Accelerating Self-Modeling in Cooperative Robot Teams

Josh C. Bongard

***Abstract*—One of the major obstacles to achieving robots capable of operating in real-world environments is enabling them to cope with a continuous stream of unanticipated situations. In previous work, it was demonstrated that a robot can autonomously generate self-models, and use those self-models to diagnose unanticipated morphological change such as damage. In this paper, it is shown that multiple physical quadrupedal robots with similar morphologies can share self-models in order to accelerate modeling. Further, it is demonstrated that quadrupedal robots which maintain separate self-modeling algorithms but swap self-models perform better than quadrupedal robots that rely on a shared self-modeling algorithm. This finding points the way toward more robust robot teams: a robot can diagnose and recover from unanticipated situations faster by drawing on the previous experiences of the other robots.**

***Index Terms*—Collective robotics, evolutionary robotics, self-modeling.**

I. 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—or the effects such environments have on them—has had limited success.

Rather than continuously having to reprogram new controllers for a robot once it or its environment changes, evolutionary robotics [28] 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 [36]; or a hand-designed simulator is used to evolve controllers before transferal to the physical device [20], [31]. 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. This detracts from the overall goal of evolutionary robotics, which is to realize an automated mechanism for behavior generation.

In previous work [7], [9], a fourth method was introduced 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. Much work has been conducted on enabling a robot to autonomously model its environment [37], but our work is the only method proposed thus far that enables a robot to explicitly model its own body. This approach can be used by a robot to diagnose and recover from body damage when such damage cannot be inferred by direct observation or retrieved from a database of past experiences [21].

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 (e.g., [23], [1], [16]), the framework developed in [7] and [9] uses an active learning [3] 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.* [33] 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, the estimation–exploration algorithm [7] (EEA) was introduced, 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. The EEA can also be viewed as a type of co-evolutionary algorithm [17], in which models and tests alter the structure of one another's fitness landscapes.

The EEA has been applied to problems in machine learning [6], gene network identification [11], damage localization in truss structures [22], biomechanics [38], and robotics [10], [9]. In this paper, it is shown how multiple, independent physical robots with the same body plans can accelerate self-modeling by sharing their experiences. This work builds on some preliminary work in [5], in which the algorithm variants reported here were developed using a virtual system. This paper validates those approaches on a physical robot.

Robot teams is an area of intense research. Collective robotics [24], [15], [25] is concerned with robots working together on some collective task. Much of the work in the field has focused on external modeling such as creating global maps [13], [26], [18] or estimating object positions [35]. Other work has focused on modeling an entire robot group for collaborative work [27], or parallel learning of useful controllers [32]. The work reported

Manuscript received August 13, 2007; revised February 4, 2008 and April 21, 2008. This work was supported in part by the NASA Program for Research in Intelligent Systems under Grant NNA04CL10A and the National Science Foundation (NSF) under Grant DMI 0547376. Generalization of the algorithm to multiple robots was supported by an NSF Experimental Program to Stimulate Competitive Research (EPSCoR) Grant (EPS-0236976) awarded to the University of Vermont.

The author is with the Department of Computer Science, University of Vermont, Burlington, VT 05405 USA (e-mail: josh.bongard@uvm.edu).

Digital Object Identifier 10.1109/TEVC.2008.927236

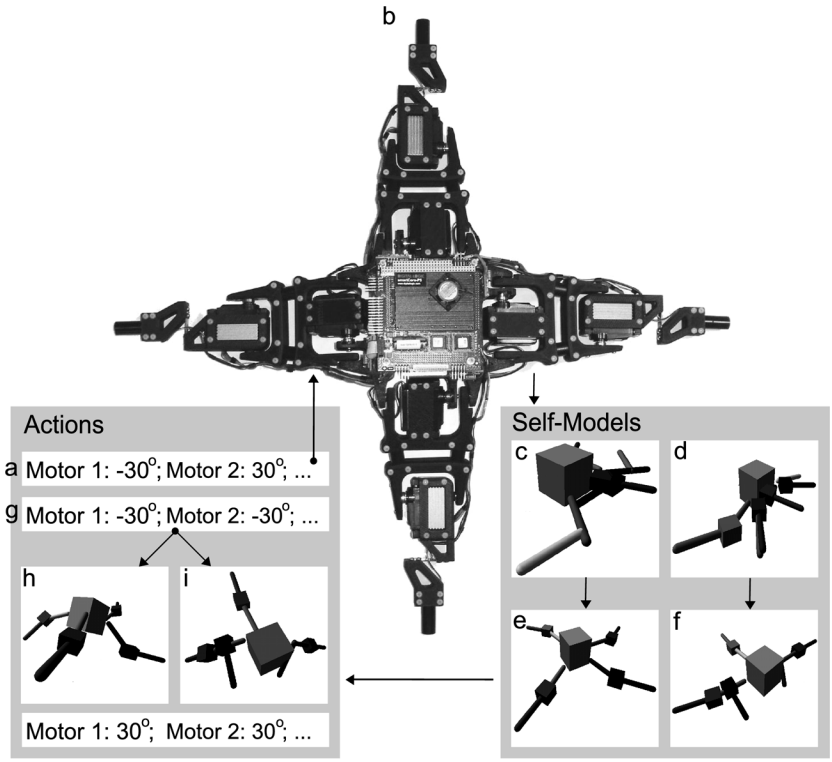

Fig. 1. Physical robot capable of autonomous self-modeling. An action is selected at random (a) and executed by the physical robot (b), which moves it from a planar configuration into a static pose. The resulting orientation of the main body is recorded, and, along with the action that caused it, is passed to a modeling component. The modeling component then optimizes a set of self-models (c) and (d) into a new set of self-models (e) and (f) that better mimic the behavior of the physical robot. A new action is then sought (g) that causes the self-models to assume maximally different poses (h) and (i). This action is then executed on the physical robot (b). Self-modeling then recommences (c) and (d) with two action/result pairs. The process continues until a sufficiently accurate self-model is found, or a set number of cycles elapse.

here differs from these projects in that it enables robots to work together on topological modeling: indirectly inferring their body topologies without directly sensing that topology. As mentioned previously, this is of use when the robot is confronted with some unanticipated change, such as body damage.

The next section describes the algorithm as applied to a single robot, as well as a series of strategies for adapting this algorithm for use by a team of cooperating robots. Section III provides results indicating the relative performance of the variants. Section IV presents discussion of the results and Section V provides some concluding remarks.

II. Methods

The EEA, as applied to a single physical robot, is outlined in Fig. 1. The robot performs a series of *actions*, each of which is a collection of motor commands that move the robot from a planar configuration into a static pose. The sensors record the orientation of the robot during the pose: This sensor data is referred to as the action's *result*. The action/result pair is then used to optimize a set of *self-models*. A good self-model is regarded as one that produces the same sensor data as the physical robot when it is supplied with the same action.

A. The Robot

The physical robot [Fig. 1(b)] is composed of four upper leg parts, four lower leg parts, and a main body. The upper legs are attached to the main body symmetrically, and the lower legs to the upper legs. All joints are actuated by servo motors that can rotate the joint between $-90\,^\circ$ and $90\,^\circ$. The default position for the motors ($0\,^\circ$) causes the robot to lie flat. Rotations to negative angles cause the body part to rotate downward; positive angles cause it to rotate upward. Once the robot performs an action, the orientation of the main body was measured manually to a precision of $\pm 0.1^\circ$: How much it is tilted to the left or right, and how much forward or backward. The tilt information is bundled with the action that caused it and supplied to the self-model synthesis component.

The robot was controlled by an onboard PC-104 computer with a Pentium 166-MHz processor, 64 MB of RAM, and 512 MB of compact flash memory for data storage. For external communication with the computational components (which were run off board on a standard desktop PC), data from the robot was collected on a compact flash card, and uploaded to the external PC for processing. Similarly, actions were transferred to the physical robot on the compact flash card. Batteries supplied the robot with on-board power; a Diamond DMM-32X-AT data acquisition board was used for collecting the joint angle sensor data; and an SV-203 servo control board was used for driving the hinge motors.

B. Self-Models

In the modeling phase [Fig. 1(c)–(f)], a set of 15 self-models are evolved using a parallel hill climber to infer the way in which the robot's body parts are attached together. In previous work [9], it was shown that 15 self-models was sufficient to infer the body plan of the physical robot: more complex robots may require more self-models. The training set is composed of the set

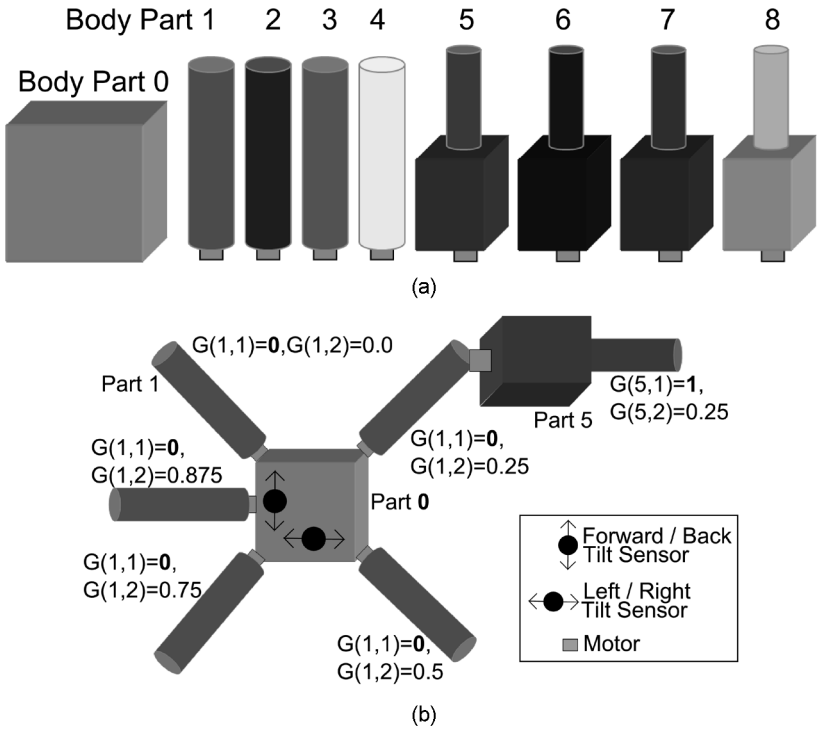

Fig. 2. Genotype to phenotype translation for the self-models. The algorithm begins by knowing how many motorized parts the robot is composed of (a). A self-model genotype G encodes information for connecting the parts together. $G(i,1)$ indicates to which body part i should attach. $G(i,2)$ indicates where on the periphery it should attach. (b) shows five possible placements for part 1, and a sample placement for part 5.

of action/result pairs that have been obtained from the 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 relaxed. The algorithm must indirectly infer how the parts are connected using only the tilt information in the training data.

Each self-model is encoded using two sets of data: *specific self-information* and *invariant self-information*. Specific self-information encodes information assumed to be specific to each individual robot, such as local terrain topology, wear and tear on the joints, and sensor noise. Invariant self-information encodes information that is assumed to be relatively invariant across the robot team, such as the robots' mechanical topology. In the present work, specific self-information encodes information about the slope of the ground on which each robot moves, and invariant self-information encodes the placement of orientation of the robot's body parts.

1) Specific Self-Information: In the present work, the same robot [Fig. 1(b)] is used to simulate two or three robots working in concert. However, it is assumed that multiple robots may be working in different locales and may have accumulated slight wear and tear on both sensors and motors. To encapsulate this uniqueness for each robot, specific self-information is encoded in two values: $B = [b_{lr}, b_{fb}]$, each of which encodes a floating-point value in $[-2, 2]$. These numbers indicate a specific tilting bias for the robot, which may be caused by motor or sensor inaccuracies, or by sloping ground. When a self-model is actuated, and the left/right tilt and forward/back tilt of the self-model's main body is recorded in degrees, b_{lr} is added to the left/right tilt and b_{fb} is added to the forward/back tilt to simulate these undetectable conditions in the physical robot. A self-model with $b_{lr} = -2$, $b_{lr} = 2$, $b_{fb} = -2$ or $b_{fb} = 2$ denotes that the physical robot tilts two degrees to the left, right, forward, or back, respectively, more than it should.

For example, if the physical robot is standing on ground canted two degrees to the right, or its left/right tilt sensor accrues a bias such that it emits a faulty, higher signal, the left/right tilt value returned by the physical robot could always be two degrees greater than a perfect self-model that can only simulate flat ground. If a self-model sets $b_{lr} = 2$, it will be able to successfully reproduce the sensor data of the tilted physical robot. In short, these two values, if inferred correctly, help the robot to produce an accurate self-model of itself, even though there may be aspects of its locale or construction that it cannot model explicitly (such as ground tilt or sensor noise).

2) Invariant Self-Information: In the present work, invariant self-information encodes the mechanical topology of the physical robot: which body parts are attached to which others, and the orientation of those parts.

This information 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. 2(a)] 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. 1(c)–(f). 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. 2(b), 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 positions around the periphery of the parent part, proceeding in a clockwise direction. In the example shown in Fig. 2(b), 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 $(G(5,2)=0.25)$.

The current encoding scheme for G ensures that any set of values for G produces a fully connected self-model. However, it does aid self-modeling in that body parts with lower indices are easier to model than body parts with higher indices. Body part 1, for example, can only be attached to body part 0, which is the correct connection, and body part 2 has a 50% probability of being attached to the correct parent body part (body part 0). In future work a less biased encoding scheme will be implemented.

3) Self-Model Evaluation: Once a self-model is formed, it is actuated with each of the actions that have already been executed by the physical robot.

The self-models are simulated within a three-dimensional, real-time dynamical simulator.[1] At each time step of the simulation, all of the external forces (gravity, momentum and friction), internal forces (such as motor actuation) and constraints (attachment points, collisions with other objects) acting on each body part are combined, and are passed to an integrator that computes the new position and velocity of the part. Each candidate self-model starts in a planar configuration [such as is shown in Fig. 1(c)–(f)], and the virtual motors are rotated to the angles encoded in the current action. The equations of motion are integrated until all of the motors in the self-model reach their respective angles: this often moves the robot into a nonplanar configuration [such as is shown in Fig. 1(h), (i)]. In this configuration, the left/right and forward/back tilt of the self-model's main body is recorded as $m_{lr}^{(i)}$ and $m_{fb}^{(i)}$, respectively, for action i.

The self-model is then relaxed back to the planar configuration, and the second action performed by the physical robot is used to actuate the self-model again. This process is repeated for each action that the physical robot has performed so far.

The **subjective error** of a self-model is then given by

$$m_e = \frac{\sum_{i=1}^{k}\left(|t_{lr}^{(i)} - \left(m_{lr}^{(i)} + b_{lr}\right)| + |t_{fb}^{(i)} - \left(m_{fb}^{(i)} + b_{fb}\right)|\right)}{2k} \tag{1}$$

where k is the total number of actions that have been performed by the robot so far; $t_{lr}^{(i)}$ is the amount the robot tilted to the left or right when it executed action i, $m_{lr}^{(i)}$ is the amount the self-model tilted to the left or right when it executed action i, b_{lr} is the left/right bias encoded by the self-model, $t_{fb}^{(i)}$ is the amount the robot tilted forward or backward when it executed action i, $m_{fb}^{(i)}$ is the amount the self-model tilted to the left or right when it executed action i, and b_{fb} is the forward/back bias encoded by the self-model. m_e therefore computes the mean error, in degrees, between all tilt angles recorded by the robot and all corresponding tilt angles produced by the self-model. In short, the accuracy of a self-model is how well it reproduces the behaviors of the robot when supplied with the same actions.

At the beginning of the algorithm, a random action is performed by the physical robot, and 15 random self-models are created. A hill climber then optimizes each of the 15 self-models in an attempt to minimize m_e. Once a self-model has been evaluated, both B and G are copied, and one or the other is chosen for mutation, with an equal probability. A single value in either B or G is then chosen at random, and a small value chosen from a random Gaussian distribution is added to it. The resulting self-model is then evaluated. If the new self-model achieves a lower subjective error than the parent self-model, the parent genotype is discarded; otherwise, the child is discarded. This process is continued for 200 generations, for each of the 15 self-models. These optimized self-models [Fig. 1(e), (f)] are then passed to the testing phase [Fig. 1(h), (i)] for locating a new action.

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

C. The Actions

The testing phase attempts to find a new action that, when executed by the robot, will provide more information about its topology. The physical robot is capable of 36 possible actions.

The first eight actions command one of the motors to rotate downward by 30 ° and the other seven to rotate upward by 30 °. The remaining 28 actions command two of the motors to rotate downward by 30 ° and the other six upward by 30 °. In earlier work [5], it was shown that by allowing only one or two body parts to affect the tilt of the main body, the robot could more rapidly infer its own topology. Body parts rotated downward lift the main body, while those rotated upward do not affect tilt. This is illustrated by the four sample poses shown in Fig. 3.

Initially, one of the 36 actions is selected at random and sent to the target robot to generate the first action/result pair [Fig. 1(a)]. On the second and subsequent cycles, the optimized self-models from the modeling phase are used to determine which new action to send to the robot. During these cycles, the fitness of each remaining untried action is computed. The action with the highest fitness is sent to the physical robot for evaluation. The fitness of each action is determined as followed. Each action is sent, in turn, to the 15 self-models. The behaviors of the self-models in response to the action determine its fitness.

More precisely, the fitness of an action a_f is computed as

$$a_f = v - d \tag{2}$$

$$v = \frac{\sigma^2(\mathbf{m}_{lr}) + \sigma^2(\mathbf{m}_{fb})}{2} \tag{3}$$

$$d = \frac{\sum_{i=1}^{15} |m_{lr}^{(i)} - m_{lr}^{\prime(i)}| + |m_{fb}^{(i)} - m_{fb}^{\prime(i)}|}{30} \tag{4}$$

where v rewards the candidate action for inducing variance across the self-models. $\sigma^2(\mathbf{m}_{lr})$ represents the variance across

[1] www.ode.org

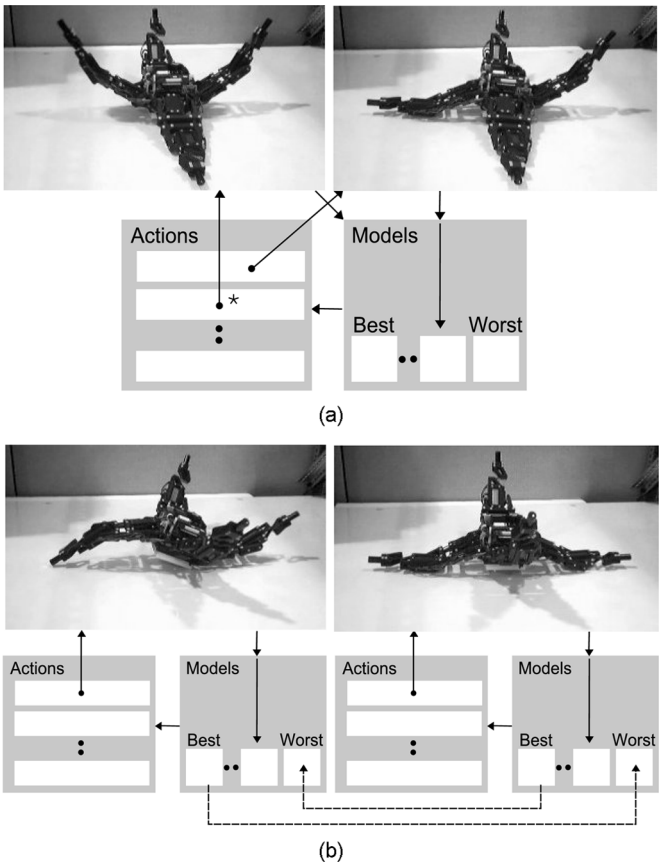

Fig. 3. Alternative approaches for exploiting two or more robots for self-modeling. In the *Combined* variant (a), two robots each execute different actions, and then feed those two actions (along with their results) into a common base algorithm. In the *Swap* variant (b), each robot maintains its own independent base algorithm, but they swap their current best self-models.

the left and right tilting of the 15 optimized self-models when supplied with action a, and $\sigma^2(\mathbf{m}_{fb})$ represents the variance across the forward and backward tilting of the 15 optimized self-models. For example a low value for $\sigma^2(\mathbf{m}_{lr})$ indicates that all the self-models tilted to the left, or all to the right; a high value indicates some tilted to the left, some did not tilt left or right, and some tilted to the right. Likewise for $\sigma^2(\mathbf{m}_{fb})$.

The first term v rewards an action for how much it causes the optimized self-models to tilt in different directions. This reflects the theoretical finding from active learning in which it was shown [33] that the best way to choose new training data for labeling by the target system (in this case, the robot) is the one that causes the self-models to disagree in their predictions about the label for this training data. Once this training data is labeled by the system and added to the training set, now only some of the self-models, not all, will agree with the results from the system, because the self-models disagree about the new training data. Further modeling can then replace these recently revealed erroneous self-models with new self-models that explain all the old training data, plus the new training data.

The second term d penalizes potentially "dangerous" actions. Dangerous actions are those which may cause the robot to behave very differently from the self-models, even though the self-models are topologically very similar to the robot. For example, if an action causes the robot to balance on its left and right upper legs, it may maintain an untilted main body [as shown in Fig. 3(b)], while topologically similar, yet not perfectly accurate self-models may predict that the main body will tilt. These dangerous actions have the effect of making the self-model search space very rugged: relatively accurate self-models suddenly experience a misleading increase in error when such new training data is collected. "Dangerous" actions are avoided by taking the current self-model set and producing mutants of each of them, as described in Section II. The 15 self-models $(m^{(1,\ldots,15)})$ and their corresponding mutant self-models $(m'^{(1,\ldots,15)})$ are then actuated with the current candidate action, and the predicted tilt angles from the original models are stored in $m_{lr}^{(1,\ldots,15)}$ and $m_{fb}^{(1,\ldots,15)}$, and the predicted tilt angles from the mutated models are stored in $m'^{(1,\ldots,15)}_{lr}$ and $m'^{(1,\ldots,15)}_{fb}$. d then records the mean disagreement between the tilt angles of the original and mutant model pairs. This amount of disagreement predicts the potential disagreement between the robot and a topological similar self-model. Self-models with high values of d are thus avoided. For a more detailed treatment of this phenomenon, see [4].

D. Algorithm Variants

A total of 14 algorithm variants were tested (Table I), and the relative ability of the variants to produce accurate self-models was measured. Some of the variants relied on a single robot operating in isolation (Fig. 1), and is referred to as the base algorithm. Some of the variants rely on several robots drawing training data from, and combining resulting sensor data into the base algorithm [Fig. 3(a)]. Some of the variants also rely on several robots, but each robot maintains its own base algorithm [Fig. 3(b)]. In the variants where multiple robots are employed, a single physical robot simulates several robots in the following manner. When an action meant for robot 1 is received, the physical robot performs the action, and adds a small bias angle to both the resulting left/right and forward/back tilt angle. The two biases (one for left/right, the other for forward/back) are chosen randomly from $[-2^\circ, 2^\circ]$. When an action for robot 2 is received by the physical robot, it again performs the action but now adds a bias unique to robot 2, and so forth. This simulates several topologically similar robots moving over differently tilted ground, or with accrued biases in their motors or sensors. A small amount of uniform noise is also added to each tilt angle returned by the robot selected from $[-0.1^\circ, 0.1^\circ]$. Rather than perform the algorithm variants on the robot in real-time, each of the 36 actions were supplied to the robot and the resulting tilt angles were recorded before any of the experiments were attempted. For each "robot" involved in an algorithm variant, the 36 action/result pairs were copied and the tilt angle biases for that robot were added.

1) Combined Variant: This variant allows two robots to share a common instance of the EEA, as shown in Fig. 3(a). 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

TABLE I
SUMMARY OF ALGORITHM VARIANTS

Index	Variant	Description
1	B__1	The [b]ase algorithm runs on a single robot. (Fig. 1)
2	C__2	Two robots [c]ombine data into a common algorithm. (Fig. 3a)
3	S__2	Two robots run their own algorithm, but [s]wap best self-models. (Fig. 3b)
4	C__3	Three robots [c]ombine data into a common algorithm.
5	S__3	Three robots run their own algorithm, but [s]wap best self-models.
6	SM_2	Two robots run their own algorithm, but [s]wap [m]ultiple self-models.
7	SM_3	Three robots run their own algorithm, but [s]wap [m]ultiple self-models.
8	B_P1	The [b]ase algorithm runs on a single robot; body parts are [p]assed upward from worse to better self-models.
9	C_P2	Two robots [c]ombine data into a common algorithm; body parts are [p]assed upward from worse to better self-models.
10	S_P2	Two robots run their own algorithm, but [s]wap best self-models; body parts are [p]assed upward from worse to better self-models.
11	C_P3	Three robots [c]ombine data into a common algorithm; body parts are [p]assed upward from worse to better self-models.
12	S_P3	Three robots run their own algorithm, but [s]wap best self-models; body parts are [p]assed upward from worse to better self-models.
13	SMP2	Two robots run their own algorithm, [s]wap [m]ultiple self-models, and body parts are [p]assed upward from worse to better self-models.
14	SMP3	Three robots run their own algorithm, [s]wap [m]ultiple self-models, and body parts are [p]assed upward from worse to better self-models.

sent to the common modeling phase. In the base algorithm, the first pass through the modeling phase optimizes models against one action/result pair; during the kth pass, models are optimized against k action/result pairs. In variant C_2, two robots combine results into a single base algorithm: the first pass through the modeling phase optimizes models against two action/result pairs; during the kth pass through the modeling phase, they are optimized against $2k$ action/result pairs. In variant C_3, three robots combine results into a single base algorithm. The three actions that best induce self-model disagreement are sent to the three robots after each pass through the testing phase. During the kth pass through the modeling phase, self-models are optimized against $3k$ action/result pairs.

2) Swap Variant: Fig. 3(b) outlines *Swap*, in which two robots maintain their own base algorithm. However, unlike the base algorithm, when a pass through the self-modeling phase in *Swap* finishes, the robot waits for a signal from the other robot (or robots) that it has also 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 sends it to robot 2. Conversely, robot 2 sends its best self-model to robot 1. Each robot then deletes its worst self-model and replaces it with the model received from the other robot. The robots then continue on to the testing phase. The best self-model is defined as the self-model among the current 15 candidate self-models with lowest subjective error (1); the worst self-model is defined as the self-model with the highest subjective error. In variant S_2 two robots swap best self-models. In variant S_3, three robots swap best self-models: each robot sends its best self-model to the other two robots, and overwrites its worst and second worst self-model with the best self-models received from the other two robots.

3) Mult Extension: An extension was applied to the *Swap* variants in which robots swap not only their best self-model, but multiple self-models. In variant SM_2, each robots sends its $\lfloor(15)/(2)\rfloor = 7$ best self-models to the other robot, and overwrites its seven worst self-models with those received from the other robot. In SM_3, each of the three robots sends its $\lfloor(15)/(3)\rfloor = 5$ best self-models to the other two robots. Each robot overwrites its middle five self-models with those from the first of the other two robots, and its worst five self-models with those from the second of the other two robots.

4) Passup Extension: In the self-modeling phase, self-models are optimized independently using a hill climber. The *PassUp* extension introduces a method for propagating genetic material from one self-model to another, without reducing the variation in the self-model population. In an earlier study [4], crossover was introduced to the base algorithm, but the resulting algorithm variant performed worse. It was observed that because crossover reduces the overall variation in the self-model population, it is more difficult in the testing phase to locate an action that induces disagreement among the models. *PassUp* is a modification of the crossover operator, and functions as follows. After each generation in the self-modeling phase, the 15 child self-models are mutated normally. Then, two of the child self-models are chosen at random, and each row from G in the worse of the two self-models is passed up into and overwrites the corresponding row in the better self-model with a 50% probability. The better self-model does not contribute any genetic material to the worse self-model. In this way, a self-model from one robot, when imported by another robot, may have a relatively high subjective error within the new self-model set. However, it may model a few body parts correctly, such as the body parts that were tested in

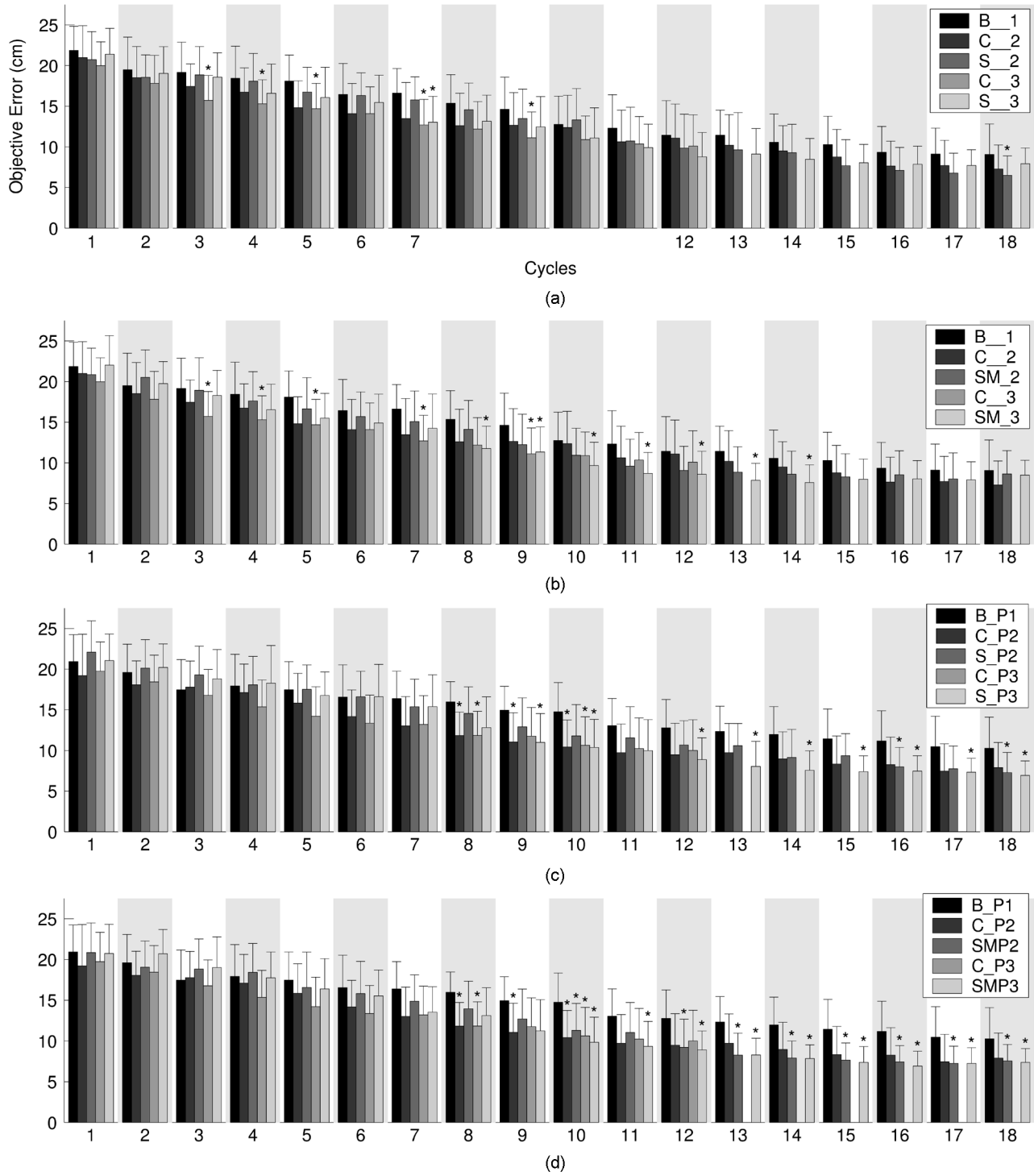

Fig. 4. Relative modeling performance for the 14 algorithm variants. (a) The *Swap* and *Combined* variants are compared to the base algorithm; (b) the *Swap* variants with trading of multiple self-models are compared against the base algorithm and *Combined* variants; (c) all variants with the *PassUp* extension are compared; and (d) all variants with the *PassUp* extension are compared against the *Swap* variants with both the *PassUp* and *Mult* extensions. Runs that produced models with statistically significant more accuracy than the base algorithm are marked with an asterisk. Error bars indicate one unit of standard deviation.

the other robot. Using *PassUp*, an imported self-model may contribute well-modeled body parts up into the more accurate self-models of the current set.

The *PassUp* extension was introduced into the base algorithm (B_P1), two robots combining results into a shared base algorithm (C_P2), two robots swapping best self-models (S_P2), three robots combining results into a shared base algorithm (C_P3), three robots swapping best self-models (S_P3), two robots swapping multiple self-models (*SMP2*), and three robots swapping multiple self-models (*SMP3*).

III. Results

Thirty independent trials of the 14 algorithm variants were conducted. Each trial was conducted for 18 cycles, with the ex-

ception of the $C\_\_3$ and C_P3 variants, which were only run for 12 cycles. Because these variants use up three actions at each cycle, all 36 actions are executed by the robots after 12 cycles. For each pass through the self-modeling phase of $C\_\_2$, the self-models are optimized for only 100 generations, rather than 200 generations as in the other algorithm variants. In $C\_\_3, 67$ generations of self-modeling optimization are performed. This normalizes the amount of computational effort performed by all algorithm variants to nearly 300 000 self-model evaluations per independent trial.

For each trial, and after each pass through the modeling phase, the self-model with the lowest m_e was selected, and its **objective error** was measured, given as

$$m_o = \frac{\sum_{i=1}^{8} \sqrt{\left(t_x^{(i)} - m_x^{(i)}\right)^2 + \left(t_z^{(i)} - m_z^{(i)}\right)^2}}{8} \quad (5)$$

where $t_x^{(i)}$ is the horizontal position of body part i on the robot when it lies flat, $m_x^{(i)}$ is the horizontal position of body part i on the self-model, $t_z^{(i)}$ is the z-position of body part i on the robot, and $m_z^{(i)}$ is the z-position of body part i on the self-model. The objective error of a self-model is equal to the mean Euclidean distance between the positions of the robot's body parts, and the self-model's body parts. This metric then gives an indication of how close the self-model is to the topology of the robot, independent of the collected action/result pairs.

Fig. 4(a) reports the relative modeling abilities of the first five algorithm variants, without the *Mult* and *PassUp* extensions. Bars indicate the mean objective errors for that variant, for that cycle. As can be seen, the objective errors of the self-models decrease with increasing cycles, as expected, for all variants. Despite a few, early sporadic improvements by the *Combined* variant with three robots due to its threefold increase in the number of action/result pairs it receives per cycle, no variants outperformed the base algorithm.

Adding the *Mult* extension to the *Swap* variants (SM_2 and SM_3) does not improve these variants relative to the base algorithm and *Combined* variants, as indicated by Fig. 4(b). However, adding the *PassUp* extension to all five variants does confer a significant advantage to the *Swap* variant with three robots (S_P3), as indicated by Fig. 4(c); S_P3 consistently generates more accurate self-models than the base algorithm from cycle 12 onward. When both the *Mult* and *PassUp* extensions are integrated into the *Swap* variant, *SMP2* and *SMP3* significantly outperform the base algorithm from cycle 12 onward [Fig. 4(d)].

A second performance metric for measuring self-model quality was also applied to the algorithm variants. At the end of each pass through the modeling phase, the self-model with the lowest subjective error was extracted, and it was determined whether it had the same topology as the robot. Fig. 5(a)–(f) reports the topologies of the best self-models taken from cycle 10 of the first six trials of the base algorithm. As can be seen, none of the self-models have the correct topology. However, four of the first six trials for the *SMP2* variant discovered the robot's true topology after ten cycles [Fig. 5(g)–(l)]. That is, all eight body parts in the self-model are attached to the correct parent body part. Five of the first six trials for *SMP3* variant discovered the robot's true topology after 10 cycles [Fig. 5(g)–(l)]. As can be seen, some of the topologically correct self-models may still be relatively inaccurate in that their attachment positions are skewed.

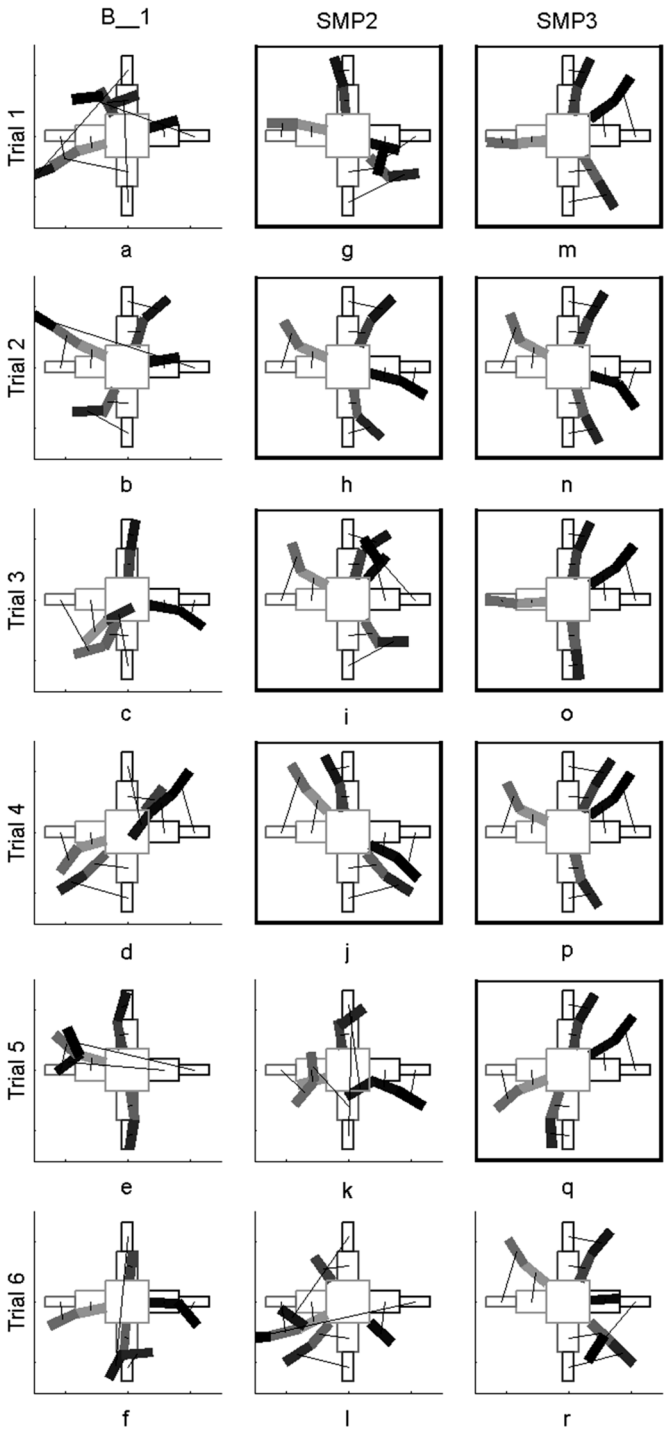

Fig. 5. Sample topologies of optimized self-models. Each panel displays the best self-model extracted after ten passes through the modeling phase for the first six trials of the base algorithm $B\_\_1$; (a)–(f), the *Swap* variant using the *Mult* and *PassUp* extensions for two robots *SMP2*; (g)–(l), and the *Swap* variant using the *Mult* and *PassUp* extensions for three robots *SMP3*; (m)–(r). Topologically correct models are boxed.

For the base algorithm and all of the *Swap* variants, the fraction of trials that converged on a self-model with the correct

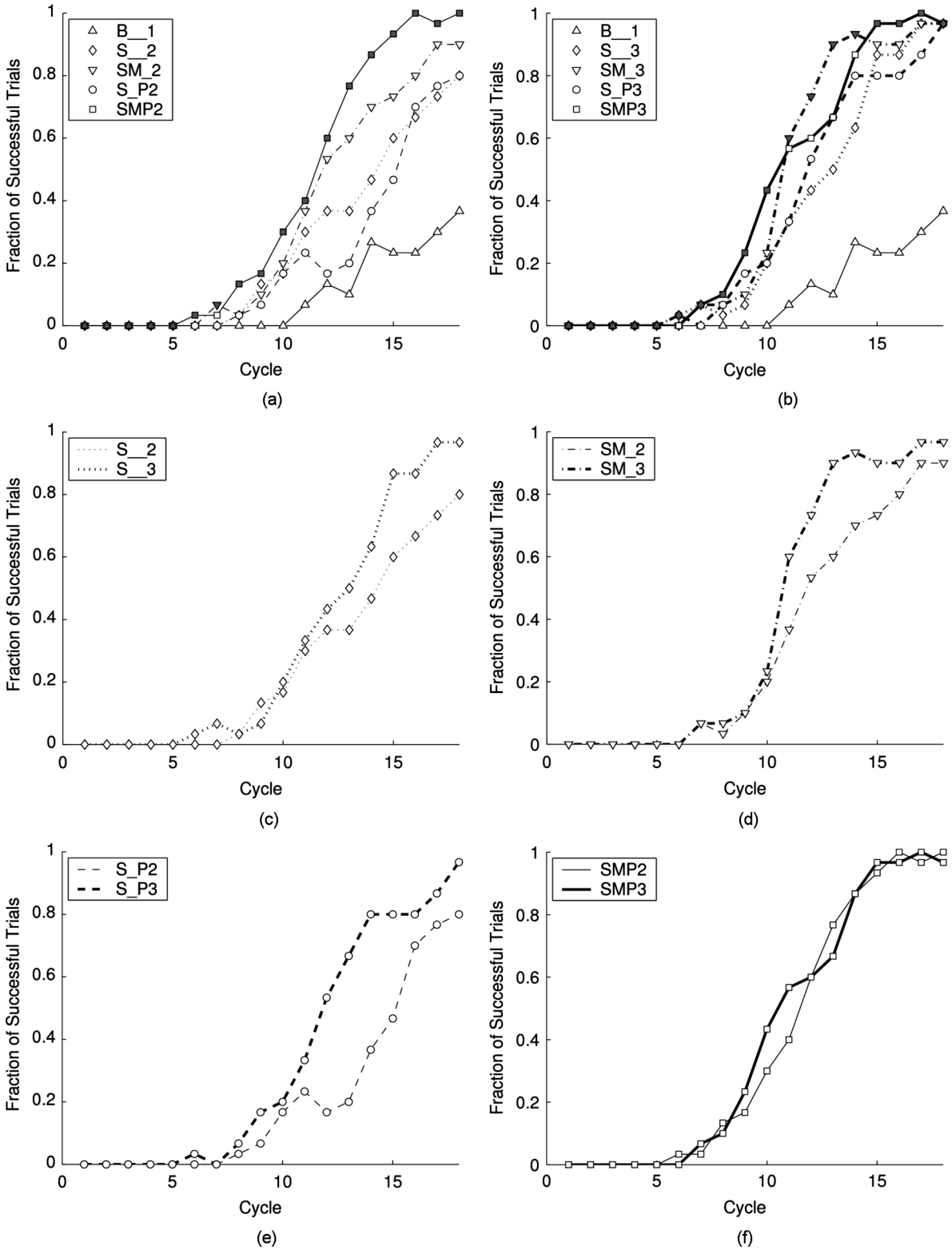

Fig. 6. Relative modeling performance of the base algorithm and *Swap* variants. Each marker indicates the fraction of independent trials that have discovered a topologically correct self-model for that algorithm variant, by that cycle. (a) Modeling abilities of the base algorithm and *Swap* variants used by two robots. (b) Modeling abilities of the base algorithm and *Swap* variants used by three robots. Comparative modeling abilities for two and three robots swapping self-models without the *Mult* or *PassUp* extension (c); with only the *Mult* extension (d); with only the *PassUp* extension (e); and with both the *Mult* and *PassUp* extensions (f).

topology was computed, for each cycle. This metric indicates how often, and how early, a given algorithm variant converges on a topologically correct self-model. Fig. 6(a) compares the four *Swap* variants, using two robots, against the base algorithm. As can be seen, the base algorithm $B\_\_1$ does not discover a topologically correct self-model until cycle 11, while

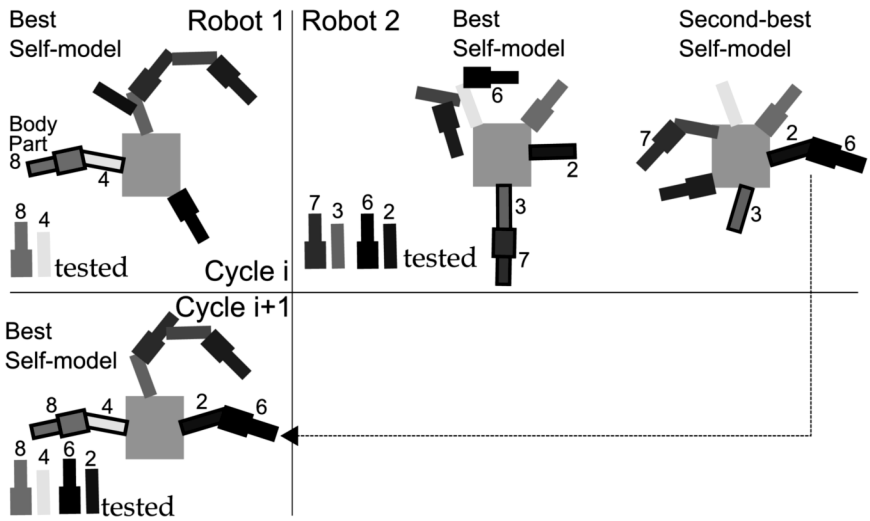

Fig. 7. Illustration of how *Mult* and *PassUp* accelerate self-modeling. Correctly modeled body parts from robot 2 (parts 2 and 6) are sent to robot 1 via the *Mult* variant, and are then integrated during model optimization into the best model generated by robot 1 via the *PassUp* extension.

the *Swap* variants begin discovering topologically correct self-models after the fifth cycle. The *SMP2* variant outperforms the base algorithm and other *Swap* variants, in that it consistently achieves a greater fraction of independent trials exhibiting topologically correct self-models than the other algorithm variants (shaded squares).

Fig. 6(b) compares the four *Swap* variants, using three robots, against the base algorithm. Adding a third robot to the two collaborating robots does not reduce the number of cycles necessary to achieve a topologically correct self-model: the *Swap* variants still only discover such self-models after cycle 5. However, the *Swap* variant with both the *Mult* and *PassUp* extensions (*SMP3*, shaded squares) outperforms the other *Swap* variants, with the exception of cycles 11 to 14 (shaded triangles).

Fig. 6(c)–(f) indicate that three robots swapping self-models outperform two robots, for all *Swap* variants except *SMP3*. Presumably, the *Mult* and *PassUp* extension empower *SMP2* sufficiently that there is no additional performance gain to be had by adding a third robot.

IV. Discussion

The results indicate that in general, two robots swapping self-models synthesize accurate self-models using fewer action-result data pairs than one robot operating alone, or two robots sharing a common modeling algorithm. This is indicated in Fig. 4(d), in which only the *Swap* variants that incorporate the *Mult* and *PassUp* extensions produce significantly more accurate self-models than both the base algorithm and the *Combined* variants. The performance advantage realized by the *Swap* variants can be explained as follows. When imported by a recipient robot, a self-model (or set of self-models) may be optimized to explain different body parts not yet tested by the donor robot. When the improved self-model (or set of self-models) is sent back, it may be further improved to explain the new body parts just tested by the donor robot. Therefore self-models are optimized against the body parts tested by both robots. Further, self-models in the *Swap* variants need only account for the biases of their current host robot: when received, self-models may be adapted to account for the local biases, and then adapted back to the original biases of the donor when transferred back to it. In the *Combined* variants, self-models must be adapted to explain all of the biases of the contributing robots, which is not achievable.

Fig. 6(a) and (b) indicate that the *Swap* variants which incorporate the *Mult* and *PassUp* extensions outperform the other *Swap* variants, indicating that both extensions confer some advantage compared to variants with only one or none of these extensions. Fig. 7 illustrates why this combination confers an advantage. Different robots perform different actions, and thereby obtain indirect evidence about the positions of different body parts. For instance in Fig. 7, the first robot obtains data about the positions of the left-hand body parts (parts 4 and 8) while the second robot obtains data about the right-hand and lower body parts (parts 2 and 6, and 3, and 7, respectively). If the first robot then obtains information about the right-hand body parts, it may incorporate those parts from the best self-model exported by the second robot using *PassUp*, without having to model those parts *ab initio*. If on the other hand it obtains information about the lower body parts, it may incorporate those parts from the second-best self-model exported by the second robot using *Mult* (indicated by the dotted line in Fig. 7).

Fig. 6(c)–(f) indicates how three robots swapping self-models improve over two robots. First, it can be seen that all algorithms start to create topologically correct self-models only after five actions have been performed. That is, teams of three robots do not begin to discover correct models earlier than pairs of robots, or robots working alone. This indicates that there is some hard limit on the minimum number of actions required before topologically correct self-models can be found. However, consistent performance is more prevalent in teams of three robots than it is in teams of two, or in independent robots: all *Swap* variants relying on three robots discover more correct self-models, with less actions, than *Swap* variants relying on two robots (or one robot acting alone). This indicates that the more robots that share self-models, the less actions (and attendant self-modeling) are required for the majority of them to realize correct self-models.

V. Conclusion

In this paper, it has been shown that quadrupedal robots which share experiences more rapidly synthesize models describing their own body. More specifically, a series of alternatives for sharing experiences were explored, and it was found that it is more advantageous for quadrupedal robots to maintain independent self-modeling algorithms, and swap their best self-models, rather than combining experiences into a common modeling algorithm. This is a particularly encouraging result as robots operating in the field need only communicate with their peers in order to swap experiences, rather than communicate back to a central computational resource: For increasing numbers of independent agents, peer-to-peer communication strategies are most robust and scalable than many-to-one communication strategies.

Also, it has been shown how robots with a specific morphology should share experiences. Each robot should maintain not just a single self-model, but a set of candidate self-models, and more accurate self-models should only import genetic material from less accurate self-models. This constraint keeps the self-model population from converging on seemingly correct, but objectively incorrect self-models. Second, robots should share several of their best self-models, rather than just the currently best self-model, as often randomly selected genetic material from a self-model with intermediate accuracy contains correctly modeled body parts, and can be imported into the self-models of another robot that has not modeled those parts correctly yet. Previously, it has been shown that maintaining multiple models provides a mechanism for extracting new information from the system being modeled [33], [9]. The current work provides another reason why it is beneficial for a robot to maintain several, rather than a single self-model.

In previous work [9], it was demonstrated that a physical robot that can autonomously model its own body can use that self-model to recover from unanticipated situations, such as body damage. By extension, robots that collaborate to accelerate self-modeling will, as a group, be capable of more rapid recovery from unanticipated situations. More specifically, if one robot diagnoses damage through self-modeling, and generates a recovery strategy, it can communicate both the self-model and recovery strategy to other robots. Other robots that then suffer similar injury can rapidly diagnose the damage by matching imported self-models against the new sensor data. The recovery strategy associated with the best matching imported self-model can then be executed, thereby greatly accelerating diagnosis and recovery. This dynamic will be explored in future work.

It is interesting to note that a team with at least one damaged robot would confer an even greater performance advantage on the distributed approach to self-modeling (i.e., *Swap*), as self-models exported by a damaged robot would simply be discarded by intact robots and assimilated by similarly damaged robots. The centralized approach to self-modeling (i.e., *Combined*) would be forced to settle on self-models that fail to accurately reflect either the damaged or intact robots.

In related work, we have recently shown how topological modeling can be applied to other nonlinear, coupled systems in addition to robots [8]. It was shown there how the time complexity for model search can be reduced from exponential to polynomial time. In future work, this simplifying mechanism will be introduced to robots performing self-modeling. Also, the approach will be applied to robots with different body plans and greater degrees of freedom to determine the generality of the algorithm.

For higher animals, language and imitation are the only conduits available for sharing experience: a human may tell another human how to operate a new tool, or a parent may demonstrate an action to a child. The form of communication demonstrated between intelligent agents in this paper provides a direct form of interaction: Agents can communicate information about each other's body directly, by sharing self-models. The field of biorobotics [39], [19], [30] is concerned with instantiating biological structures or behaviors in robots. This work demonstrates an additional mechanism by which artificial agents can surpass biological agents: communicating self-knowledge directly with one another, rather than indirectly through language or imitation.

Most collective robotics projects realize robot teams that communicate with one another in order to establish cooperative behaviors [2], [29], [34], synthesize global maps [13], [26], or estimate object positions [35]. This work demonstrates the utility of another form of collaboration: The sharing of experiences for self-modeling, and eventually for accelerating adaptation to and recovery from unanticipated situations, which constantly challenge an autonomous robot operating in unstructured environments.

Acknowledgment

The author would like to thank members of the Computational Synthesis Laboratory at Cornell University, as well as staff of the Vermont Advanced Computing Center.

References

[1] H. Andrew, "System identification using genetic programming," in *Proc. 2nd Int. Conf. Adaptive Comput. Eng. Design Control*, 1996, pp. 57–62.

[2] T. Balch and R. Arkin, "Communication in reactive multiagent robotic systems," *Auton. Robots*, vol. 1, no. 1, pp. 27–52, 1994.

[3] Y. Baram, R. El-Yaniv, and K. Luz, "Online choice of active learning algorithms," *J. Mach. Learn. Res.*, vol. 5, pp. 255–291, 2004.

[4] J. Bongard, "Action-selection and crossover strategies for self-modeling machines," in *Proc. Genetic Evol. Comput. Conf. (GECCO'07)*, 2007, pp. 198–205.

[5] J. Bongard, "Exploiting multiple robots to accelerate self-modeling," in *Proc. Genetic Evol. Comput. Conf. (GECCO'07)*, 2007, pp. 214–221.

[6] J. Bongard and H. Lipson, "Active coevolutionary learning of deterministic finite automata," *J. Mach. Learn. Res.*, vol. 6, pp. 1651–1678, Oct. 2005.

[7] J. Bongard and H. Lipson, "Nonlinear system identification using coevolution of models and tests," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 361–384, Aug. 2005.

[8] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," *Proc. National Acad. Sci.*, vol. 104, no. 24, pp. 9943–9948, 2007.

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

[10] J. C. Bongard and H. Lipson, "Automated robot function recovery after unanticipated failure or environmental change using a minimum of hardware trials," in *Proc. 2004 NASA/DoD Conf. Evolvable Hardware*, Seattle, WA, 2004, pp. 169–176.

[11] J. C. Bongard and H. Lipson, "Automating genetic network inference with minimal physical experimentation using coevolution," in *Proc. 2004 Genetic Evol. Comput. Conf.*, Seattle, WA, 2004, pp. 333–345.

[12] D. Cliff, P. Husbands, and I. Harvey, "Evolving visually guided robots," in *Proc. 2nd Int. Conf. Simulation of Adaptive Behavior*, J.-A. Meyer, H. Roitblat, and S. Wilson, Eds., 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 Trans. Robotics Autom.*, vol. 19, no. 2, pp. 238–249, Apr. 2003.
[14] D. Floreano and F. Mondada, P. Husbands and J.-A. Meyer, Eds., "Hardware solutions for evolutionary robotics," *EvoRobots*, pp. 137–151, 1998.
[15] *Distributed Autonomous Robotic Systems 7*, M. Gini and R. Voyles, Eds. New York: Springer, 2006.
[16] G. Gray, D. Murray-Smith, Y. Li, K. Sharman, and T. Weinbrenner, "Nonlinear model structure identification using genetic programming," *Control Eng. Practice*, vol. 6, pp. 1341–1352, 1998.
[17] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D*, vol. 42, pp. 228–234, 1990.
[18] A. Howard, G. S. Sukhatme, and M. J. Mataric, "Multi-robot mapping using manifold representations," *Proc. IEEE—Special Iss. Multi-Robot Syst.*, vol. 94, no. 7, pp. 1360–1369, 2006.
[19] A. Ijspeert, A. Crespi, D. Ryczko, and J.-M. Cabelguen, "From swimming to walking with a salamander robot driven by a spinal cord model," *Science*, vol. 315, no. 5817, pp. 1416–1420, 2007.
[20] N. Jakobi, "Evolutionary robotics and the radical envelope of noise hypothesis," *Adaptive Behavior*, vol. 6, no. 1, pp. 131–174, 1997.
[21] D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi, "Online evolution for a self-adapting robotics navigation system using evolvable hardware," *Artif. Life*, vol. 4, pp. 359–393, 1998.
[22] B. Kouchmeshky, W. Aquino, H. Lipson, and J. C. Bongard, "Co-evolutionary strategy for structural damage identification using minimal physical testing," *Int. J. Numer. Methods Eng.*, vol. 69, no. 5, pp. 1085–1107, 2006.
[23] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Boston, MA: MIT Press, 1992.
[24] C. Kube and E. Bonabeau, "Cooperative transport by ants and robots," *Robotics Auton. Syst.*, vol. 30, no. 1–2, pp. 85–101, 2000.
[25] K. Lerman, C. Jones, A. Galstyan, and M. J. Mataric, "Analysis of dynamic task allocation in multi-robot systems," *Int. J. Robotics Res.*, vol. 25, no. 3, pp. 225–242, 2006.
[26] R. Madhavan, K. Fregene, and L. Parker, "Distributed cooperative outdoor multirobot localization and mapping," *Auton. Robots*, vol. 17, no. 1, pp. 23–39, 2004.
[27] A. Martinoli, K. Easton, and W. Agassounon, "Modeling of swarm robotic systems: A case study in collaborative distributed manipulation," *Int. J. Robotics Res.*, vol. 23, no. 4, pp. 415–436, 2004.
[28] S. Nolfi and D. Floreano, *Evol. Robotics*. Boston, MA: MIT Press, 2000.
[29] E. Pagello, A. D'Angelo, C. Ferrari, R. Polesel, R. Rosati, and A. Speranzon, "Emergent behaviors of a robot team performing cooperative tasks," *Adv. Robotics*, vol. 17, no. 1, pp. 3–19, 2003.
[30] R. Pfeifer, M. Lungarella, and F. Iida, "Self-organization, embodiment, and biologically inspired robotics," *Science*, vol. 318, no. 5853, p. 1088, 2007.
[31] J. B. Pollack, H. Lipson, S. Ficici, P. Funes, G. Hornby, and R. Watson, "Evolutionary techniques in physical robotics," in *Evolvable Systems: From Biology to Hardware*, J. Miller, Ed. New York: Springer-Verlag, 2000, pp. 175–186.
[32] J. Pugh and A. Martinoli, "Multi-robot learning with particle swarm optimization," in *Proc. 5fth Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2006, pp. 441–448.
[33] H. S. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proc. Fifth Workshop on Computational Learning Theory*, New York, 1992, pp. 287–294, ACM.
[34] K. Støy, W. Shen, and P. Will, "A simple approach to the control of locomotion in self-reconfigurable robots," *Robotics Auton. Syst.*, vol. 44, no. 3–4, pp. 191–199, 2003.
[35] A. W. Stroupe, M. C. Martin, and T. Balch, "Distributed sensor fusion for object position estimation by multi-robot systems," in *Proc. IEEE Int. Conf. Robotics Auton.*, 2001, vol. 2, pp. 1092–1098.
[36] R. Tedrake, T. Zhang, and H. Seung, "Learning to walk in 20 minutes," in *Proc. Fourteenth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, 2005, Yale Univ..
[37] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.
[38] F. J. Valero-Cuevas, V. V. Anand, A. Saxena, and H. Lipson, "Beyond parameter estimation: Extending biomechanical modeling by the explicit exploration of model topology," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 11, pp. 1951–1964, Nov. 2007.
[39] B. Webb and R. Thomas, *Biorobotics: Methods and Applications*. Cambridge, MA: AAAI Press/MIT Press, 2001.

Josh C. Bongard received the B.Sc. honors degree in computer science from McMaster University, Hamilton, ON, Canada, in 1997, the M.Sc. degree in evolutionary and adaptive systems from the School of Cognitive and Computing Sciences, University of Sussex, Sussex, U.K., in 1999, and the Ph.D. degree from the Artificial Intelligence Laboratory, University of Zurich, Zurich, Switzerland, for research in the field of evolutionary robotics.

He joined the faculty of the Department of Computer Science, University of Vermont, Burlington, in 2006. Prior to this appointment, he was a Postdoctoral Researcher in the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY.