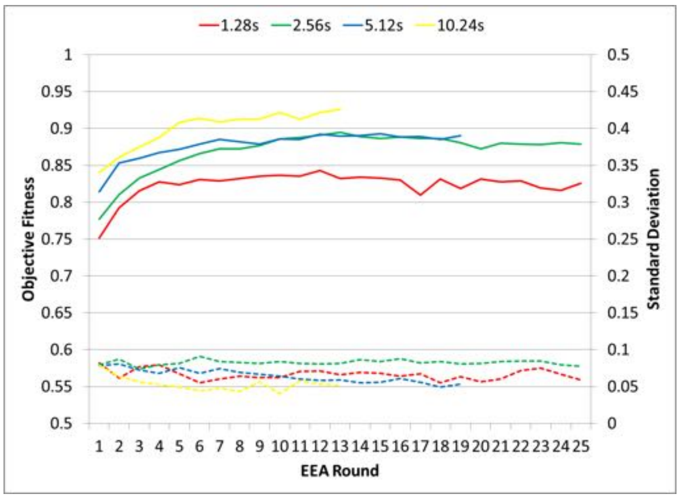


Figure 8. Performance of the modeling process versus EEA round number with *In-Band Inference* and a larger population size of 200 individuals.

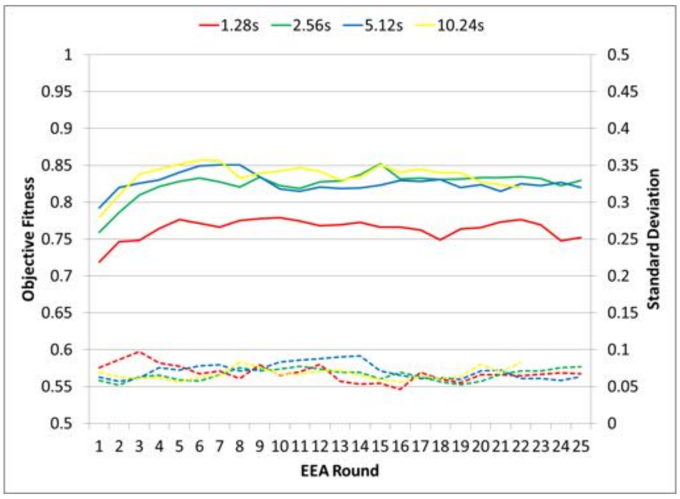


Figure 7. Performance of the modeling process versus EEA round number with *In-Band Inference*.

4.3 In-Band Inference

Next, we evaluated the *In-Band Inference* algorithm. In this configuration, the target system is randomized between each run and, as in the no-inference baseline, the starting position of the flippers is not stored as meta information in the experiment. Once the modeling process begins, the flipper/fin positions of each model are randomized and the inference algorithm is allowed to process the information to deduce likely flipper positions. As the GA works to find likely flipper dimensions, the *In-Band Inference* works to reduce the uncertainty in flipper/fin position, hopefully producing a better estimation of the other parameters through a reduction in uncertainty.

This algorithm was executed with a few different values of K (the number of elite individuals whose starting conditions are analyzed), and a value of K=3 offered the best performance. The results of applying this inference algorithm with K=3 are reported in Figure 7. As shown, the algorithm is able to improve the objective fitness of all series by 10-15%, demonstrating its ability to reduce the uncertainty in initial flipper configurations. The standard deviation between runs is also reduced across the board, but remains mostly constant over time.

After analyzing the results of this algorithm, we observed that the inferred flipper angles never seemed to converge. Instead, the algorithm would appear to identify a set of highly fit initial conditions and begin to converge, then suddenly jump to an entirely different set of initial conditions. We hypothesized that this sporadic behavior could be the result of the rather large number of individuals (half the population) that were randomized each generation. In an effort to curtail this behavior, we implemented a dynamic split ratio between models that would be randomized, and those that would use the inferred angles. Initially the population is split in half (randomized vs. inferred) as before, but over time this ratio is reduced such that by the midpoint of model optimization, 100% of the models are using the inferred angles. We reasoned that increasing level of confidence in the inferred angles would help the algorithm to converge to flipper orientations, and then spend the remaining optimization generations working solely on deducing the dimensions of the flippers.

Unfortunately, the results of this modification did not show any improvement. Objective fitness and standard deviation for all runs remained at similar levels to the basic algorithm original In-Band algorithm. After some additional analysis, we decided to test another modification to the algorithm. The previous versions of the algorithm used flipper angles uniformly sampled from within the inferred range. This mechanism provided a good search technique for exploring within the bounds, but did not provide a means for the algorithm to locate good candidates just outside these bounds. In order to allow the algorithm to "walk" the inferred angles to one side or the other, we extended the range of the inferred angles to reach 20% past the limits on either side.

However, the results from this modification told a similar story as before: objective fitness remained nearly the same, and variance between replicate runs was still high. It appeared that this methodology of inferring angles was simply too disruptive to the modeling process, at least with current optimization parameters. To further explore the bounds of this algorithm, we increased the population of model candidates from 25 to 200. This modification improved objective fitness. Results are shown in Figure 8, where the objective fitness is slightly below 90% and the standard deviations is under 10%.

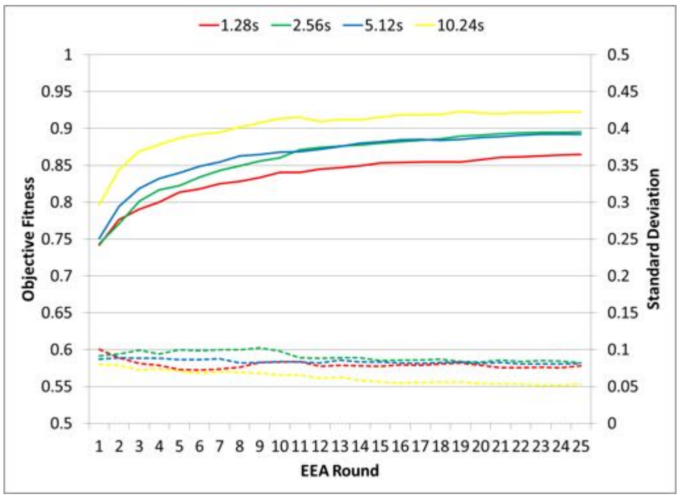


Figure 9. Performance of the modeling process versus EEA round number with *Out-Of-Band Inference*.

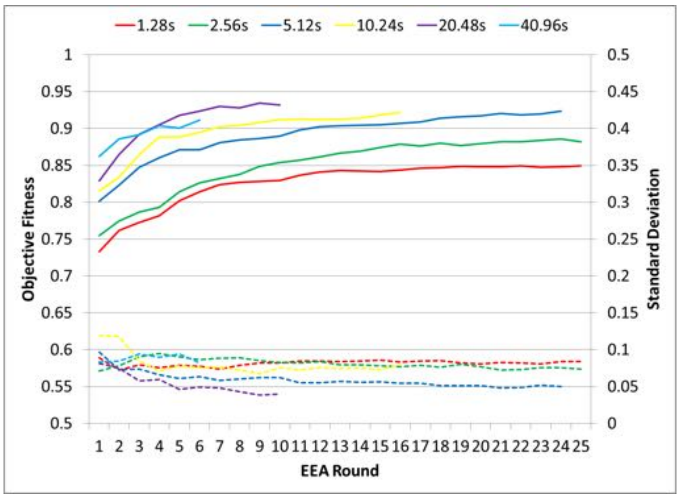


Figure 10. Performance of the modeling process versus EEA round number with *Out-Of-Band Inference*, an increased population size of 100, increased meta-level population size of 50, and extended simulation times.

4.4 Out-Of-Band Inference

After observing the disruption to the underlying optimization process caused by *In-Band Inference*, we developed the *Out-Of-Band Inference* algorithm, described earlier. We hoped that by separating the two tasks of flipper size estimation and flipper position inference, the algorithm would produce better, more consistent results.

Our first run of this algorithm, reported in Figure 9, showed promising results. Objective fitness approached 90%, was monotonically increasing and consistent from round to round. The standard deviation between replicate runs was similar to results obtained using the *In-Band Inference* algorithm. In order to explore the bounds of this algorithm's performance, we again decided to increase the population size of the models in the estimation phase. However, increasing the model population to 100 (from 25) resulted in prohibitively long computation time, since complexity is proportional to the product of estimation phase population size and the meta-level algorithm's population size. In seeking ways to reduce complexity, we hypothesized that it was only necessary to check that the fitness of a small sample of the models was maximized, rather than the entire population. This hypothesis appears to be correct, as demonstrated in Figure 10. Despite only using a sample of 5 models instead of the entire population of 100, the algorithm still effectively increased the objective fitness to the best levels we had seen thus far. With a reasonable performing self-modeling algorithm in hand, we next tested it on a simulated robot that has incurred damage.

5 Testing on a Simulated Robot with Damage

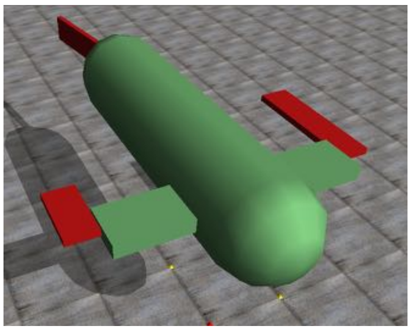
In this section, we present results of a case study where we applied the original EEA approach and our *Out-of-Band Inference* enhanced EEA algorithm in order to estimate the morphology of a simulated target robot with a damaged pectoral flipper (half the flipper is missing); see Figure 11(a). In Figure 11, flippers and fins are highlighted in red for clarity. Three runs were conducted using morphological models derived from, respectively, the standard EEA approach, our *Out-of-Band Inference* EEA, and the original configuration for our robot (i.e., no damage). In each case, after producing a model of the robot, we evolved a controller for each model, with a goal of enabling the robot to swim forward effectively given this damaged state.

As shown in Figures 11(b) and 11(c), the two EEA approaches arrived at very different morphologies, with the *Out-of-Band Inference* EEA result closely resembling the damaged robot. These individuals were then placed into a digital simulation environment, where we used the NEAT algorithm [27] to evolve artificial neural networks (ANNs) as controllers for them. We chose to use ANN controllers based on our previous experience with them in a station keeping task [24]. Fitness was based on swimming as far forward as possible in 20 seconds of simulation time.

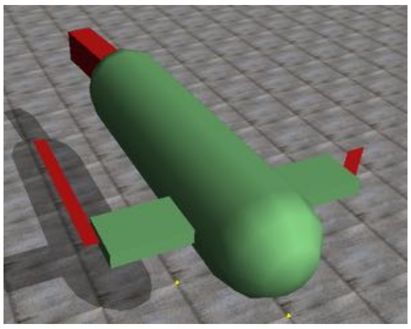
As the goal of self-modeling is to produce internal representations of the robot and apply that model to controller development, solutions were then evaluated in the following manner. First, the best performing individuals from each of the three treatments were evaluated, with both their overall progress forward and position over time logged. We then took the best controllers from (a) runs for the original undamaged morphology, (b) the unmodified EEA, and (c) the *Out-of-Band Inference* EEA, and executed them on the target, damaged morphology. These three runs allow us to evaluate the relative performance of solutions developed using the two EEA-based algorithms, when compared to the original target morphology.

Trajectories for evolved controllers are depicted in Figure 12. The original controller evolved for the undamaged robot swims effectively when paired with the undamaged morphology, but does not do well when executed on the damaged morphology, as seen in Figure 12(a). Such a situation might occur under normal operating conditions, where damage and wear are likely to alter the morphology of an individual. Since the controller was developed for a known morphology, coordination between the pectoral flippers and caudal fin break down when the morphology changes. This breakdown is apparent in Figure 12(a), where the damaged system with the original controller stalls and drifts.

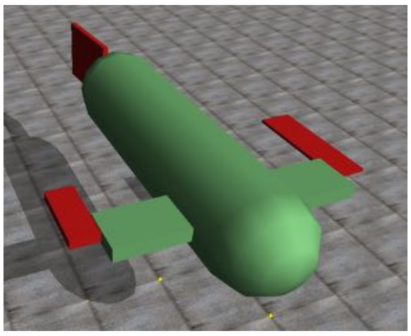
In contrast, solutions attained through self-modeling attempt to account for changes in the morphology through evolution of new models. Figure 12(b) plots the tragectories using the original EEA approach. Although the evolved controller does well when applied to the EEA-generated model (its "native" morphology), when applied to the actual target it fails to attain even the performance of the original controller. This follows, given that the model shown in Figure 11(b) does not accurately reflect the damage. When the con-



(a)

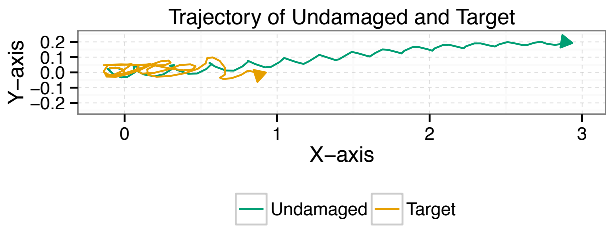


(b)

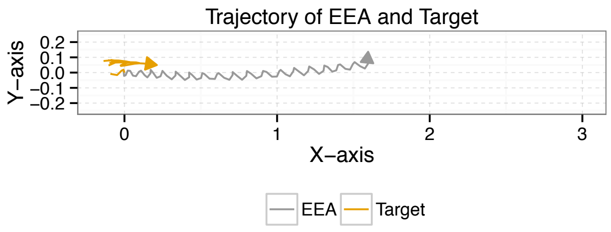


(c)

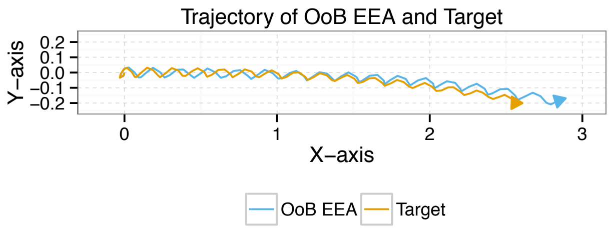
Figure 11. Simulated robots from the case study: (a) damaged target robot with a pectoral fin half the length of a normal one; (b) morphological model produced using the original EEA approach; (c) morphological model produced by the *Out-of-Band Inference* EEA approach.



(a)



(b)



(c)

Figure 12. Trajectories of the evolved solutions from EEA runs and their performance when used to control the damaged target robot: (a) controller evolved for the original undamaged robot, when evaluated on both original and the target; (b) controller evolved using the morphology found by unmodified EEA, evaluated on both the EEA model and the target; (c) controller evolved using the morphology found by the *Out-of-Band Inference* EEA, evaluated on both the OOB-EEA model and the target.

troller is transferred into the damaged morphology, the robot flounders about the starting point while making no little forward progress. Using the *Out-Of-Band Inference* EEA, however, results of the transfer improve noticeably, as shown in Figure 12(c). The controller operates effectively both in its native morphology and in the morphology of the target; both are able to swim forward with only a small advantage for the native morphology.

Results of these validation runs demonstrate both the effectiveness of the EEA approach, as well as the importance of reducing uncertainty as part of the process, at least for certain contexts. While the original EEA approach was not able to arrive at an adequate internal representation of the target, the *Out-of-Band Inference* approach was able to build a fair internal representation, enabling evolution of an effective swimming behavior for the damaged robot.

6 Conclusions

Robotic and cyber-physical systems of the future need to provide high levels of resilience and adaptability. Self modeling represents a key technology in reaching that goal, enabling systems to compensate at run time for unanticipated scenarios. Although a solid foundation of work has been laid in this area, several outstanding issues have received relatively little attention. Among these issues is dealing with uncertainty in the modeling process; uncertainty can stem from the environment, or in the case of our study, limitations of the robot itself.

We have developed and studied two techniques for reducing the negative impact of the unknowns by inferring them during the modeling process. We found that although these algorithms cannot completely overcome the lack of sensor information, much of the uncertainty can be removed. In our application, the aquatic robot was able to infer information about its initial configuration, substantially improving the performance of its model estimations.

These inference algorithms may be applicable to other domains within the EEA self-modeling framework. Although we investigated only the inference of missing servo motor information, the same techniques could be applied to deduce environmental factors, such as water flow in an aquatic environment, or the terrain slope for a terrestrial robot. By inferring the unknown values on a per-experiment basis we have avoided making any assumptions about these values remaining static over time. In addition, the inferred values do not need to be encoded in the model's genome, reducing the complexity of the model search space.

Acknowledgments

The authors gratefully acknowledge the contributions and feedback provided by Professor Xiaobo Tan, members of the Software Engineering and Network Systems Laboratory, the Smart Microsystems Laboratory, and the BEACON Center at Michigan State University. This work was supported in part by National Science Foundation grants CNS-1059373, CNS-0915855, DBI-0939454, CCF-0820220 and CNS-0751155.

REFERENCES

[1] Awards for human-competitive results produced by genetic and evolutionary computation. Competition held as part of the annual Genetic and Evolutionary Computation Conference (GECCO), sponsored by ACM SIGEVO. Results available at http://www.human-competitive.org.

[2] *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, eds., Ajith Abraham and Robert Goldberg, Springer, 2005.

[3] Joshua E. Auerbach and Josh C. Bongard, 'Evolution of functional specialization in a morphologically homogeneous robot', in *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference (GECCO)*, pp. 89–96, Montreal, Quebec, Canada, (2009). ACM.

[4] Gordon Blair, Geoff Coulson, and Nigel Davies, 'Adaptive middleware for mobile multimedia applications', in *Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 259–273, (1997).

[5] J.C. Bongard and H. Lipson, 'Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials', in *Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on*, pp. 169–176, (2004).

[6] Josh Bongard, Victor Zykov, and Hod Lipson, 'Automated synthesis of body schema using multiple sensor modalities', in *In Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems (ALIFEX*, pp. 220–226, (2006).

[7] Josh C. Bongard and Hod Lipson, 'Automated damage diagnosis and recovery for remote robotics', in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pp. 3545–3550, New Orleans, Louisiana, (May 2004).

[8] Josh C. Bongard and Hod Lipson, 'Automating genetic network inference with minimal physical experimentation using coevolution', in *In Proceedings of The 2004 Genetic and Evolutionary Computation Conference*, pp. 333–345. Springer, (2004).

[9] Josh C Bongard and Hod Lipson, 'Once more unto the breach: Coevolving a robot and its simulator', in *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems*, pp. 57–62, Boston, Massachusetts, USA, (2004).

[10] Rodney A. Brooks, 'Artificial life and real robots', in *Proceedings of the First European Conference on Artificial Life*, pp. 3–10. MIT Press, Cambridge, MA, (1992).

[11] Betty H. C. Cheng, Andres J. Ramirez, and Philip K. McKinley, 'Harnessing evolutionary computation to enable dynamically adaptive systems to manage uncertainty', in *Proceedings of the First International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE)*, San Francisco, California, USA, (May 2013).

[12] Kenneth A. De Jong, *Evolutionary Computation: A Unified Approach*, MIT Press, 2002.

[13] F. C. Dyer and H. J. Brockmann, 'Biology of the Umwelt', in *Foundations of Animal Behavior*, eds., L. D. Houck and L. C. Drickamer, 529–538, University of Chicago Press, (1996).

[14] Dario Floreano, Phil Husbands, and Stefano Nolfi, 'Evolutionary Robotics', in *Handbook of Robotics*, Springer Verlag, Berlin, (2008).

[15] John H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, University of Michigan Press, 1975.

[16] Nick Jakobi, 'Running across the reality gap: Octopod locomotion evolved in a minimal simulation', in *Proceedings of the First European Workshop on Evolutionary Robotics*, pp. 39–58, Paris, France, (1998). Springer-Verlag.

[17] Jeffrey O. Kephart and David M. Chess, 'The vision of autonomic computing', *IEEE Computer*, **36**(1), 41–50, (2003).

[18] Sylvain Koos, Jean Baptiste Mouret, and Stéphane Doncieux, 'Crossing the reality gap in evolutionary robotics by promoting transferable controllers', in *Proceedings of the 2010 ACM Genetic and Evolutionary Computation Conference*, pp. 119–126, New York, New York, USA, (2010). ACM.

[19] Hod Lipson, 'Evolutionary robotics and open-ended design automation', in *Biomimetics*, ed., Bar Cohen, 129–155, CRC Press, (2005).

[20] Hod Lipson and Josh Bongard, 'An exploration-estimation algorithm for synthesis and analysis of engineering systems using minimal physical testing', in *Proceedings of the 2004 ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2004)*, pp. 1087–1093, Salt Lake City, Utah, (2004).

[21] Pattie Maes, 'Concepts and experiments in computational reflection', in *Proceedings of the ACM Conference on Object-Oriented Languages (OOPSLA)*, pp. 147–155. ACM Press, (December 1987).

[22] S. H. Mahdavi and P. J. Bentley, 'An evolutionary approach to damage recovery of robot motion with muscles', in *Proceedings of the Seventh European Conference on Artificial Life*, pp. 248–255, Dortmund, Germany, (September 2004).

[23] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, 'Composing adaptive software', *IEEE Computer*, **37**(7), 56–64, (2004).

[24] Jared M. Moore, Anthony J. Clark, and Philip K. McKinley, 'Evolution of station keeping as a response to flows in an aquatic robot', in *Proceedings of the 2013 ACM Genetic and Evolutionary Computing Conference*, pp. 239–246, Amsterdam, The Netherlands, (2013).

[25] R. Smith. Open Dynamics Engine. Manual and source code available online at: http://www.ode.org.

[26] Andres J. Ramirez, Betty H. C. Cheng, Philip K. McKinley, and Benjamin E. Beckmann, 'Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration', in *Proceedings of the 7th International Conference on Autonomic Computing*, pp. 225–234, Washington, DC, (June 2010).

[27] Kenneth O. Stanley and Risto Miikkulainen, 'Evolving neural networks through augmenting topologies', *Evolutionary Computation*, **10**(2), 99–127, (June 2002).

[28] Kenneth O. Stanley and Risto Miikkulainen, 'Competitive coevolution through evolutionary complexification', *Journal of Artificial Intelligence Research*, **21**, 63–100, (2004).

[29] Daniel M Wolpert, R Chris Miall, and Mitsuo Kawato, 'Internal models in the cerebellum', *Trends in cognitive sciences*, **2**(9), 338–347, (1998).

[30] Ji Zhang and Betty H. C. Cheng, 'Model-based develpment of dynamically adaptive software', in *IEEE International Conference on Software Engineering (ICSE06)*, Shanghai, China, (May 2006). IEEE.