Task Switching in Multirobot Learning through Indirect Encoding

In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2011 San Fransisco, CA)

David B. D'Ambrosio, Joel Lehman, Sebastian Risi, and Kenneth O. Stanley
Department of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816-2362 USA
(ddambro,jlehman,risi,kstanley)@eecs.ucf.edu

***Abstract*— Multirobot domains are a challenge for learning algorithms because they require robots to learn to cooperate to achieve a common goal. The challenge only becomes greater when robots must perform heterogeneous tasks to reach that goal. Multiagent HyperNEAT is a neuroevolutionary method (i.e. a method that evolves neural networks) that has proven successful in several cooperative multiagent domains by exploiting the concept of *policy geometry*, which means the policies of team members are learned as a function of how they relate to each other based on canonical starting positions. This paper extends the multiagent HyperNEAT algorithm by introducing *situational policy geometry*, which allows each agent to encode multiple policies that can be switched depending on the agent's state. This concept is demonstrated both in simulation and in real Khepera III robots in a patrol and return task, where robots must cooperate to cover an area and return home when called. Robot teams that are trained with situational policy geometry are compared to teams that are not and shown to find solutions more consistently that are also able to transfer to the real world.**

I. INTRODUCTION

Training multiple robots to cooperate and accomplish a goal is difficult because each robot must learn to perform a complementary task. Multiagent HyperNEAT [1], [2] is a learning method that addresses this challenge by exploiting a concept called *policy geometry*. Inspired by real-life teams, the idea in policy geometry is that the policies of team members are derived from their canonical location within the team (e.g. at the start of a match). For example, on a soccer team, the goalie is the most defensive player and closest to the edge of the field while the players become more offensive who start closer to the midfield. Thus, rather than learning individual policies for each agent, the approach of multiagent HyperNEAT is to learn a pattern of policies and how they relate to one another based on the team's policy geometry. In this way teams trained by multiagent HyperNEAT can share basic or important skills without the need for each agent to independently discover them.

The problem of multirobot learning is further complicated when robots must switch between tasks during operation. To do so, the policy of the robot must encode the ability to perform all the desired tasks in addition to knowing the appropriate time to switch between them, which may be difficult in noisy or uncertain environments. In addition, such problems can be deceptive to the learning algorithm because of local optima caused by solving one task at the expense of solving the others. While there are existing approaches to this problem such as decomposing the tasks [3], exploiting modular structures [4], or training plastic networks [5], the problem of training autonomous robots to switch between tasks remains challenging.

To address this challenge, this paper introduces an extension to multiagent HyperNEAT and the standard policy geometry concept called *situational policy geometry* that generates *multiple policies* for the same agent, among which it can switch depending on its current state. That way, individual robots can learn how to perform multiple *tasks* by learning how they relate to each other, which is similar to the way multiagent HyperNEAT learns the policies of multiple *agents*.

This paper tests the idea of training teams using both standard policy geometry and situational policy geometry to create robust multirobot teams in a patrol and return domain both in simulation and with robots in the real world. In this task robots must work together to cover an area, but must return home when called, which means that there are two similar tasks with different goals. To demonstrate the benefit of exploiting situational geometry, teams trained with situational geometry will be compared with those that are trained with the standard multiagent HyperNEAT method that does not incorporate information about the relationship among tasks.

The main conclusion is that methods that take advantage of situational policy geometry find solutions more consistently than those that cannot and that those solutions are robust enough to transfer to the real world.

II. BACKGROUND

This section reviews popular approaches to multiagent learning, highlighting several robotics applications, past work in task-switching, and the NEAT and HyperNEAT methods that form the backbone of multiagent HyperNEAT.

A. Traditional Cooperative Multiagent Learning

There are two primary traditional approaches to multiagent learning. The first, multiagent reinforcement learning (MARL), encompasses several specific techniques based on off-policy and on-policy temporal difference learning [6]–[8]. The basic principle that unifies MARL techniques is to identify and reward promising cooperative states and

actions among a team of agents [9], [10]. The other major approach, cooperative coevolutionary algorithms (CCEAs), is an established evolutionary method for training teams of agents that must work together [10]–[12]. The main idea is to maintain one or more populations of candidate agents, evaluate them in groups, and guide the creation of new candidate solutions based on their joint performance.

While reinforcement learning and evolution are mainly the focus of separate communities, Panait, Tuyls, and Luke [13] showed recently that they share a significant common theoretical foundation. One key commonality is that they break the learning problem into separate roles that are semi-independent and thereby learned separately through interaction with each other. Although this idea of separating multiagent problems into parts is appealing, one problem is that when individual roles are learned separately, there is no representation of how roles relate to the team structure and therefore no principle for exploiting regularities that might be shared across all or part of the team. Thus in cases where learning has been applied to real-world applications, it usually exploits inherent homogeneity in the task [14], [15].

The multiagent HyperNEAT approach extended in this paper addresses this challenge and is augmented to switch tasks. Prior approaches to task switching are reviewed next.

B. Prior Work in Task Switching

There are many approaches to solving problems in which agents must perform multiple tasks. One important strategy is to decompose the main task into hierarchies of subtasks [3]. In this approach agents can focus on these specific subtasks and complete them according to the hierarchy. However, the tasks must be decomposed by the experimenter. Another method is to learn modular controllers [4] wherein different parts of the controller (e.g. different sets of outputs) are active depending on the state of the robot. Evolving adaptive artificial neural networks (ANNs) is another significant technique [5], wherein local learning rules facilitate the policy transition from one task to the other.

Despite the abundance of methods, task switching is still a difficult problem, especially for cooperative multiagent learning. Such systems tend to be less robust because if a single robot fails to perform as expected the entire team can fail. The extension of multiagent HyperNEAT in this paper overcomes this obstacle. The next section reviews the Neuroevolution of Augmenting Topologies (NEAT) method, the foundation for multiagent HyperNEAT.

C. Neuroevolution of Augmenting Topologies

The multiagent HyperNEAT method that enables learning from geometry in this paper is an extension of the NEAT algorithm for evolving ANNs. NEAT performs well in a variety of control and decision-making problems [16], [17]. It starts with a population of small, simple neural networks and then increases their complexity over generations by adding new nodes and connections through mutation. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. Furthermore, it allows NEAT to establish high-level features early in evolution and then later elaborate on them.

The important property of NEAT for this paper is that it evolves *both* the topology and weights of a neural network. Because it starts simply and gradually adds complexity, NEAT tends to find a solution network close to the minimal necessary size. In principle, another method for learning the topology and weights of networks could also fill the role of NEAT in this paper. Nevertheless, what is important is to begin with a principled approach to learning both such features, which NEAT provides. Stanley and Miikkulainen [16], [17] provide a complete overview of NEAT.

The next section reviews the HyperNEAT extension to NEAT that is itself extended in this paper to generate multiagent teams.

D. HyperNEAT

A key similarity among many neuroevolution methods, including NEAT, is that they employ a *direct* encoding, that is, each part of the solution's representation maps to a single piece of structure in the final solution. Yet direct encodings impose the significant disadvantage that even when different parts of the solution are similar, they must be encoded and therefore discovered separately. This challenge is related to the *problem of reinvention* in multiagent systems, which occurs when similar skills must be learned for agents that otherwise differ: After all, if individual team members are encoded by separate representations, even if a component of their capabilities is shared, the learner has no way to exploit such a regularity. Thus this paper employs an *indirect* encoding instead, which means that the description of the solution is compressed such that information can be reused, allowing the final solution to contain more components than the description itself. Indirect encodings are powerful because they allow solutions to be represented as a pattern of policy parameters, rather than requiring each parameter to be represented individually [18]–[20]. HyperNEAT, reviewed in this section, is an indirect encoding extension of NEAT that is proven in a number of challenging domains that require discovering regularities [21]–[23], including several robotics applications [24], [25]. For a full description of HyperNEAT see Stanley et al. [22] and Gauci and Stanley [23].

In HyperNEAT, NEAT is altered to evolve an indirect encoding called *compositional pattern producing networks* (CPPNs [19]) *instead* of ANNs. CPPNs, which are also networks, are designed to encode *compositions of functions*, wherein each function in the composition (which exists in the network as an activation function for a node) loosely corresponds to a useful regularity. For example, a Gaussian function induces symmetry. Each such component function also creates a novel geometric *coordinate frame* within which other functions can reside. For example, any function *of* the output of a Gaussian will output a symmetric pattern because the Gaussian is symmetric.

The appeal of this encoding is that it allows spatial patterns to be represented as networks of simple functions (i.e. CPPNs). Therefore NEAT can evolve CPPNs just like ANNs; CPPNs are similar to ANNs, but they rely on more than one activation function (each representing a common regularity) and act as an *encoding* rather than a network.

The indirect CPPN encoding can compactly encode patterns with regularities such as symmetry, repetition, and repetition with variation [19], [26]. For example, while including a Gaussian function, which is symmetric, can cause the output or part of the output to be symmetric, a periodic function such as sine creates segmentation through repetition. Most importantly, *repetition with variation* (e.g. such as the fingers of the human hand) is easily discovered by combining regular coordinate frames (e.g. sine and Gaussian) with irregular ones (e.g. the asymmetric x-axis). For example, a function that takes as input the sum of a symmetric function and an asymmetric function outputs a pattern with imperfect symmetry. In this way, CPPNs produce regular patterns with subtle variations. The potential for CPPNs to represent patterns with motifs reminiscent of patterns in natural organisms has been demonstrated in several studies [19] including an online service on which users collaboratively breed patterns represented by CPPNs [26].

The main idea in HyperNEAT is that CPPNs can naturally encode *connectivity patterns* [22], [23]. That way, NEAT can evolve CPPNs that represent large-scale ANNs with their own symmetries and regularities. This capability will prove essential to encoding multiagent policy geometries in this paper because it will ultimately allow connectivity patterns to be expressed as a function of team geometry, which means that a smooth gradient of policies can be produced across possible agent locations.

Formally, CPPNs are *functions* of geometry (i.e. locations in space) that output connectivity patterns whose nodes are situated in n dimensions, where n is the number of dimensions in a Cartesian space. Consider a CPPN that takes four inputs labeled $x_1, y_1, x_2,$ and y_2; this point in four-dimensional space *also* denotes the connection between the two-dimensional points (x_1, y_1) and (x_2, y_2), and the output of the CPPN for that input thereby represents the weight of that connection (Fig. 1). By querying every possible connection among a set of points in this manner, a CPPN can produce an ANN, wherein each point is a neuron position. Because the connection weights are produced by a function of their endpoints, the final structure is produced with *knowledge* of its geometry. In effect, the CPPN is painting a pattern on the inside of a four-dimensional hypercube that is interpreted as the isomorphic connectivity pattern, which explains the origin of the name *hypercube-based NEAT* (HyperNEAT). Connectivity patterns produced by a CPPN are called *substrates* to verbally distinguish them from the CPPN itself, which has its own internal topology.

Each queried point in the substrate is a node in a neural network. The experimenter defines both the location and role (i.e. hidden, input, or output) of each such node. Nodes should be placed on the substrate to reflect the geometry of the task [22]–[24]. That way, the connectivity of the substrate is a function of the the task structure.

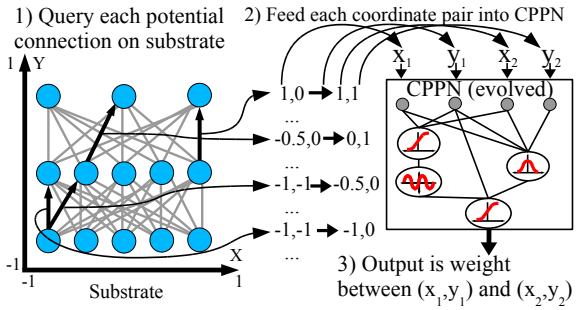


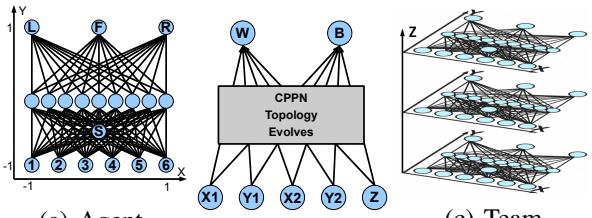
Fig. 1. CPPN-based Geometric Connectivity Pattern Encoding. A collection of nodes, called the *substrate*, is assigned coordinates that range from -1 to 1 in all dimensions. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines in the substrate depicted in the figure represent a sample of connections that are queried. (2) Internally, the CPPN (which is evolved by NEAT) is a graph that determines which activation functions are connected. As in an ANN, the connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection. For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. Thus, CPPNs can produce regular patterns of connection weights in space.

For example, the sensors of an autonomous robot can be placed from left to right on the substrate in the same order that they exist on the robot. Outputs for moving left or right can also be placed in the same order, allowing HyperNEAT to understand from the outset the correlation of sensors to effectors. In this way, knowledge about the problem geometry can be injected into the search and HyperNEAT can exploit the regularities (e.g. adjacency, or symmetry) of a problem that are invisible to traditional encodings.

In summary, the capabilities of HyperNEAT are important for multiagent learning because they provide a formalism for producing policies (i.e. the output of the CPPN) as a function of geometry (i.e. the inputs to the CPPN). As explained next, not only can such an approach produce a single network but it can also produce a set of networks that are each generated as a function of their location in space.

III. APPROACH: EXTENDING MULTIAGENT HYPERNEAT

The multiagent HyperNEAT algorithm was first introduced by D'Ambrosio and Stanley [2] and D'Ambrosio et al. [1]. However, it has never been applied to a real-world patrol task like the one in this paper and its evolved controllers have never been transferred to the real world before now. These achievements are made possible in this paper by introducing the extension of *situational policy geometry*. Thus this paper shows how the ideas of indirect encoding and policy geometry can impact real-world problems. This section summarizes the details of the standard multiagent HyperNEAT method that encodes a team based on policy geometry and then explains the extension to the method that allows multiagent HyperNEAT to exploit situational policy geometry.



(a) Agent Substrate (b) CPPN (c) Team Substrate

Fig. 2. Multiagent HyperNEAT. The CPPN and substrates that multiagent HyperNEAT employs for this paper are shown. The CPPN (b) evolves a pattern that describes the connectivity and weights of the neural network controllers for each agent on the team. The CPPN queries all possible connections in the team substrate (c), which is made up of several individual substrates (a). Each of these is located at a different z-coordinate, which represents the team's policy geometry. The S node in (a) is intentionally elevated to reflect its special status as the "come home" signal. The additional B output on the CPPN allows it to encode biases in addition to the usual connection weight output W.

A. Standard Policy Geometry

The *policy geometry* of a team is the relationship between the canonical starting positions of agents on the field and their behavioral policies. Multiagent HyperNEAT is based on the idea that policy geometry is an effective level of description for a team because it can be encoded naturally as a pattern. This section describes how multiagent HyperNEAT extends HyperNEAT to encode heterogeneous teams as a pattern of policies.

To generate a controller for a single agent, a CPPN accepts inputs x_1, y_1, x_2, and y_2 and queries the weights of all possible connections for the single controller (Fig. 2a), as described in the previous section (Fig. 1). For a single CPPN to encode a *set* of networks in a pattern, thereby exploiting policy geometry, changes must be made to both the CPPN and the substrate. In the CPPN, additional inputs are added to represent the dimensions of the policy geometry. In this paper only one additional input z is added (Fig. 2b) to give a one-dimensional policy geometry, but in principal there is no limit to the number of dimensions of the policy geometry. Additionally, the HyperNEAT substrate is composed of multiple (three in this paper) single-agent substrates stacked along the z-axis (Fig. 2c), representing the three agents in the team.

The main idea is that the CPPN is able to create a pattern based on both the agent's internal geometry (x and y) and its position on the team (z) (Fig. 2a,c). That way, each network is encoded as a function of *both* its internal geometry *and* its position (z) on the team. The CPPN can thus emphasize connections from z for increasing heterogeneity or minimize them to produce greater homogeneity. Furthermore, because z is a spatial dimension, the CPPN can literally generate policies based on their positions on the team.

The team substrate (Fig. 2c) formalizes the idea of encoding a team as a pattern of policies. This capability is powerful because generating each agent with the same CPPN means they can share tactics and policies while still exhibiting variation across the policy geometry. In other words, policies are spread across the substrate in a pattern just as role assignment in a human team forms a pattern across a field. However, even as roles vary, many skills are shared, an idea elegantly captured by indirect encoding. The complete multiagent HyperNEAT algorithm is enumerated in Algorithm 1.

Algorithm 1 Multiagent HyperNEAT

1) Set the substrate to contain the number of agents.
2) Initialize a population of minimal CPPNs with random weights that correspond to the chosen substrate.
3) Repeat until a solution is found or the maximum number of generations are reached:
   a) For each CPPN in the population:
      i) Query its CPPN for the weight of each connection in the substrate within each agent's ANN. If the absolute value of the output exceeds a threshold magnitude, create the connection with a weight scaled proportionally to the output value (Fig. 1).
      ii) Assign the generated ANNs to the agents and run the team to ascertain fitness.
   b) Reproduce the CPPNs according to the NEAT method to create the next generation's population.

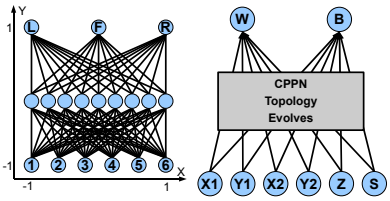
B. Situational Policy Geometry

Situational policy geometry allows agents to learn to switch to different policies depending on their current state. That way, not only can multiagent HyperNEAT exploit similarities among tasks, but the solutions for individual subtasks are likely to be simpler (and thus more easily discovered) than a single policy that must solve all tasks.

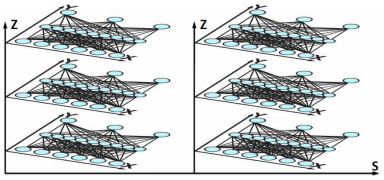
To exploit situational policy geometry the CPPN must be made aware of it. Thus new inputs are added to the CPPN that represent the dimensions of the tasks (Fig. 3b), similarly to how inputs are added to the CPPN to represent the standard policy geometry that represents dimensions of the team (Fig. 2b). For example, in this paper a single new dimension S is added, which is either 1 or -1, depending on whether the robot must come home or not. Because the signal is now a part of the CPPN (Fig. 3b), the controller for an individual robot (Fig. 3a) does not need the signal input. Also, because the CPPN now has an extra dimension, there are now two stacks of controllers (one for each value of the signal) instead of one (Fig. 3c). In effect, each agent now has two brains that it can switch between depending on their current task.

IV. PATROL AND RETURN EXPERIMENT

To demonstrate that multiagent HyperNEAT can produce teams that are robust enough to function in the real world and to explore the capabilities of situational policy geometry, teams are evolved to solve a patrol and return task in which robots must spread out and observe an environment and then return home when signaled to do so. Patrolling tasks are



(a) Agent Substrate (b) Situational CPPN



(c) Situational Team Substrate

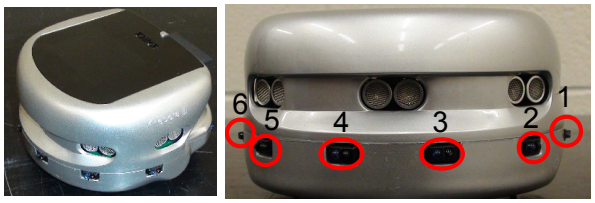
Fig. 3. Multiagent HyperNEAT with Situational Policy Geometry. The CPPNs and substrates that multiagent HyperNEAT employs for exploiting situational policy geometry in this paper are shown. The main difference between this situational setup and the standard setup in Fig. 2 is that the CPPN (b) has an additional input S that describes the location in the situational policy geometry, which is formalized in the new S-axis as the situational team substrate (c). Thus the network stack to the left along S is team policies for one situation while that on the right is policies for a different situation.

common in multiagent learning [8], [27] because they require agents to cooperate to ensure that they do not collide with each other and to achieve uniform coverage of the area. The task in this paper is made more complex by the fact that the robots must return home on command, meaning that each agent must effectively learn two roles and remember how to return home. This requirement also makes the task more realistic: If a group of robots were sent to patrol a building, recalling them after a period of time to recharge batteries or when the patrol is over would be preferable to manually collecting the robots.

Unlike other approaches to multiagent patrolling [27], [28] the robots in this task cannot communicate with each other. This limitation means that the agents must learn *a priori* roles to maximize coverage and minimize overlaps. By exploiting the policy geometry, multiagent HyperNEAT can accomplish this goal by finding a general patrolling and collision-avoidance policy for all the agents, and at the same time by varying the policy for each agent so that they patrol different areas. Also, the patrolling robots must respond to a "come home" signal, requiring a robust dual policy that keeps them deployed until the signal, at which point they must quickly return home. By exploiting situational policy geometry, the idea is that robots can employ separate yet related policies for these conflicting tasks.

A. Robots

The robots used in these experiments are three Khepera IIIs outfitted with KoreBot II extensions (Fig. 4a), which make it possible for the neural network controllers to run on the robots themselves, thereby minimizing command latency and reducing the need for communication with a base station to only general broadcast signals (i.e. start and return). The



(a) Khepera III (b) Front Sensors

Fig. 4. Khepera III with Korebot II. The Khepera III mobile robots (a) in these experiments come equipped with a Korebot II extension that runs an embedded Linux operating system and allows the robots to receive broadcast communications over a wireless network. Although the Khepera III has many sensors available, only the front six infrared rangefinders (b) are utilized in these experiments.

Khepera III is equipped with both long-range, ultrasonic and short-range infrared rangefinder sensors; however for this task only the front six infrared sensors are utilized (Fig. 4b). The sensors can detect both walls and robots, but *cannot* distinguish between them. The Khepera III can achieve speeds of up to 30cm/sec, but because of the size of the environments and to avoid damage to the robots during testing the motors were run at a reduced speed with an approximate velocity of 6cm/sec. For more information on the Khepera III see `http://www.k-team.com/`.

Each robot is controlled by a separate neural network (either Fig. 2a or Fig. 3a depending on whether or not they are learning situational policy geometry) generated by the same CPPN (either Fig. 2b or Fig. 3b). If the robot is using situational policy geometry, it has six input nodes corresponding to the six rangefinder sensors. The robots without situational policy geometry have one additional input (called S in Fig. 2a) that indicates whether the robot should return home or continue patrolling. The rangefinders on the robot return values between 0 and 4,000; larger numbers represent farther distances. Preliminary experiments indicated that the sensor values beyond four inches are very noisy and that the response curve of the sensors is non-linear. Therefore, before the values are fed into the neural network, the raw sensory input is modified to clip values beyond four inches, to be more linear, and to be scaled between zero and one through the function $1.43 - \frac{(\log(s)-0.51)}{2.25}$, where s is the raw sensor value returned by the robot.

A robot can select from one of three actions: go forward, turn left, or turn right. The action is selected based on the values of the three network outputs in Fig. 2a or Fig. 3a; the output with the highest value is the action for that timestep. The robots are allowed to select actions every 33 milliseconds because that is the update rate of the Khepera III's infrared sensors. Robots also have a collision avoidance policy to reduce the chance of damaging themselves that overrides the robot's forward command: If either of the two leftmost sensors fall below 0.25, the robot turns right; the opposite is true for the two rightmost sensors and if either of the front sensors fall below the threshold, the robot stops.

Training teams through artificial evolution with real robots would be time consuming and could potentially damage

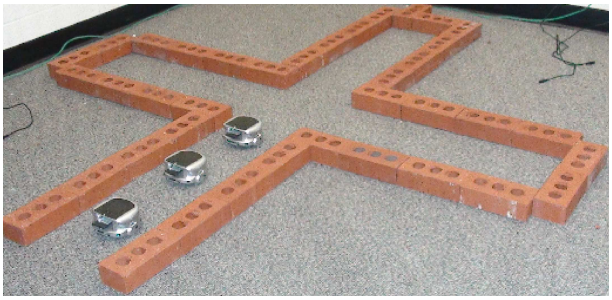
the robots. Instead the teams are trained in a custom simulator made in our research group (available at: `http://eplex.cs.ucf.edu/software.html`). Only simple two-dimensional kinematics are simulated with an update rate of 33msec. This approach is faster than modeling and simulating three-dimensional motion and/or realistic physics, and was nevertheless found to be just as accurate as transferring from e.g. Webots [29] in real-world transfer in preliminary experiments. The Khepera IIIs are modeled based on manufacturer specifications and preliminary calibration experiments.

B. Environments

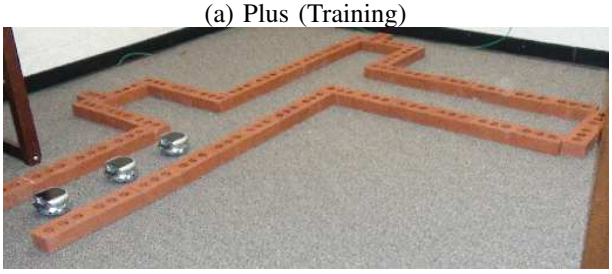
Each team is trained in the same environment to encourage robots to adopt specific roles, but in the real world they are also tested on a variant of that environment to ensure generality. The training environment is called *the plus* (Fig. 5a) and is made of an entrance that leads to three branching paths. Branches are approximately 29cm wide and 77.5cm long. To cover this environment, the robots must split up and take separate paths, even though they cannot communicate with each other. Thus the robots must have some *a priori* bias that allows them to cooperate. The testing environment is called *the asymmetric plus* (Fig. 5b) and is similar to the plus, but with several changes. First, the left path is shortened to approximately 48.43cm and the right and center paths are 1.5 times as long as in the regular plus. Also, the right branch is shifted up by 77.5cm These changes cause very different sensor activations where robots would typically turn and stop, thereby testing the generality of the learned policies. The environments are designed to capture the general idea of patrolling, while not being too complex to build physically (out of bricks) in the real world.

For the real robots, the environments are constructed out of red $7\frac{5}{8}$in$\times 3\frac{5}{8}$in$\times 2\frac{1}{4}$in bricks with a carpet base, which are the same dimensions as in the simulator. The three robots are placed in the starting branch of the environment, 30cm apart. They are then simultaneously started and begin patrolling. A good solution is for all agents to reach the end of a different branch and stop. After all agents are stopped, they are called back by activating their "come home" signal in the order that they left. Only one agent is called back at a time to maintain as much coverage as possible. When the agent returns home, its signal is turned off and it is placed back at the home point facing the environment so that it can return to patrolling and the next robot can be called back.

In simulation fitness is assigned to each team based on two criteria: If a robot receives the signal to come home, minimizing distance to home is rewarded, but if it does not yet have the signal, minimizing distance to the end of a hall is rewarded. For every simulated second each robot is given a score of $\frac{D-d}{D}$, where D is the maximum possible distance to either the end of a hall or home, depending on the state of that robot's signal, and d is the current distance to that objective. If the robots have not reached the end of the hall when the signal activates, their fitness for returning home is divided by ten, so that solutions that never leave home are discouraged. Similarly, teams in which all agents do not change position or heading after they receive the signal or were still moving forward when they received it have their fitness for patrolling divided by 10 to encourage them to respond to the signal. These scores are summed over each robot over each second (out of 45) to give the overall fitness of a team for that trial. Thus the maximum fitness is three times the number of seconds of the trial, although in practice such a fitness is not possible to reach because the robots spend time moving between points. To simplify training, evaluations in simulation are carried out slightly differently than in the real world: Instead of calling the robots one by one, all robots are called simultaneously when half of the evaluation time has passed and robot-to-robot sight and collision are turned off. This method tests the essential requirements of the policies to return home, while speeding up evaluation significantly.



(a) Plus (Training)



(b) Asymmetric Plus (Testing)

Fig. 5. Real-World Environment. The real environments with which the robots interact are constructed out of red bricks on a carpet with the same dimensions used in the simulator. The plus (a) is the environment the robots are trained on, and the asymmetric plus (b) tests the generality of the learned policies. In both cases robots are placed 30cm apart in the open branch and then sent a signal to begin patrolling. Individual robots can then be called back by the experimenter by broadcasting a command to them.

C. Experimental Parameters

Because HyperNEAT differs from original NEAT only in its set of activation functions, it uses the same parameters [16]. Both experiments were run with a modified version of the public domain SharpNEAT package [30]. The size of each population was 500 with 20% elitism. The number of generations was 1,000. Sexual offspring (50%) did not undergo mutation. Asexual offspring (50%) had 0.96 probability of link weight mutation, 0.03 chance of link addition, and 0.01 chance of node addition. The coefficients for determining species similarity were 1.0 for nodes and connections and 0.1 for weights. The available CPPN ac-

tivation functions were sigmoid, Gaussian, absolute value, and sine, all with equal probability of being added to the CPPN. Parameter settings are based on standard SharpNEAT defaults and prior reported settings for NEAT [16], [17]. They were found to be robust to moderate variation through preliminary experimentation.

V. RESULTS

Fifteen independent runs of multiagent HyperNEAT were conducted in the simulated plus environment with situational policy geometry and with standard policy geometry. In the simulator, a *solution* to an environment is a policy that successfully navigates each of the three robots to the end of a different wing of the map, where they each wait until a signal is sent to them, and then navigate back to the starting location after the signal is received. In each of the fifteen runs with situational policy geometry a solution was evolved in 264.6 generations on average (stdev=246.28). However, with standard policy geometry only three solutions were evolved. This difference is significant ($p < 0.0001$; Fischer's exact test), highlighting the advantage that can be gained from recognizing and exploiting situational regularities.

Note that evolution did not stop when the first solution was found, so each successful run actually produced a number of viable solutions. To determine which solutions from these successful runs to evaluate in the real world, a generalization test was developed. This test averages the performance of an evolved policy in 25 additional evaluations on the plus environment with varying levels of noise in the robots' sensors, stochastic turning and locomotion, and small random perturbations of the initial location and heading of the robots. The idea is that the policies that are more general will perform better in the real world because they will be more robust to the inevitable slight discrepancies between an imperfectly modeled simulated environment and reality.

Confirming this motivation, the five most general solutions from distinct runs with situational policy geometry were all successfully transferred from simulation to the real world, where the solution criteria was even more strict to make sure teams are genuinely robust in real robots without further training: Each robot must go out to its proper position, return home upon receiving the signal, and then return back to its position. In addition, when tested in the real world in the asymmetric plus environment, for which they were *not* trained, these five most general solutions still successfully patrolled and returned, thereby demonstrating that the policies learned by multiagent HyperNEAT with situational policy geometry were not specific to a single map. Videos of such successful transfers from simulation to both the real world plus and asymmetric plus environments are available at: `http://eplex.cs.ucf.edu/patrolling.html`. Of the three runs without situational policy geometry that solved the task, two transferred successfully to the real-world symmetric environment, and only one solved the asymmetric environment. Because so few runs could solve the task at all without situational policy geometry even in simulation, the sample size is too small to draw significant conclusions on transferability of such solutions. However, because the chance of even finding a solution is statistically so much smaller without situational policy geometry, in effect if the aim is to find a real-world solution, situational policy geometry provides a significant advantage.

VI. DISCUSSION

Teams that were trained with situational policy geometry outperformed those trained without it in both simulation and in the real world. One reason for this advantage is that by giving multiagent HyperNEAT situational policy geometry information, it was able to exploit the regularities of the tasks. For example, a major difference between leaving to patrol and coming back is the direction the robot must turn. Agents without situational policy geometry must utilize the value of a specific input to decide how to turn, but those with it utilize a different neural network once the signal fires. A common strategy for situational teams was simply to invert connection weights in the network, allowing them to keep a similar policy, but with an opposite turning bias.

Because they can exploit situational regularities, the policies of the teams with situational policy geometry were simpler than those represented by the teams with standard policy geometry. That is, the agents without situational policy geometry must effectively encode in the same network both how to perform the tasks and how to switch between them. Thus, it is possible for the robot to encounter situations (e.g. those conflated or obscured by noise in robot sensors) that cause it to switch tasks when it is not appropriate. In fact, a frequent failure of these teams in the real world was that they would come back too early or not come back at all. In contrast, the situational teams with their simpler task policies did not exhibit this behavior and were able to transfer consistently to real robots in both the training and testing environments. Thus these experiments verify the utility of breaking up complex tasks into simpler subtasks as in previous work and offer a new method by which to learn these subtasks that exploits the regularities among them.

Another consequence of this work relates to generative and developmental systems such as HyperNEAT. These systems rely heavily on discovering and exploiting the regularities of a problem such as how the leftmost sensor relates to the left turn effector. However, sometimes there is information that is critical to a problem that does not easily fit into these patterns, such as the "come home" signal in this paper: There is no simple geometric relationship between the signal input and the sensors and effectors, so it is unclear where exactly it should be placed on the substrate to best exploit the geometry of the problem. By moving the signal to the CPPN, this information is effectively incorporated into the pattern without disrupting the existing geometry. Thus situational policy geometry opens up a new possibility for indirect encodings wherein information that does not clearly fit with existing patterns can still be elegantly incorporated into the encoding.

A future direction for this work is to investigate more complex domains with larger numbers of tasks or subtasks.

Multiagent HyperNEAT should be able to discover the relationships between varying numbers of tasks just as it is able to do so among varying numbers of agents [1]. More tasks can be added by either increasing the sampling rate of a single dimension of situational policy geometry or by introducing new dimensions, depending on the relationships of the tasks. Another intriguing possibility is to allow the agent to decide when to switch between tasks through an output that could either determine when an agent wants to switch tasks, or which task (i.e. S-coordinate) the agent wants to perform. In this way, a continuum of tasks could be automatically generated by sampling intermediate S-coordinates, allowing agents to discover new and interesting ways to divide work and cooperate.

VII. CONCLUSION

This paper presented an extension to the multiagent HyperNEAT algorithm and standard policy geometry called situational policy geometry. The new approach allows each agent to encode multiple policies that can be switched depending on the agent's state, which are learned as a function of their relationship to each other. These novel capabilities were tested in a patrol and return task, demonstrating that exploiting situational policy geometry leads to teams that find efficient solutions more consistently and that these solutions transferred to real Khepera III robots.

VIII. ACKNOWLEDGMENTS

This research was supported by DARPA under grants HR0011-08-1-0020 and HR0011-09-1-0045 (Computer Science Study Group Phases I and II).

REFERENCES

[1] D. B. D'Ambrosio, J. Lehman, S. Risi, and K. O. Stanley, "Evolving policy geometry for scalable multiagent learning," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 2010, pp. 731–738.

[2] D. B. D'Ambrosio and K. O. Stanley, "Generative encoding for multiagent learning," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*. New York, NY: ACM Press, 2008.

[3] R. Makar, S. Mahadevan, and M. Ghavamzadeh, "Hierarchical multiagent reinforcement learning," in *Proceedings of the fifth international conference on Autonomous agents*, ser. AGENTS '01. New York, NY, USA: ACM, 2001, pp. 246–253.

[4] S. Nolfi, "Using emergent modularity to develop control systems for mobile robotics," *Adaptive Behavior*, vol. 3–4, pp. 343–364, 1997.

[5] D. Floreano and J. Urzelai, "Evolutionary robots with on-line self-organization and behavioral fitness," *Neural Networks*, vol. 13, pp. 431–4434, 2000.

[6] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: theoretical framework and an algorithm," in *Proc. 15th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1998, pp. 242–250.

[7] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

[8] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," *Autonomous Agents and Multiagent Systems, International Joint Conference on*, vol. 3, pp. 1122–1129, 2004.

[9] L. Busoniu, B. D. Schutter, and R. Babuska, "Learning and coordination in dynamic multiagent systems," Delft University of Technology, Tech. Rep. 05-019, 2005.

[10] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 11, pp. 383–434, November 2005.

[11] L. Panait, R. Wiegand, and S. Luke, "Improving coevolutionary search for optimal multiagent behaviors," *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 653–658, 2003.

[12] S. Ficici and J. Pollack, "A game-theoretic approach to the simple coevolutionary algorithm," *Lecture notes in computer science*, pp. 467–476, 2000.

[13] L. Panait, K. Tuyls, and S. Luke, "Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective," *The Journal of Machine Learning Research*, vol. 9, pp. 423–457, 2008.

[14] N. Correll and A. Martinoli, "Collective inspection of regular structures using a swarm of miniature robots," in *Experimental Robotics IX*, ser. Springer Tracts in Advanced Robotics, M. Ang and O. Khatib, Eds. Springer Berlin / Heidelberg, 2006, vol. 21, pp. 375–386.

[15] M. Quinn, L. Smith, G. Mayley, and P. Husbands, "Evolving controllers for a homogeneous system of physical robots: Structured cooperation with minimal sensors," *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1811, p. 2321, 2003.

[16] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[17] ——, "Competitive coevolution through evolutionary complexification," *Journal of Artificial Intelligence Research*, vol. 21, pp. 63–100, 2004.

[18] J. C. Bongard, "Evolving modular genetic regulatory networks," in *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

[19] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, vol. 8, no. 2, pp. 131–162, 2007.

[20] K. O. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial Life*, vol. 9, no. 2, pp. 93–130, 2003.

[21] P. Verbancsics and K. O. Stanley, "Evolving static representations for task transfer," *Journal of Machine Learning Research*, pp. 1737–1763, 2010.

[22] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based indirect encoding for evolving large-scale neural networks," *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009.

[23] J. Gauci and K. O. Stanley, "Autonomous Evolution of Topographic Regularities in Artificial Neural Networks," *Neural Computation Journal*, 2010, to appear.

[24] J. Clune, B. E. Beckmann, C. Ofria, and R. T. Pennock, "Evolving coordinated quadruped gaits with the hyperneat generative encoding," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Section on Evolutionary Robotics*. Piscataway, NJ, USA: IEEE Press, 2009.

[25] E. Haasdijk, A. Rusu, and A. Eiben, "HyperNEAT for Locomotion Control in Modular Robots," *Evolvable Systems: From Biology to Hardware*, pp. 169–180, 2010.

[26] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, "Picbreeder: Evolving pictures collaboratively online," in *CHI '08: Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM, 2008, pp. 1759–1768.

[27] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, "Recent advances on multi-agent patrolling," in *Advances in Artificial Intelligence SBIA 2004*, ser. Lecture Notes in Computer Science, A. L. C. Bazzan and S. Labidi, Eds. Springer Berlin / Heidelberg, 2004, vol. 3171, pp. 126–138.

[28] J.-S. Marier, C. Besse, and B. Chaib-draa, "Solving the continuous time multiagent patrol problem," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 941 –946.

[29] Webots, "http://www.cyberbotics.com," commercial Mobile Robot Simulation Software. [Online]. Available: http://www.cyberbotics.com

[30] C. Green, "SharpNEAT homepage," http://sharpneat.sourceforge.net/, 2003–2006.