Exploiting Multiple Robots to Accelerate Self-Modeling

Josh Bongard
Department of Computer Science
University of Vermont
33 Colchester Ave., Burlington, VT 05405
josh.bongard@uvm.edu

ABSTRACT

In previous work [8] a computational framework was demonstrated that allows a mobile robot to autonomously evolve models its own body for the purposes of adaptive behavior generation or recovery from damage. Conceivably, robots working in tandem could share their experiences such that one robot, when faced with a situation already encountered by another robot, could draw on that experience and adapt more rapidly. A first demonstration of this is given here: multiple robots with the same or similar body plan, but acting independently, combine self-models such that they accelerate modeling. Two approaches are investigated: the robots feed their experiences back into a common modeling engine, or they maintain their own modeling engine but share their best self-models with each other. It was found that the latter approach achieves a significant improvement in modeling compared to a single robot and compared to the former approach. This finding has implications for how to design autonomous robots acting in concert to achieve large-scale tasks.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*

General Terms

Algorithms

Keywords

Evolutionary robotics, self-modeling, artificial intelligence

1. INTRODUCTION

Industrial robots have permeated and revolutionized every aspect of heavy industry because they can execute preprogrammed actions in fixed, indoor industrial environments. Robots would be equally useful in outdoor or home environments, but creating devices that can continuously adapt and autonomously cope with the constantly changing aspects of such environments has had limited success. Rather than continuously having to re-program new controllers for a robot once it or its environment changes, evolutionary robotics [27] is a field that uses evolutionary computation to autonomously generate behaviors for robots. There are three main approaches to evolutionary robotics: controllers are either evolved directly on the physical device, requiring thousands of evaluations [12, 14]; controllers are adapted from an existing, hand-designed controller [31]; or a hand-designed simulator is used to evolve controllers before transferal to the physical device [20, 28]. The first approach is infeasible for continuous, rapid adaptation; the second approach requires a human to create the starting behavior; and the third approach requires a human to craft a simulation of the robot.

In previous work [7, 8] we introduced a fourth method that overcomes these obstacles by allowing the robot to evolve simulations of itself and its local surroundings, and then use the best of the evolved simulations to internally rehearse behaviors before attempting them in reality. Rather than most evolutionary computation-based modeling approaches in which a set of training data is generated first and then models are evolved to explain that data (eg. [24, 1, 16]), the framework developed in [7, 8] uses an active learning [2] approach: modeling alternates with a search for new training data, based on the current state of the models.

This raises the question of how to search for new training data. Seung *et al.* [29] showed that in theory, the optimal choice for the next training data is the one which causes the current set of models to disagree in their predictions. In previous work we introduced the **estimation-exploration algorithm** [7], or **EEA**, which uses an evolutionary algorithm to search for these informative training samples: a fitness function rewards candidate training data for how much model disagreement it causes. A second evolutionary algorithm optimizes a set of models against the current set of training data evaluated by the target system being modeled. We have applied the EEA to problems in machine learning [5], gene network identification [10], damage localization in truss structures [23], and to robotics [4, 9, 8].

In the robotics application we showed that a robot could use this technique to diagnose and recover from body damage [8] by inferring the structure of its own body (or changes in its body structure as a result of damage). While much progress has been made allowing robots to model their environment [32], relatively little is known about how a robot can learn its own morphology, which cannot be inferred by

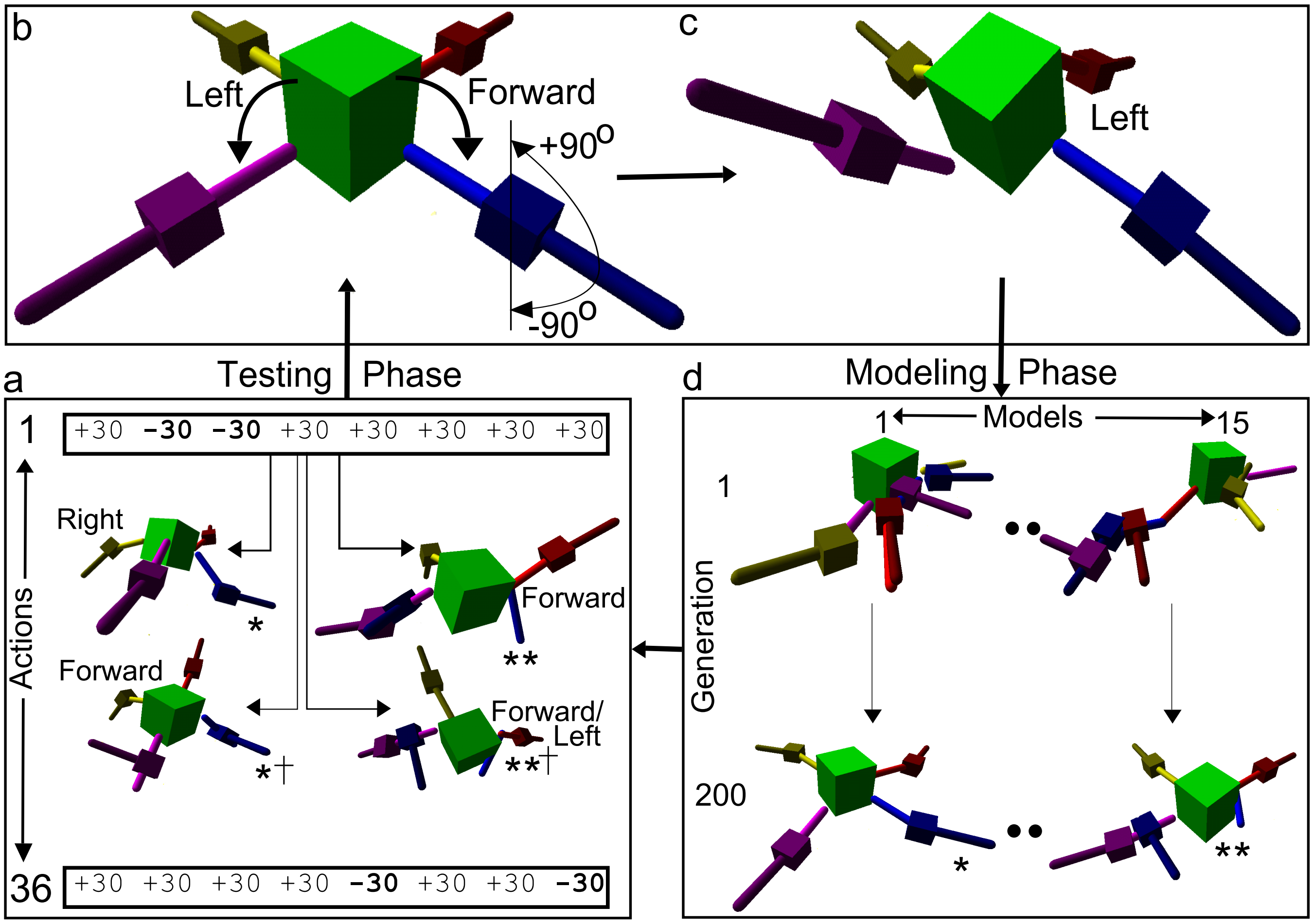

Figure 1: The EEA applied to a quadrupedal robot. The algorithm begins by supplying a random action (1. in a) to the target robot (b). The action causes the robot to move, and its main body tilts (c). The tilt information is bundled with the action that caused it, and supplied to the modeling phase. The robot meanwhile relaxes back to the default flat position. In the modeling phase (d), a set of 15 models are evolved to explain all the tilt information extracted from the target robot so far. These evolved models are then used to search for a new action (a). This process is repeated for a set number of cycles.

direct observation or retrieved from a database of past experiences [22]. Self-modeling would be useful when a robot operates in remote or dangerous environments, and would therefore have to adapt its behavior to unforeseen circumstances autonomously. Other approaches to robot damage recovery have focussed on built-in redundancy [33, 11] or contingency plans designed for anticipated failures [34]. However, self-modeling requires the potentially damaged robot to perform behaviors to ascertain how it is damaged, and use those experiences to internally model itself, which takes time.

In this paper it is shown how multiple, independent robots with the same body plans can accelerate self-modeling by sharing their experiences. Robot teams is an area of intense research. Collective robotics [25, 15] is concerned with robots working together on some collective task. Although robot teams have been used for collaborative modeling, it has been restricted to external modeling: for example creating global maps [13, 26] or estimating object positions [30].

The next section describes the algorithm as applied to a single robot, as well as two alternative approaches for combining the experiences of multiple, independently-acting robots. Section 3 provides results indicating how these two approaches fare against the algorithm applied to a single robot, and against each other. Section 4 provides some discussion and concluding remarks.

2. METHODS

The EEA, applied to a robotics application, is outlined in Fig. 1. In this application the algorithm attempts to discover the correct topology of a simulated, four-legged robot, without direct information about how the robot's body parts are connected together: the robot cannot see its own body with a camera, nor can it sense at its joints which parts are connected there. Previously [8] we showed that this framework allows a physical robot to automatically construct a simulation of itself, use that simulation to internally rehearse behaviors, and automatically diagnose and recover from damage. The framework is composed of three components: the robot to be modeled (Fig. 1b,c), a set of self-models (Fig. 1d), and a set of candidate actions that can be executed on the robot (Fig. 1a).

2.1 The robot

The **target robot** used here is a simulated, four-legged robot (Fig. 1b). The robot is composed of nine body parts: the main body, and four upper and lower legs (Fig. 2a). The eight leg parts are motorized, and can rotate through the vertical plane that lies parallel to the leg. Positive rotations cause the part to rotate upward, and negative rotations downward. The robot is simulated within a three-dimensional, physics-based environment[1]. A training data in this application is an **action**: a set of eight desired motor rotations, expressed in degrees (Fig. 1a), that causes all eight motors to rotate, sending the robot to a fixed position (Fig. 1c). In this position, two tilt sensors record how much the robot's main body tilts to the left or right (-90^o to $+90^o$), and how much forward or backward (-90^o to $+90^o$). These two angles together form the **result** generated by the target robot. The robot then rotates the motors back to 0^o, causing the robot to lie flat again. The result is bundled with the action that caused them, and sent to the modeling phase (Fig 1d).

2.2 The models

In the modeling phase (Fig. 1d), a set of 15 **model robots** are evolved using a parallel hill climber to infer the way in which the robot's body parts are attached together. The training set is composed of the set of action/result pairs that have been obtained from the target robot so far. On the first cycle through the algorithm, the modeling phase has one pair; on the second cycle through it has two action/result pairs; and so on. The algorithm is assumed to know: how many motorized parts there are (eight); the mass and geometry of each part; parts are attached perpendicularly to each other; each body part is horizontal; and that actuating a body part with a positive angle will cause it rotate upward by that amount, and a negative angle downward by that amount. In future work these constraints will gradually be removed. The algorithm must indirectly infer how the parts are connected using only the tilt information in the training data.

A model is encoded as a 8×2 genotype G with floating-point values in the range $[0.0, 1.0]$, and dictates how the nine known body parts (Fig. 2a) should be connected to one another to produce a phenotype. The phenotype is a three-dimensional, physically realistic simulated robot, like those shown in Fig. 1d. The genotype is translated into a phenotype as follows. Each row in the matrix corresponds to one of the eight body parts. For each of the $i = 1, 2, \ldots, 8$ body parts, entry $G(i, 1)$ is scaled to an integer value in $[0, i-1]$. This indicates to which body part the current part attaches. A value of 0 indicates the part attaches to the main body; a value of 1 indicates it attaches to motorized body part 1; and so on. In the example shown in Fig. 2b, part 1 attaches to part 0 ($G(1, 1) = 0$), and part 5 to part 1 ($G(5, 1) = 1$). The second value in the row, $G(i, 2)$, indicates where on the periphery of the parental body part the current part should be attached. A value of $G(i, 2) = 0$ indicates that part i should connect to the upper left of the parent body part; larger values attach the part at further

[1]ode.org

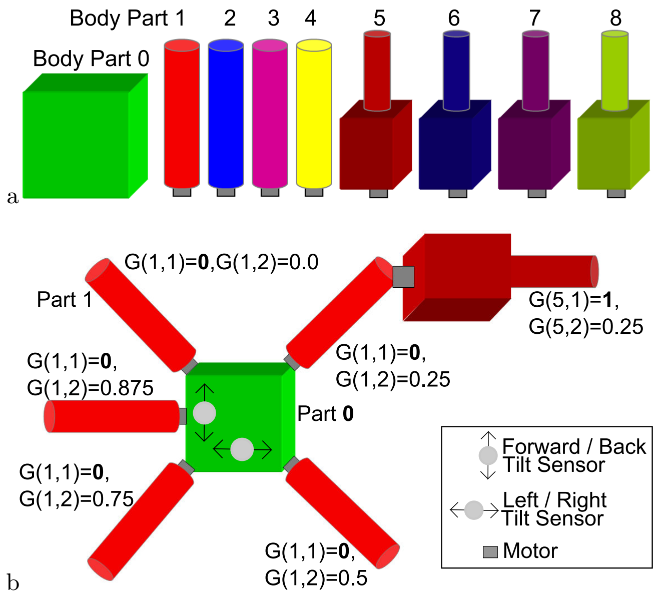

Figure 2: Genotype to phenotype translation for the self-models. The algorithm begins by knowing how many motorized parts the robot is composed of (a). A model genotype G encodes information for connecting the parts together. $G(i, 1)$ indicates to which body part i should attach. $G(i, 2)$ indicates where on the periphery it should attach.

positions around the periphery of the parent part, proceeding in a clockwise direction. In the example shown in Fig. 2b, body part 1 attaches to the upper-right of the main body (G(1,2)=0.25), and then body part 5 attaches in turn to the upper-right of body part 1 (G(5,2)=0.25).

Once a model robot is formed, it is actuated with each of the actions in the training data that have already been executed on the target robot. For each action, the resulting tilt of the model robot's main body is recorded. The **subjective error** of a model is then given by

$$m_e = \frac{\sum_{i=1}^{k}(|t_{lr}^{(i)} - m_{lr}^{(i)}| + |t_{fb}^{(i)} - m_{fb}^{(i)}|)}{2k}, \quad (1)$$

where k is the total number of actions that have been performed by the target robot so far; $t_{lr}^{(i)}$ is the amount the target robot tilted to the left or right when it executed action i; $m_{lr}^{(i)}$ is the amount the model robot tilted to the left or right when it executed action i; $t_{fb}^{(i)}$ is the amount the target robot tilted forward or backward when it executed action i; and $m_{fb}^{(i)}$ is the amount the model robot tilted to the left or right when it executed action i. In short, the accuracy of a model is how well it reproduces the behaviors of the target robot when supplied with the same actions.

A hill climber then optimizes each of the 15 models in an attempt to minimize m_e. Once a model has been evaluated, its genotype is copied, a single value in the matrix is chosen at random, and Gaussian mutation is applied. A new model is created from this matrix, and evaluated. If the new model achieves a lower error than the parent model, the parent genotype is discarded; otherwise, the child is discarded. This process is continued for 200 generations, for each of the 15 models. These optimized models are then passed to the testing phase (Fig. 1a) for finding a new action.

On the second and subsequent cycles through the modeling phase, hillclimbing begins with the best models from the previous cycle, but the models are re-evaluated against the larger training set, which contains the original action/result pairs plus the new pair just obtained from the target robot.

2.3 The actions

The testing phase attempts to find a new action that, when executed by the target robot, will provide more information about the robot's topology. At the outset of a trial, 36 random actions are generated, but not evaluated on the target robot (Fig. 1a). Initially, one of the 36 actions is selected at random and sent to the target robot to generate the first training data. On the second and subsequent cycles, the optimized models from the modeling phase are used to determine which new action to send to the target robot. Each action is supplied to the fifteen models, and the fitness of an action (a_f) is computed using

$$\frac{\sigma^2(\mathbf{m}_{lr}) + \sigma^2(\mathbf{m}_{fb})}{2} - \frac{\sum_{i=1}^{15} |m_{lr}^{(i)} - m_{lr}^{'(i)}| + |m_{fb}^{(i)} - m_{fb}^{'(i)}|}{30}, \quad (2)$$

where $\sigma^2(\mathbf{m}_{lr})$ is the variance across the left and right tilting of the 15 optimized models when supplied with action a, and $\sigma^2(\mathbf{m}_{fb})$ is the variance across the forward and backward tilting of the 15 optimized models. In short, the first term rewards an action for how much it causes the optimized models to tilt in different directions. This reflects the theoretical finding from active learning in which it was shown [29] that the best way to choose a new piece of training data to be evaluated by the target system (in this case, the simulated robot) is the one that causes the models to disagree in their predictions about this training data. Once this training data is evaluated by the system being modeled and added to the training set, now only some of the models, not all, will agree with the results from the system, because the models disagree about the new training data. Further modeling can then replace these recently-revealed erroneous models with new models that explain all the old data, plus the new training data. This has the effect of altering the fitness landscape of the self-models to allow for further evolution, a common and desirable property of co-evolutionary algorithms [18, 21]. The EEA is a co-evolutionary algorithm in the sense that actions try to disprove self-models, and self-models attempt to explain actions.

The second term penalizes potentially 'dangerous' actions. 'Dangerous' actions are those which may cause the target robot to behave very differently from the self-models, even though the self-models very accurate and therefore topologically similar to the target robot. For example, if an action causes the target robot to balance on its front and back legs, it may fall to the left, while accurate models may fall to the right simply because of slight topological differences. These 'dangerous' actions have the effect of making the model search space very rugged: accurate models suddenly experience a misleading increase in error. 'Dangerous' actions are avoided by taking the current model set and producing mutants of them: a copy of a model is made, and then topologically perturbed slightly. For example, in Fig. 1a, model $*$ tilts to the right, while its mutant $*\dagger$ tilts forward. The more that the mutants disagree with their models, the more likely the target will disagree with the models as well. For a more detailed treatment of this phenomenon, see [3]. In short, this fitness functions selects both for informativeness (the first term) and against potential danger (the second term).

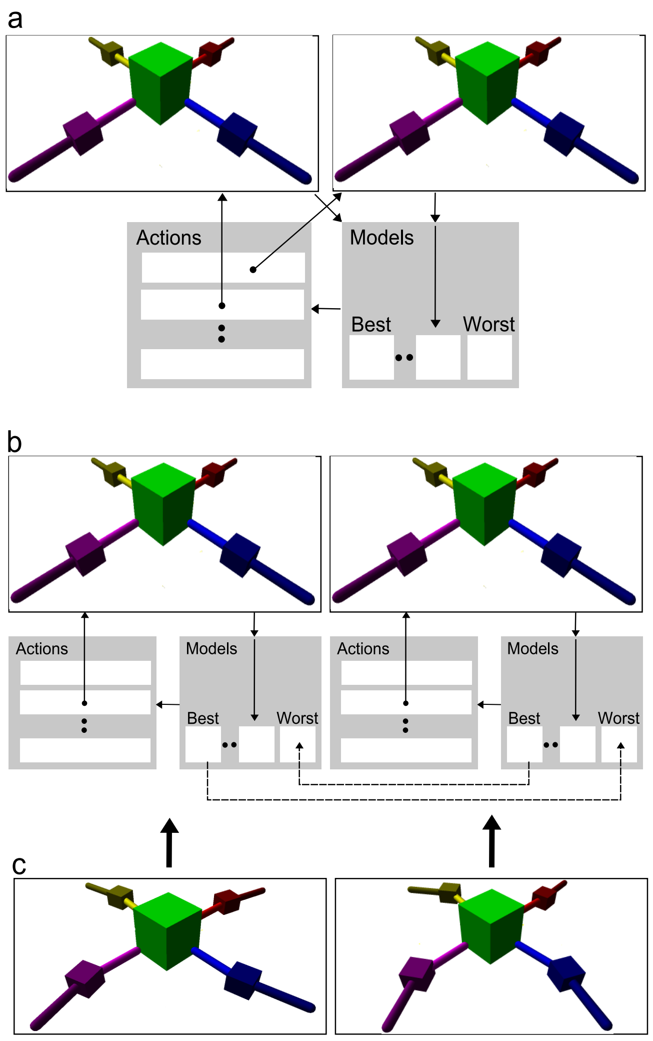

Figure 3: Alternative approaches for exploiting two or more robots for self-modeling. In the *Combined* approach (a), two robots each execute different actions, and then feed those two actions (along with their results) into a common EEA. In the *Swap* approach (b), each robot maintains its own EEA, but they swap their current best self-models. *SwapCrooked* uses the same regime as *Swap*, but the two communicating robots are slightly different (c).

Of the 36 actions, the one with maximal a_f (and which has not yet been executed) is sent to the target robot for execution. It is then bundled with the resulting tilt information, and passed to the modeling phase, where the number of training data is incremented from k to $k+1$. Once the modeling phase has been cycled through 16 times, the algorithm halts, and returns the model with the lowest error as its best guess as to the topology of the target robot.

2.4 The EEA on Multiple Robots

Two alternative approaches to parallelizing the EEA across several independently-acting robots is possible: they may share a common EEA (*Combined*), or each robot may execute its own EEA and swap self-models between them (*Swap*).

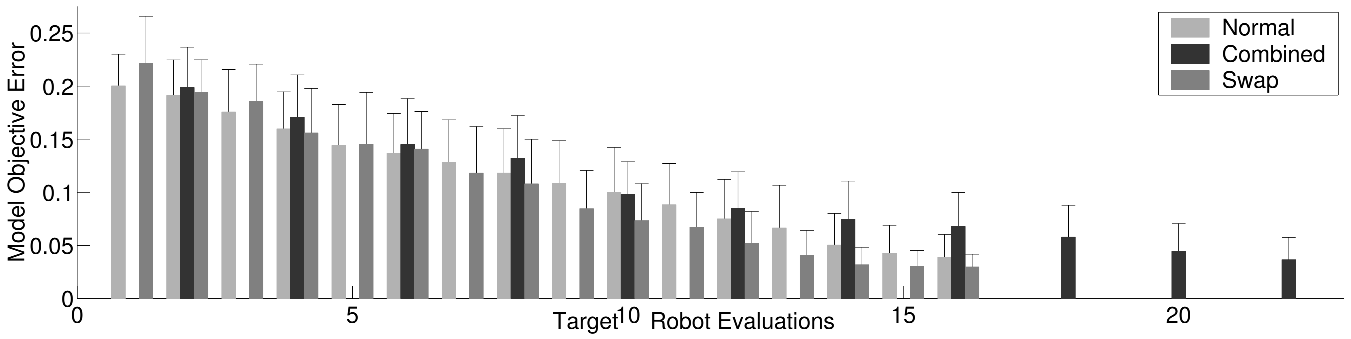

Figure 4: Comparison of modeling ability for a single robot self-modeling (the normal EEA); two robots that swap their best models at the end of each modeling phase (*Swap*); and two robots that feed their experiences into a common EEA (*Combined*). The *Combined* trial accumulates two target robot evaluations per cycle: therefore, the third bar is plotted every two evaluations.

Combined—Fig. 3a outlines the first approach, in which two or more robots share a common EEA. Rather than the testing phase outputting a single action, two actions are output. During the first pass through the testing phase, two actions are chosen at random from the 36 and output; during the second and subsequent passes, the action with highest a_f, and the one with the second-highest a_f are output. The first robot executes the first action, and the second robot executes the second action. The two actions, along with their results, are bundled and sent to the common modeling phase. In the normal EEA, the first pass through the modeling phase optimizes models against 1 action/result pair; during the kth pass, models are optimized against k action/result pairs. In *Combined*, models are first optimized against two action/result pairs; during the kth pass through the modeling phase, they are optimized against $2k$ action/result pairs.

Swap—Fig. 3b outlines the second approach, in which two or more robots maintain their own EEA. However, unlike the normal EEA, when a cycle through the modeling phase is finished, the robot waits for a signal from the other robot (or robots) that it has also just finished a cycle of self-modeling. At this point, they swap best self-models: the first robot makes a copy of its best self-model, and overwrites the worst self-model maintained by the second robot, and vice versa. The robots then continue with the testing phase as usual.

Swap+Crossover—In order to allow the EEA to combine genetic material from the best self-models across different robots, crossover was introduced into the modeling phase of *Swap*, in the hopes that the accurate parts of the best self-models from different EEAs would be combined. Crossover is accomplished as follows. Whenever new child self-models are being created in the modeling phase by the hillclimber, the genotype matrices of the 15 parents are copied and mutated as explained above. With a 50% probability, a child is paired with another child in preparation for crossover. Then, each row i is swapped between the children with a 50% probability.

Swap3—*Swap* (without crossover) was expanded to three robots working together. At the end of each cycle through the modeling phase, each robot receives a copy of the best self-model from each of the other two robots. It then replaces its second-worst self-model with one of these, and its worst self-model with the other.

SwapCrooked—Finally, *Swap* was run again, but instead of using two identical robots, slight random perturbations were introduced into the masses, sizes and positions of the body parts (Fig. 3c). In reality, no two robots in a group are ever exactly the same: slight manufacturing error and different wear-and-tear experienced by the machines causes them to have slightly different morphologies, and therefore behave slightly differently. This regime reflects this reality.

3. RESULTS AND DISCUSSION

Thirty independent trials were performed using the normal EEA, 30 of *Combined*, and 30 of *Swap*. For *Swap*, the first two trials were allowed to communicate with one another; trials 3 and 4 communicated with one another; and so on. For the normal EEA and *Swap*, 16 cycles were allowed (Fig. 1a,b,c and d were cycled through 16 times). *Combined* was only executed for 11 cycles, because more model evaluations are performed per cycle by this algorithm than by the normal EEA or *Swap*: each model in *Combined* must be evaluated $2k$ times during cycle k, whereas each model in the other two algorithms only need be evaluated k times. It was found that when the 11th cycle of self-modeling was completed during a trial of *Combined*, it had performed about the same total number of model evaluations as those performed by the normal EEA and *Swap* after 16 cycles: slightly less than 250,000 evaluations.

For each trial, at the end of each cycle through the modeling phase, the accuracy of the self-model with lowest subjective error was assessed. This is accomplished by computing that self-model's **objective error**, given as

$$m_o = \frac{\sum_{i=1}^{8} \sqrt{(t_x^{(i)} - m_x^{(i)})^2 + (t_z^{(i)} - m_z^{(i)})^2}}{8}, \qquad (3)$$

where $t_x^{(i)}$ is the horizontal position of body part i on the target robot when it lies flat, $m_x^{(i)}$ is the horizontal position of body part i on the model robot, $t_z^{(i)}$ is the z-position of body part i on the target robot, and $m_z^{(i)}$ is the z-position of body part i on the model robot. Therefore, the objective error of a model is the mean Euclidean difference between the positions of the target robot's body parts, and the model robot's body parts. This metric then gives an unbiased indication of how close the model is to the topology of the target robot.

Fig. 4 reports the mean accuracies of the best models produced by the algorithm variants at the end of each modeling phase. As can be seen, *Combined* shows no modeling

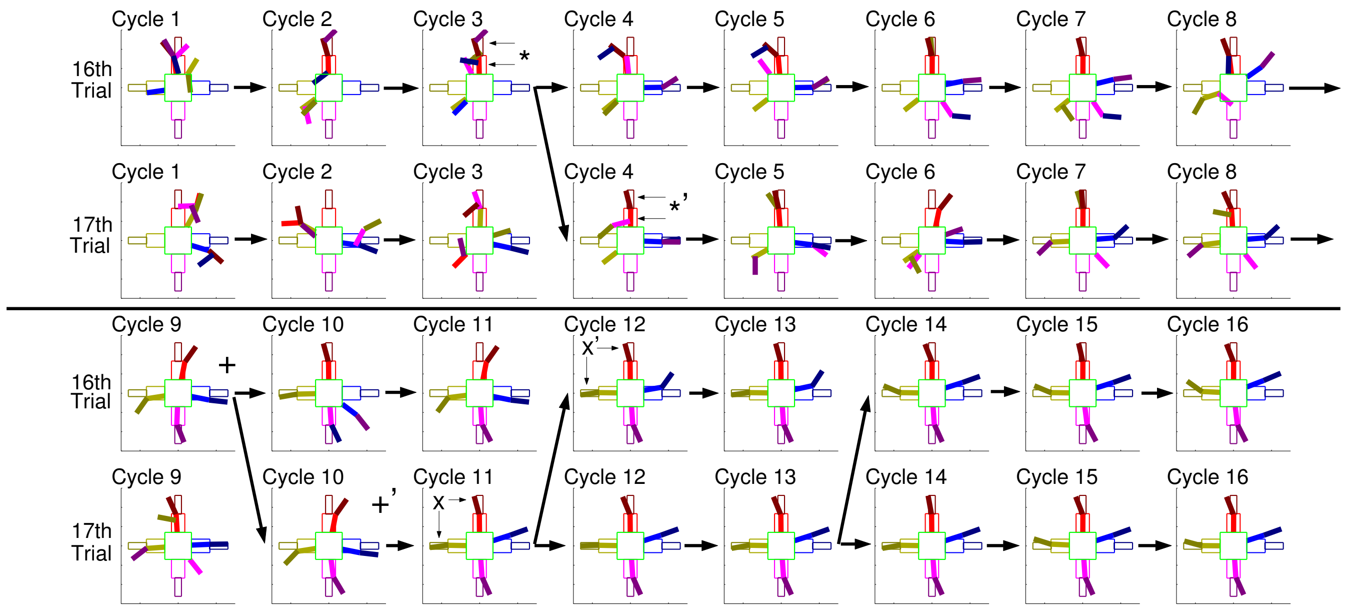

Figure 5: An illustration of two robots successfully sharing self-models. The best models obtained by two robots working together are shown: the model with the lowest subjective error (Eqn. 1) at the end of each cycle through the modeling phase is shown. Diagonal arrows indicate when sharing helped: the best self-model from one robot became the best self-model in the other robot's model population.

improvement over the normal EEA, and indeed its models are less accurate after 16 target robot evaluations have been performed (the second bar is statistically significantly higher than the first in the 16th grouping). In contrast, *Swap* begins to outperform the normal EEA during the 13th cycle (the third bar is statistically significantly lower than the first bar in the 13th grouping onward).

Although the reason why *Combined* performs worse than the normal EEA is not known, it is suspected that the modeling process falters as it accumulates training data faster than it can find models to explain it. This was observed previously for a different application [6]. Regardless, it is clear that between these two approaches to parallelizing self-modeling across robots, allowing the robots to maintain their own EEA is superior to making them share one.

Fig. 5 illustrates how two robots using *Swap* can outperform a single self-modeling robot. In the 16th and 17th inter-communicating trials, the first robot manages to correctly model its two forward body parts during its third cycle of self-modeling (indicated by $*$). This self-model is transferred to the second robot, and becomes that robot's best self-model: the two forward body parts are both modeled correctly in the second robot's best self-model in the next cycle (indicated by $*'$). Later, the first robot develops a topologically correct model during the ninth modeling phase (indicated by $+$), although the body parts are still slightly crooked. This self-model, when transferred, rapidly supplants the best self-model of the second robot (indicated by $+'$). In the 11th modeling cycle the second robot straightens the front and left body parts (indicated by x), and this improvement is transferred successfully to the first robot (indicated by x').

The reason why the second robot is able to improve how it models the forward and left body parts in cycle 11 is due to the fact that it has accumulated more experiences related to those body parts. In other words, those body parts were actuated downward (and thus contributed to the main body's tilt) more often by the second robot than they were by the first robot (data not shown). It is this process—reception of a good self-model, subsequent improvement of that self-model in response to the receiving robot's different experiences, and transferral of the resulting better self-model back to the donating robot—that is hypothesized to the reason why *Swap* outperforms the normal EEA. This improvement stands despite the fact that a robot cannot combine well-modeled body parts from two different self-models into a new, better self-model.

To address this limitation, 30 independent trials of *Swap* with crossover were performed (*Swap+Crossover*). As can be seen in Fig. 6, incorporating crossover into *Swap* actually begins to harm the algorithm's performance, relative to *Swap* without crossover, during the 14th cycle (the third bar is statistically significantly higher than the second bar in groupings 14, 15 and 16). In another paper [3] it is shown that for the normal EEA, crossover reduces the mean error of the model population, but it also reduces the population's variation. This frustrates the testing phase's ability to find informative actions based on model disagreement, because more similar models disagree less. This phenomenon reappears here in *Swap+Crossover*, even though in this set-up, self-models collected from different robots are (potentially) more different from one another than different self-models generated by the same robot. It is assumed that despite this greater initial self-model variation, crossover still dilutes this variation sufficiently during modeling to compromise the testing phase.

Thirty trials with three robots intercommunicating were then performed (*Swap3*). In *Swap3*, the first, second and third trials communicated with each other; the fourth, fifth and sixth trials intercommunicated; and so on. Fig. 6 also

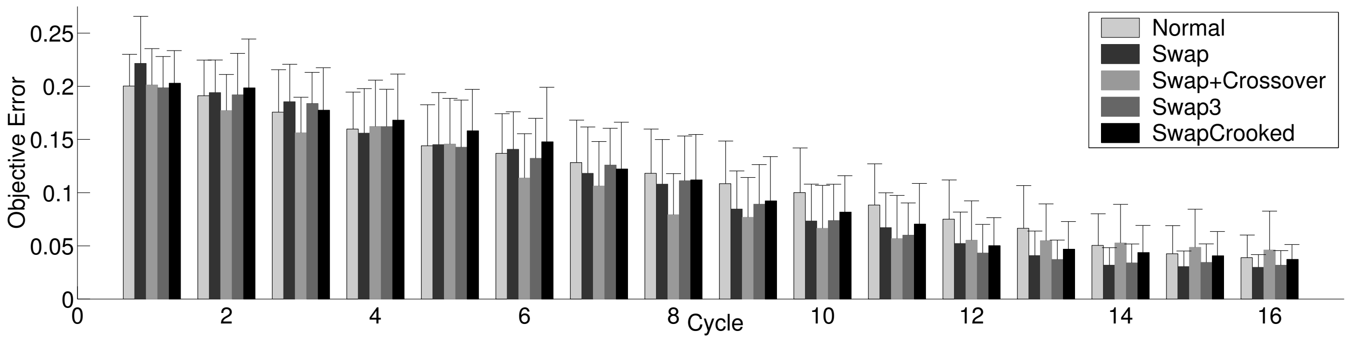

Figure 6: Comparative performance of EEA variants. The mean objective errors (Eqn. 3) of the best models produced by the algorithm variants at the end of each cycle through their modeling phase are reported. The first two bars in each grouping (*Normal* and *Swap*) reproduce the data reported in Fig. 3.

shows that three robots sharing self-models (*Swap3*) do not outperform pairs of robots sharing self-models (*Swap*): the fourth bar in each group is not significantly lower than the second bar. The reason for this is not immediately clear, however it seems likely that the lack of a mechanism for combining good genetic material from the different robots into a single self-model is to blame.

Finally, 30 trials of *SwapCrooked* were performed. In the first and second trial, two slightly different robots traded self-models; in the third and fourth trial, another two slightly different robots traded self-models; and so on. Fig. 6 indicates that even if two slightly different robots share self-models (*SwapCrooked*), they start to produce more accurate models on average during the 10th cycle onward than one robot acting in isolation (*Normal*): the fifth bar is statistically significantly lower than the first bar during cycles 10, 12 and 13, and slightly lower in cycles 11, and 14 onward.

4. CONCLUSIONS

Here is has been shown that it is advantageous to combine experiences from mobile robots attempting to infer their own morphologies. It was shown that a parallelized framework in which two robots maintain separate modeling engines, but share copies of their best self-models with one another, performed better than a single robot modeling alone. Moreover, it performed better than an alternative parallelized framework, in which experiences from pairs of robots were fed into a common modeling engine. This is additionally desirable because in this former approach the robots can be more autonomous: not only do they behave independently, but they model themselves independently as well.

It was also shown that even if the two robots sharing self-models are slightly different morphologically—which would be true for any physical robots—they can still help one another with the modeling process, thereby converging on good self-models more quickly than a single robot operating alone.

Finally, it was found that allowing robots to combine genetic information from different self-models does not accelerate modeling, and in fact slows it. This agrees with a similar result that was found for a single robot attempting to model its own body [3]. This inability to combine genetic material from different self-models also disbars a robot from combining genetic material from two self-models originating from different robots. Nonetheless, a robot may receive a better self-model from another robot, improve that self-model in light of its own experiences, and return the improved self-model to the donating robot. In future work alternative crossover strategies will be investigated which transfer genetic information between self-models from different robots, but maintain diversity across the model set of a single robot.

Despite the improved modeling obtained by allowing two robots to work together, when the same framework was applied to three robots working together, they did not model their own bodies any better than the robot pairs. This indicates that further improvements to the framework are required such that the ability of the robots to self-model themselves when working as a team scales with the number of team members.

Although robot teams working toward a common goal has been studied intensively, focus has been on enabling the robots to collect global information by exploring their environment [13, 26] or collectively manipulating an object that could not be manipulated by a single robot [19, 17]. This work points to a new way in which robots can help one another: they can share their experiences by sharing self-models. It is interesting to note that biological agents can only share experiences indirectly and slowly through genetic inheritance, imitation, or language: the robots demonstrated here share experience directly by trading self-models. Previously we have shown [8] that self-modeling can enable a robot to adapt to unforeseen situations such as body damage. By sharing experiences in order to accelerate self-modeling, one robot may be able to adapt to a new situation more rapidly if it draws on the experiences of a second robot that has already encountered that situation. In this way, the ability of a robot team to continuously adapt to a changing environment could be increased simply by enlarging the size of the team.

5. REFERENCES

[1] H. Andrew. System identification using genetic programming. In *Proceedings of the Second Intl. Conf. on Adaptive Computing in Engineering Design and Control*, pages 57–62, 1996.

[2] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.

[3] J. Bongard. Action-selection and crossover strategies for self-modeling machines. *Proceedings of The Genetic and Evolutionary Computation Conference (GECCO 2007)*, 2007. to appear.

[4] J. Bongard and H. Lipson. Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pages 57–62, 2004.

[5] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6(Oct):1651–1678, 2005.

[6] J. Bongard and H. Lipson. 'Managed challenge' alleviates disengagement in co-evolutionary system identification. In *Proceedings of the 2005 Genetic and Evolutionary Computation Conference*, pages 531–538, Washington DC, 2005.

[7] J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

[8] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314:1118–1121, 2006.

[9] J. C. Bongard and H. Lipson. Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials. In *Proceedings of The 2004 NASA/DoD Conference on Evolvable Hardware*, pages 169–176, Seattle, WA, 2004.

[10] J. C. Bongard and H. Lipson. Automating genetic network inference with minimal physical experimentation using coevolution. In *Proceedings of The 2004 Genetic and Evolutionary Computation Conference*, Seattle, WA, 2004.

[11] F. Caccavale, L. Villani, and P. Ax, editors. *Fault Diagnosis and Fault Tolerance for Mechatronic Systems*, New York, 2002. Springer Verlag.

[12] D. Cliff, P. Husbands, and I. Harvey. Evolving visually guided robots. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the Second International Conference on the Simulation of Adaptive Behaviour*, Boston, MA, 1993. MIT Press.

[13] M. Di Marco, A. Garulli, A. Giannitrapani, and A. Vicino. Simultaneous localization and map building for a team of cooperating robots: A set membership approach. *IEEE Transactions on Robotics and Automation*, 19:238–249, 2003.

[14] D. Floreano and F. Mondada. Hardware solutions for evolutionary robotics. In P. Husbands and J.-A. Meyer, editors, *EvoRobots*, pages 137–151, 1998.

[15] M. Gini and R. Voyles, editors. *Distributed Autonomous Robotic Systems 7*, New York, 2006. Springer.

[16] G. Gray, D. Murray-Smith, Y. Li, K. Sharman, and T. Weinbrenner. Nonlinear model structure identification using genetic programming. *Control Engineering Practice*, 6:1341–1352, 1998.

[17] R. Groß and M. Dorigo. Cooperative transport of objects of different shapes and sizes. *Proceedings of ANTS*, pages 107–118, 2004.

[18] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[19] A. Ijspeert, A. Martinoli, A. Billard, and L. Gambardella. Collaboration Through the Exploitation of Local Interactions in Autonomous Collective Robotics: The Stick Pulling Experiment. *Autonomous Robots*, 11(2):149–171, 2001.

[20] N. Jakobi. Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6(1):131–174, 1997.

[21] E. D. Jong and J. Pollack. Ideal evaluation from coevolution. *Evolutionary Computation*, 12(2):159–192, 2004.

[22] D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi. Online evolution for a self-adapting robotics navigation system using evolvable hardware. *Artificial Life*, 4:359–393, 1998.

[23] B. Kouchmeshky, W. Aquino, H. Lipson, and J. C. Bongard. Coevolutionary strategy for structural damage identification using minimal physical testing. *International Journal for Numerical Methods in Engineering*, 69(5):1085–1107, 2006.

[24] J. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Boston, MA, 1992.

[25] C. Kube and E. Bonabeau. Cooperative transport by ants and robots. *Robotics and Autonomous Systems*, 30(1-2):85–101, 2000.

[26] R. Madhavan, K. Fregene, and L. Parker. Distributed Cooperative Outdoor Multirobot Localization and Mapping. *Autonomous Robots*, 17(1):23–39, 2004.

[27] S. Nolfi and D. Floreano. *Evolutionary Robotics*. MIT Press, Boston, MA, 2000.

[28] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson. Evolutionary techniques in physical robotics. In J. Miller, editor, *Evolvable Systems: from biology to hardware*, pages 175–186. Springer-Verlag, 2000.

[29] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294, New York: ACM Press, 1992.

[30] A. W. Stroupe, M. C. Martin, and T. Balch. Distributed sensor fusion for object position estimation by multi-robot systems. *IEEE International Conference on Robotics and Automation*, 2:1092–1098, 2001.

[31] R. Tedrake, T. Zhang, and H. Seung. Learning to walk in 20 minutes. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, Yale University, New Haven, CT, 2005.

[32] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[33] M. Visinsky, J. Cavallaro, and I. Walker. Robotic fault detection and fault tolerance: A survey. *Reliability Engineering and System Safety*, 46:139–158, 1994.

[34] S. Zilberstein, R. Washington, D. S. Benstein, and A.-I. Mouaddib. Decision-theoretic control of planetary rovers. *Lecture Notes in Computer Science*, 2466:270–290, 2002.