Learning Distributed Control for Modular Robots

Paulina Varshavskaya, Leslie Pack Kaelbling and Daniela Rus
MIT CSAIL
Cambridge, MA, USA
{paulina,lpk,rus}@csail.mit.edu

***Abstract*— We propose to automate controller design for distributed modular robots. In this paper, we present some initial experiments with learning distributed controllers for synthesizing compliant locomotion gaits for modular, self-reconfigurable robots. We use both centralized and distributed policy search and find that the learning approach is promising, as locomotion tasks are learnt well. We also find that the additive nature of the robotic platforms can help speed up learning if we increase the robot size incrementally.**

I. Introduction

A key challenge in self-reconfiguring modular robots is controller design. The promise of robustness and self-organization requires fully distributed control algorithms. Distributed controllers such as those in [2] took hours of designer time to synthesize, as it is hard to keep track of many interacting pieces. Furthermore, while hand-designed controllers have been shown to work well for simple tasks, it is not clear how easy it is to extend the methodology to more complex problems such as general shape synthesis and repair. We believe that reinforcement learning (RL) is a way of providing automated support for these kinds of tasks. We present a first study of this idea in the context of the gait design problem for modular self-reconfigurable robots. Thus we can compare automatically designed controllers to human-designed ones to evaluate the performance of our learning methods.

RL is appealing as we only need to provide the robot with a reward signal, rather than examples of correct behaviors. We investigate a distributed reinforcement learning approach where each module of the robot is able to learn its local behaviors (henceforth known as policy) that lead to a global system goal.

This work builds on a sizeable body of research in multi-robot learning as well as in (hand-developed) control for self-reconfiguring robots.

A. Self-reconfigurable modular robotics

Several interesting self-reconfiguring robot systems have been built [6], [12], [16]. A review of this work is available in [13]. Controllers for all of those systems were designed by hand. In most cases, the physical robot did not perform any task beyond very simple locomotion. On the other hand, in simulation more complex results have been achieved. As an example, in [2] sets of rules resembling cellular automata are developed for compliant locomotion over obstacle fields. Sets of up to 22 rules long are presented, painstakingly designed and proven to be correct for the task.

B. Multi-robot learning

While reinforcement learning seems intuitively desirable as a means of devising robotic controllers, it has proven a hard problem. Robots are noisy and only observe certain parts of their environment. Some researchers have addressed specifically the problem of partial observability in their robotic systems [3]. There has been an effort in applying learning techniques to multi-robot tasks. A RL approach with shaped rewards is used in [10] on a team of robots who must avoid other robots and compete with them to deliver pucks to a home base. Centralized or partially centralized learning algorithms for distributed robots have been proposed in [4]. Distributed algorithms for a small state-space, such as adjusting weights for choice of hand-coded behaviors, were also proposed [8]. Teams learn communicative acts in [5] using Q-learning, and the same algorithm with a twist is applied in [17] on floating robotic links who learn to navigate to a goal. The approach, also taken in this paper, of following the policy gradient is based in part on the theory of [1], which was recently also applied to optimization problems in a distributed sensor network [11].

In this paper, our goal is to do a computational evaluation of the use of RL and in particular, gradient ascent policy search, for the task of synthesizing compliant locomotion gaits for robots with very high degrees of freedom (DOFs). We examine three RL algorithms that use different resources: a centralized algorithm, a distributed algorithm, and an incremental algorithm. We compare these algorithms to other learning methods and to hand-designed controllers and show experimentally that the policy learned with our approach is competitive.

II. A Simulator for Learning of Control

We have created a simulated world, which is currently 2D. The physical laws of the simulator consist of gravity: things fall to the horizontal axis or first available surface if suspended above it, collision detection to prevent objects from moving through each other, and a rule explicitly disallowing disconnection of modules. There is no momentum and objects resting on the ground are fixed with respect to the ground. At least such a primitive physics is needed for our simulated world. As our work focuses on learning the controllers, the robot will explore bad actions that should fail.

In this simulated world, our 'robot' consists of a number of identical modules, represented by dots, as in figure 1. The world may contain obstacles. We make the following

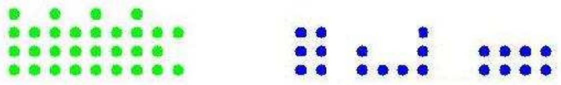


Fig. 1. 2D simulation of self-reconfigurable motion. The modules (left) are approaching an obstacle course (right).

assumptions about each module represented in our simulation:

- that it has sufficient actuated degrees of freedom, which allow it to move over a substrate of its peers
- that it cannot move by itself, without being attached to other modules
- that it has enough computational power and memory to effectively run the learning algorithms.

In addition, we assume that our distributed system is synchronous. Modules take turns computing and executing actions and no module can do so twice before all others have taken a turn.

The task to which we put our simulated robot is locomotion in one direction (eastward) with or without obstacles. We assume that the sensing or communication apparatus of each module allows it to observe whether or not other modules or obstacles are present in its immediate Moore neighborhood (8 nearest neighbors for a neighborhood of radius 1). This neighborhood condition makes up the observation of each module at each discrete time step.

A pertinent question is whether our simulated world can be effectively modelled as a Markov Decision Process (MDP) or a partially observable MDP (POMDP). A multi-agent MDP is a 4-tuple $< S, A, T, R >$, where S is the finite set of all possible states and $s \in S$ is known by all agents, A is the finite set of actions that all agents can take, $T : S \times A^n \rightarrow P(S)$ is the transition function (n is the number of agents and $P(S)$ a probability distribution over S), and $R : S \times A^n \rightarrow \mathbf{R}^n$ is the reward function. The two key conditions are that the functions T and R depend only on the current state and current action, and are independent of any history of the process or any hidden variables. A multi-agent POMDP is a 5-tuple $< S, O, A, T, R >$, where the world state $s \in S$ is not available to agents. Instead they observe something about it $o_i = O_i(s)$. The transition probabilities $T : S \times A^n \rightarrow P(S)$ and rewards $R : S \times A^n \rightarrow \mathbf{R}^n$ now may depend on the full history of observation-action pairs for all agents, due to the partial observability of state.

In our simulated world, each learning agent (i.e., each robotic module) has a finite observation space (2^8 in the simple case, 3^8 in the case with obstacles) and a finite set of actions (N, NE, E, SE, S, SW, W, NW and a NOP). One agent's observation is only part of the state of the world. We reward eastward movement and punish actions leading to movement West. Unsuccessful, physically impossible actions are not punished, they are simply not executed.

The transition function is also complex because multiple agents are learning at the same time. We must distinguish the transition functions for learning and for acting in the world. During each learning time step, a module senses its current state, takes an action according to its current stochastic policy, executes it in the world and gets a reward. For some algorithms, in order to update its policy, the agent must know what new state it ended up in as a result of executing the action. That state s' may well be different from the state s'', which the same agent will see on the next time step when all other agents have also taken their turns in the loop. We can see that while each agent is pretending to solve an MDP, the world it acts on has partially observable characteristics due to the other agents present. Their exploration of this world does not depend only on the actions they take but also on the actions the other modules take, which they cannot observe. In addition, each module is learning its policy at the same time, meaning that everyone's behavior is non-stationary. We therefore have a non-stationary partially observable world, and we can reasonably expect that learning algorithms designed for MDPs will in general fail in our case.

III. Learning Algorithms

In this section we present the learning algorithms that we have adapted and used on this new problem. As we determined our learning problem to be partially observable, we take the approach of direct search in parameterized policy space. The algorithm we have taken as the basis for our research is Gradient Ascent in Policy Space (GAPS) [14]. We have explored the centralized and the distributed versions of the basic algorithm as well as applying it to the incremental case. While the idea of policy search and the GAPS algorithm are not novel, we are applying them to a new problem in learning on robots, against the backdrop of very few such attempts.

A. Basic GAPS with lookup table

The GAPS algorithm, taken directly from [14] without modifications, does hill-climbing in policy value space. The policies are parameterized using Boltzmann's law (where $o(t)$ is one agent's observation at time t and $a(t)$ is one agent's action at time t):

$$\pi(o, a, \theta) = Pr[\, o(t) = o, a(t) = a \mid \theta \,] = \frac{e^{\theta_{oa}/\tau}}{\sum_{a'} e^{\theta_{oa'}/\tau}}$$

Each parameter θ_{oa} corresponds to an observation-action pair. The parameters are stored in a lookup table to which all agents have access. The value of a policy is the expected reward received by agents at the end of a learning episode. Following the derivation in [14] we can estimate the policy value gradient by running the policy and observing the experienced histories and rewards. This is a centralized learning algorithm (1) for the situation where observations and actions are distributed. It keeps counts of each visited observation and observation-action pair for all agents, and updates parameters in the direction of increasing policy value gradient.

The reward signal for our locomotion task is the robot's center of mass along the x axis at the end of each episode. As the task is episodic, there is no need to discount rewards.

Algorithm 1 GAPS (State-space S, Actors M)

```
Initialize parameters θ ← 0
for each episode do
  Calculate policy π(θ)
  Initialize observation counts N ← 0
  Initialize observation-action counts C ← 0
  for each timestep in episode do
    for each actor m do
      observe m's state o_m and increment N(o_m)
      choose a from π(o_m, θ) and increment C(o_m, a)
      execute a
    end for
  end for
  Get global reward R
  Update θ according to
  θ(o,a) += α R (C(o,a) − π(o,a,θ) N(o))
  Update π(θ) using Boltzmann's law
end for
```

Algorithm 2 DGAPS (State-space S, Actor m)

```
Initialize parameters θ_m ← 0
for each episode do
  Calculate policy π(θ_m) from θ_m
  Initialize state counts N_m ← 0
  Initialize state-action counts C_m ← 0
  for each timestep in episode do
    observe m's state o and increment N_m(o)
    choose a from π(o, θ_m) and increment C_m(o, a)
    execute a
  end for
  Get local reward R_m
  Update θ_m according to
  θ_m(o,a) += α R_m(C_m(o,a) − π(o,a,θ_m)N_m(o))
  Update π(θ_m) using Boltzmann's law
end for
```

Algorithm 3 IGAPS (State-space S, Actors M), θ

```
if θ not given then
  Initialize θ ← 0
end if
n ← 1
repeat
  Increase n
  Run GAPS body with n actors and θ
until n == size(M)
Return θ
```

B. Distributed GAPS

In the case where each agent not only observes and acts but also learns its own parameterized policy, the algorithm can be extended in the most obvious way to Distributed GAPS (DGAPS), as was also done in [14]. It has been proven that DGAPS makes exactly the same parameter updates as centralized GAPS in cooperative scenarios, so long as all agents get the same reward. In our domain of a distributed modular robot, DGAPS (2) is naturally preferable. However, instead of requiring an identical reward signal for all agents, we take each module's displacement along the x axis to be its reward signal: $R_m = x_m$, since we assume that modules do not communicate. This means that the policy value landscape is now different for each agent. However, the agents are physically coupled with no disconnections allowed. If the true reward is $R = \frac{\sum_{m=1}^{N}[x_m]}{N}$, and each individual reward is $R_m = x_m$, then each R_m is a bounded estimate of R that's at most $N/2$ away from it (in the worst-case scenario where all modules are connected in one line). Furthermore, as each episode is initialized, modules are placed at random in the starting configuration of the robot. Therefore, the robot's center of mass and $R = E[x_m]$ is the expected value of any one module's position along the x axis. We can easily see that in the limit, as the number of turns per episode increases and as learning progresses, each x_m approaches the true reward.

This estimate is better the fewer modules we have and the larger R is. Therefore it makes sense to simplify the problem in the initial phase of learning, while the rewards are small, by starting with fewer modules, as we demonstrate below.

C. Incremental GAPS

It is possible to leverage off the modular nature of our problem in order to improve the convergence rate of both GAPS and DGAPS. We notice that the learning problem is easier when the robot has less modules than the size of its neighborhood, since it means that each module will see and have to learn a policy for a smaller number of states. If we start with only two modules, and add more incrementally, we effectively reduce the problem state-space. With only two modules, given the physical coupling between them, there are effectively only four states to explore. Adding one other module, we add another nine possible observations and so forth. The problem becomes more manageable. Therefore, we propose the Incremental GAPS (IGAPS) algorithm (3), which takes as input the policy parameters to which a previous running instance of IGAPS with fewer modules has converged.

As will be seen in section V, we obtain a significant advantage when using this incremental strategy.

IV. Alternative Design Methods

The alternative to automated design by learning is hand-coding, which has been extensively used to date. While an obvious choice for toy domains, design by hand becomes tedious and time-consuming when the task complexity increases.

Alternative automated design methods might include other learning algorithms, in particular other RL algorithms, some of which have been applied to robotics previously. Q-learning [15] is probably the most popular online model-free RL technique. It has nice convergence properties when the learning problem is a fully observable MDP. As we have seen earlier, our world can only be modelled as a POMDP, and in those cases there are no longer any guarantees of the algorithm converging due to a very strong Markov assumption.

In practice, researchers have applied this method to partially observable robotic domains and even obtained good convergence results, such as in [5], [17]. However,

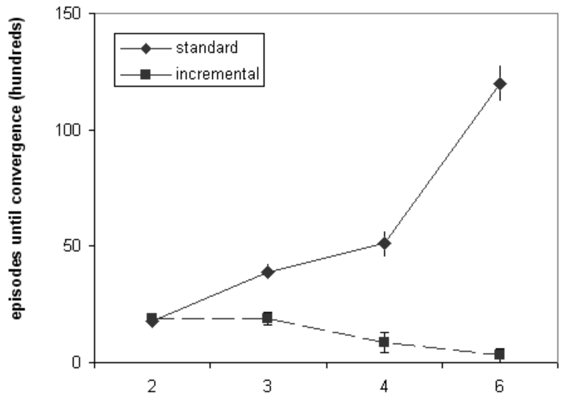


Fig. 2. Convergence time in hundreds of episodes as a function of the number of modules.

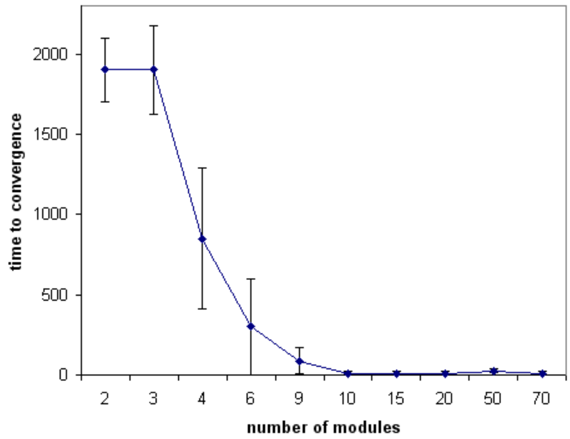


Fig. 3. Convergence times for the incremental approach.

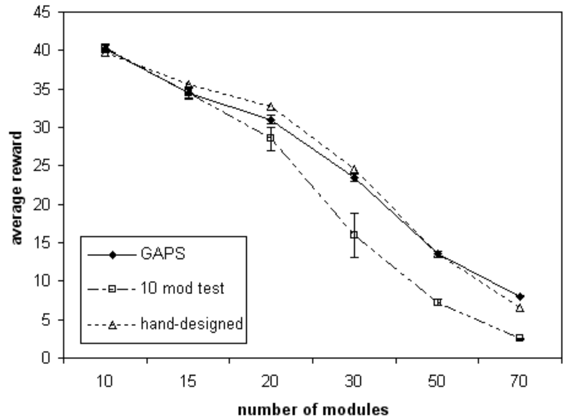


Fig. 4. Average rewards obtained by IGAPS, the best hand-designed policy, and the policy obtained by IGAPS on 10 modules. Rewards decrease because locomotion on a substrate of peers takes longer as the number of modules increases.

such results are only empirical, and nothing can be said in general about these applications. Alternatively, the results are from hybrid systems where other techniques were used to disambiguate the state, such as in, e.g., behavior selection using Q-learning [7], [9]. In section V we compare our policy search results with Q-learning and find empirically that, predictably, the latter oscillates (does not converge) most of the time even for simple instances of our problem. In table I the reward achieved by Q-learning is marked by * because the algorithm does not converge predictably.

V. Experimental Results

A. Centralized versus distributed learning

When one agent learns from the combined experience of all modules' observations, actions and rewards, it is not surprising that the learning algorithm converges faster than in the distributed case, as the learner has more data on each episode. We can see this effect in table I. However, in practice we would like to implement learning in a completely decentralized fashion, as in DGAPS. Sharing data among distributed learning agents can still be achieved with the use of near-neighbors communication, which we relegate to future work.

B. Incremental addition of modules

Figure 2 shows the convergence rates of the GAPS algorithm. On the "standard" curve we plot mean convergence times for 2, 3, 4 and 6 modules, where each of them start from the initial parameters θ set to 0 (all actions equally likely). On the "incremental" curve, whenever modules are added, the parameters to which the previous number of modules have converged are taken as a starting point. We observe a dramatic decrease in convergence time. Table I, where mean convergence times and mean rewards achieved after convergence are shown for all the algorithms shows a similar result. Also, observe in figure 3 that once the robot size has reached the number of modules in its immediate Moore neighborhood, the learner's convergence times stay steadily low.

C. Incremental increase in task complexity

Often the task structure is reflected in the robot's observations. For examples, these will be different for the modules in the front depending on whether there are obstacles present in their path. We can take advantage of that structure of the observation space by extending our incremental strategy to task complexity. However, we find that the naive approach of learning obstacle-free locomotion then adding obstacles after the algorithms has converged, does not work well. In future work, we will need to carefully design an incremental strategy for task complexity.

D. Comparison to hand-designed controllers

The structure of the reward signal used during the learning phase determines what the learned policies will do to achieve maximum reward. In the case of eastward locomotion the reward does not depend on the shape of the modular robot, only on how far east is has gone at the end of an episode. On the other hand, the hand-designed policies of [2] were made to maintain the roughly square or cubic overall shape of the robot. It turns out that one

TABLE I

COMPARISON BETWEEN LEARNING ALGORITHMS AND WITH HAND-DESIGNED POLICIES. [1]

policy	convergence time	std	mean reward
From [2]	-	-	13.1 (33%)
Hand-designed	-	-	37.3 (93%)
Distributed Q-Learning	oscillates	-	*32.9 (82%)
GAPS	10,000	1,500	37.7 (94%)
Distributed GAPS	16,000	2,200	35.5 (89%)
IGAPS (from 6 to 9 mods)	90	80	37.6 (94%)
IGAPS (cumulative)	5,000	1,100	37.6 (94%)

TABLE II

HAND-DESIGNED RULES FOR EASTWARD LOCOMOTION.

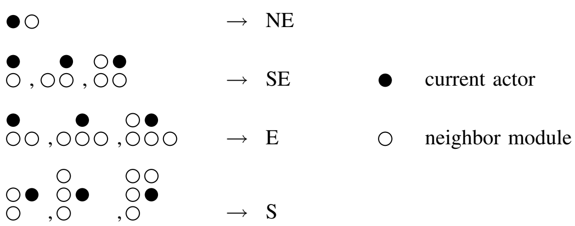
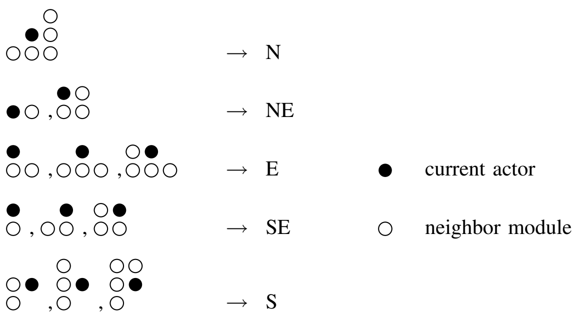


TABLE III

LEARNED RULES FOR EASTWARD LOCOMOTION: A LOCAL OPTIMUM.



can go farther faster if this constraint is relaxed, as can be seen in table I. The shape-maintaining hand-designed policy achieves an average reward per episode of 13.1, whereas its counterpart learned using DGAPS achieves an average reward of 35.5 (37.3 using GAPS).

A hand-designed policy with this constraint relaxed is presented in table II. Note that this policy may be sensitive to the initial shape and number of modules. We find experimentally that our best hand-designed policy does only as well as the one found by GAPS (figure 4).

E. Remarks on policy correctness

Consider the stochastic policy to which GAPS converged in table III. The table only shows those observation-action pairs where $\theta(o,a) >> \theta(o,a')$ for all a' and where executing a results in motion. This policy is a local optimum in policy space – a small change in any $\theta(o,a)$ will lead to less reward. It was found by the learner on a less than ideal run and it is not the global optimum, achieving an average reward of only 13.5 (34% of possible reward). We argue that this policy will still correctly perform the task of eastward locomotion with high probability as the $\theta(o,a)$ gets larger for the actions a shown.

Note first of all that if the rules in III were deterministic, then we could make an argument akin to the one in [2] where correctness is proven for a hand-designed policy. Intuitively, if we start with a rectangular array of modules and assume that each module gets a chance to execute an action during a turn, then some rule can always be applied, and none of the rules move any modules west, so that eastward locomotion will always result. This crucially depends on our assumption of synchrony. Figure 5 shows the first several actions of the only possible cyclic sequence for nm modules following these rules if treated as deterministic. The particular assignment of module IDs in the figure is chosen for clarity and is not essential to the argument.

However, the rules are stochastic. The probability of the robot's center of mass moving eastward over the course of an episode is equal to the probability, where there are T turns in an episode, that during t out of T turns the center of mass moved eastward and $t > T-t$. This will be true if $Pr\,[\,R > 0\,] >> Pr\,[\,R <= 0\,]$. The increase in parameter $\theta(o,a)$ only happens for positive R. As $\theta(o,a) \rightarrow \infty$ so $\pi(o,a,\theta) \rightarrow 1$ for the correct action a. And when the correct actions are executed, the center of mass is always expected to move eastwards during a turn.

Naturally, in practice once the algorithm has converged, we can extract deterministic rules from the table of learned parameters by selecting the highest parameter value per state.

VI. CONCLUSION AND FUTURE WORK

We investigate how to automate the design of distributed controllers for self-reconfigurable modular robots. We have formulated the locomotion problem for such a robot as a multi-agent POMDP and applied gradient-ascent search in policy value space to solve it. Our results suggest that automating controller design by learning is a promising approach. We should, however, bear in mind the potential drawbacks of direct policy search as the learning technique of choice.

As with all hill-climbing methods, there is a guarantee of GAPS converging to a local optimum in policy value

[1] The mean convergence time is measured in episodes until convergence. The mean reward is per episode in percent of achievable reward, for ease of comparison. These results are for the 9-module case.

Fig. 5. Locomotion of nm modules following rules in table III, treated as deterministic. After the last state shown, module **n** executes action S, modules **1** then **2** execute action E, then module **4** can execute action NE, and so forth. The length of the sequence is determined by the dimensions n and m of the original rectangular shape.

space, given infinite data, but no proof of convergence to the global optimum is possible. A local optimum is the best solution we can find to a POMDP problem.

In addition, we have seen that GAPS takes on average a rather long time (measured in thousands of episodes) to learn. During the initial hundreds of episodes where the algorithm explores the policy space, the modules will attempt to execute undesirable or impossible actions which could lead to damage on a physical robot. Naturally, one may not have the luxury of thousands of trial runs on a physical robot anyway.

We expect to take the following paths as we apply the learning approach to real robots. On the one hand, we are now well-posed to develop new learning algorithms to specifically fit our problem. Those would take more notice of the structure of the agents' observed environment and interaction with other agents. In particular, modules will communicate with their neighbors directly and not only through physical coupling with the world. For DGAPS, for example, near-neighbor communication of observation and observation-action pair counts or rewards may reduce the learning time. Following our preliminary results in increasing task complexity incrementally, it would also be interesting to attempt design automation in a more hierarchical way, taking explicit advantage of task decomposition. We could give the robots a set of initial behaviors or rules as a starting point, from which it would then learn to perform more advanced tasks.

We will measure the success of any future methods against the performance of algorithms presented in this paper. We are also currently working on formal arguments for probable correctness of our learning approach.

Acknowledgments

The authors would like to thank Leonid Peshkin and Luke Zettlemoyer for very helpful discussions. Support for this work was provided through NSF awards IRI-9714332, EIA-9901589, IIS-9818299, IIS-9912193 and EIA-0202789, ONR award N00014-01-1-0675, Intel, and MIT's project Oxygen. We are grateful for this support.

References

[1] J. Baxter and P. L. Bartlett. Infinite-horizon gradient-based policy search. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *Proceedings of the International Conference on Robots and Automation*, 2001.

[3] G. Z. Grudic, V. Kumar, and L. Ungar. Using policy reinforcemnet learning on autonomous robot controllers. In *Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003.

[4] K. Ito and A. Gofuku. Hybrid autonomous control for heterogeneous multi-agent system: Combining of centralized reinforcement learning and distributed rule-based control. In *Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pages 2500–2505, Las Vegas, Nevada, October 2003.

[5] K. Kawabata, H. Asama, and M. Tanaka. A study of communication emergence among mobile robots: Simulations of intention transmission. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Proceedings of the Intl. Workshop on Distributed Autonomous Robotic Systems*, volume 5, pages 71–80. Springer, 2002.

[6] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 424–431, Leuven, Belgium, 1998.

[7] G. Laurent and E. Piat. Learning mixed behaviors by parallel q-learning. In *Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[8] J. B. Lee and R. C. Arkin. Adaptive multi-robot behavior via learning momentum. In *Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, October 2003.

[9] E. Martinson, A. Stoychev, and R. C. Arkin. Robot behavior selection using q-learning. In *Proceedings of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[10] M. J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, March 1997.

[11] C. C. Moallemi and B. Van Roy. Distributed optimization in adaptive networks. In *Proceedings of Intl. Conference on Neural Information Processing Systems*, December 2003.

[12] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji. Self-repairing mechanical systems. *Autonomous Robots*, 10:7–21, 2001.

[13] L. E. Parker. Current state of the art in distributed autonomous mobile robotics. In G. Bekey and J. Barhen, editors, *Proceedings of the Workshop on Distributed Autonomous Robotic Systems*, volume 4, pages 3–12. Springer, 2000.

[14] L. Peshkin. *Reinforcement Learning by Policy Search*. PhD thesis, Brown University, November 2001.

[15] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[16] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[17] W. Yu, I. Takuya, D. Iijima, H. Yokoi, and Y. Kakazu. Using interaction-based learning to construct an adaptive and fault-tolerant multi-link floating robot. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Proceedings of the Intl. Workshop on Distributed Autonomous Robotic Systems*, volume 5, pages 455–464. Springer, 2002.