Techniques for Learning in Multi-Robot Teams

Lynne E. Parker, Claude Touzet and Fernando Fernandez

1 Introduction

Before multi-robot teams will ever become widely used in practice, we believe that advances must be made in the development of mechanisms that enable the robot teams to autonomously generate and adapt cooperative behaviors. With the current state of the art, the implementation of cooperative behaviors on physical robot teams requires expert behavior programming and experimentation, followed by extensive tuning and revision of the cooperative control algorithms. It is unlikely that a significant real-world impact of cooperative robot teams will occur as long as the current level of effort is required to implement these systems.

Researchers have recognized that an approach with more potential for the development of cooperative control mechanisms is autonomous learning. Most of the learning research to date is in the area of multi-*agent* learning (e.g., [45]). Chapter 3 of this book describes multi-agent systems from a machine learning perspective. However, much less work has been done in the area of multi-*robot* learning, although the topic is gaining increased interest. The multi-robot learning problem is particularly challenging due to a number of issues, including: very large state spaces, uncertain credit assignment, limited training time, uncertainty in sensing and shared information, nondeterministic actions, difficulty in defining appropriate abstractions for learned information, and difficulty of merging information learned from different robot experiences. However, the potential benefits are significant, and include increased robustness, reduced complexity, and increased ease of adding new assets to the team.

Some examples of multi-robot learning research include the work by Asada et al. [2], who propose a method for learning new behaviors by coordinating previously learned behaviors using Q-learning. They have applied their approach to a simulation of robots playing a simplified version of competitive soccer, and are transferring their results to physical robots. Mataric [24] introduces a method for combining basic behaviors into higher-level behaviors through the use of unsupervised reinforcement learning, heterogeneous reward functions, and progress estimators. This mechanism was applied to a team of robots learning to perform a foraging task. Kubo and Kakazu [19] proposed another reinforcement learning mechanism that uses a progress value for determining reinforcement, and applied it to simulated ant colonies competing for food.

Brooks and Mataric [5] identify four types of learning in robotic systems:

- Learning numerical functions for calibration or parameter adjustment,
- Learning about the world,
- Learning to coordinate behaviors, and
- Learning new behaviors.

This chapter discusses some of our research in multi-robot learning in two of these areas – learning new behaviors and learning for parameter adjustment.

In previous research, the types of applications that have been studied for multi-robot learning include air fleet control [39], predator/prey [4, 17, 12], box pushing [22], foraging [24], and multi-robot soccer [41, 23]. Particularly challenging domains for multi-robot learning are those tasks that are *inherently* cooperative. Inherently cooperative tasks are those that cannot be decomposed into independent subtasks to be solved by individual robots. Instead, the utility of the action of one robot is dependent upon the current actions of the other team members. This type of task is a particular challenge in multi-robot learning, due to the difficulty of assigning credit for the individual actions of the robot team members.

Of these previous application domains that have been studied in the context of multi-robot learning, only the multi-robot soccer domain addresses inherently cooperative tasks with more than two robots while also addressing the real-world complexities of embodied robotics, such as noisy and inaccurate sensors and effectors in a dynamic environment that is poorly modeled. To add to the field of challenging application domains for multi-robot learning, we have defined and have been studying a new application domain – the Cooperative Multi-robot Observation of Multiple Moving Targets (CMOMMT) – that is not only an inherently cooperative task, but, unlike the multi-robot soccer domain, is also a domain that must deal with issues of scalability to larger and larger numbers of robots. This application domain is described in Section 2.1. As a point of comparison for the learning approaches, we also briefly describe the results of a human-generated solution to the CMOMMT application in Section 2.2.

Under the topic of learning new cooperative behaviors, we present two approaches we have developed for the inherently cooperative CMOMMT application domain. The first of these approaches, described in Section 2.3, combines lazy (or instance-based) learning with Q-Learning and a new Pessimistic Algorithm for dealing with the credit assignment problem. The second approach, described in Section 2.4, combines Q-Learning with the application of the VQQL (Vector Quantization with Q-Learning) technique and the Generalized Lloyd algorithm to address the generalization issue in reinforcement learning. In the area of learning for parameter adjustment, Section 3.1 describes the L-ALLIANCE architecture [34], which enables robots to autonomously update their control parameters so that they can adapt their behavior over time in response to changing team capabilities, team composition, and mission environment. The final section of the chapter concludes with some summary remarks.

2 Learning New Cooperative Behaviors

2.1 The CMOMMT Application

The application domain that we are studying for use as a multi-robot learning testbed for new cooperative behaviors is the problem we entitle *Cooperative Multi-robot Observation of Multiple Moving Targets* (CMOMMT). This problem is defined as follows. Given:

$\mathcal{S}$:	a two-dimensional, bounded, enclosed spatial region
$\mathcal{V}$:	a team of m robot vehicles, $v_i, i = 1, 2, ...m$, with 360^0 field of view observation sensors that are noisy and of limited range
$\mathcal{O}(t)$:	a set of n targets, $o_j(t)$, $j = 1, 2, ..., n$, such that target $o_j(t)$ is located within region $\mathcal{S}$ at time t

We say that a robot, v_i, is *observing* a target when the target is within v_i's sensing range. Define

an $m \times n$ matrix $B(t)$, as follows:

$$B(t) = [b_{ij}(t)]_{m \times n} \text{ such that } b_{ij}(t) = \begin{cases} 1 & \text{if robot } v_i \text{ is } \textit{observing} \text{ target} \\ & \quad o_j(t) \text{ in } \mathcal{S} \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$$

Then, the goal is to develop an algorithm, which we call *A-CMOMMT*, that maximizes the following metric A:

$$A = \sum_{t=1}^{T} \sum_{j=1}^{n} \frac{g(B(t), j)}{T}$$

where:

$$g(B(t), j) = \begin{cases} 1 & \text{if there exists an } i \text{ such that } b_{ij}(t) = 1 \\ 0 & \text{otherwise} \end{cases}$$

That is, the goal of the robots is to maximize the average number of targets in $\mathcal{S}$ that are being observed by at least one robot throughout the mission that is of length T time units. Additionally, we define $\textit{sensor\_coverage}(v_i)$ as the region visible to robot v_i's observation sensors, for $v_i \in \mathcal{V}$. Then we assume that, in general, the maximum region covered by the observation sensors of the robot team is much less than the total region to be observed. That is, $\bigcup_{v_i \in \mathcal{V}} \textit{sensor\_coverage}(v_i) \ll \mathcal{S}$. This implies that fixed robot sensing locations or sensing paths will not be adequate in general, and instead the robots must move dynamically as targets appear in order to maintain their target observations and to maximize the coverage.

The CMOMMT application offers a rich testbed for research in multi-robot cooperation, learning, and adaptation because it is an inherently cooperative task. In addition, many variations on the dynamic, distributed sensory coverage problem are possible, making the CMOMMT problem arbitrarily more difficult. For example, the relative numbers and speeds of the robots and the targets to be tracked can vary, the availability of inter-robot communication can vary, the robots can differ in their sensing and movement capabilities, the terrain may be either enclosed or have entrances that allow objects to enter and exit the area of interest, and so forth. Many other subproblems can also be addressed, including the physical tracking of targets (e.g. using vision, sonar, IR, or laser range), prediction of target movements, multi-sensor fusion, and so forth.

2.2 A Human-Generated Solution to CMOMMT

As a baseline for comparing the learning approaches, we developed a human-generated solution to the CMOMMT problem. This solution has been implemented on both physical and simulated robots to demonstrate its effectiveness. The human-generated solution, which we call *A-CMOMMT*, involves robots using weighted local force vectors that attract them to nearby targets and repel them from nearby robots. The weights vary over time and are based on the relative locations of the nearby robots and targets. The weights are aimed at generating an improved collective behavior across robots when utilized by all robot team members.

The magnitude of the force vector attraction of robot v_l relative to target o_k, denoted $\mid \mathbf{f_{lk}} \mid$, for parameters $0 < do_1 < do_2 < do_3$, is:

$$\mid \mathbf{f_{lk}} \mid = \begin{cases} \frac{-1}{do_1} & \text{for } d(v_l, o_k) \leq do_1 \\ \frac{1}{do_2 - do_1} & \text{for } do_1 < d(v_l, o_k) \leq do_2 \\ \frac{-do_2}{do_3 - do_2} & \text{for } do_2 < d(v_l, o_k) \leq do_3 \\ 0 & \text{otherwise} \end{cases}$$

where $d(a, b)$ returns the distance between two entities (i.e., robots and/or targets). The magnitude of the force vector repulsion of robot v_l relative to robot v_i, denoted $\mid \mathbf{g_{li}} \mid$, for parameters $0 < dr_1 < dr_2$, is:

$$\mid \mathbf{g_{li}} \mid = \begin{cases} -1 & \text{for } d(v_l, v_i) \leq dr_1 \\ \frac{1}{dr_2 - dr_1} & \text{for } dr_1 < d(v_l, v_i) \leq dr_2 \\ 0 & \text{otherwise} \end{cases}$$

Using only local force vectors for this problem neglects higher-level information that may be used to improve the team performance. Thus, the human-generated approach enhances the control approach by weighting the contributions of each target's force field on the total computed field. This higher-level knowledge can express any information or heuristics that are known to result in more effective global control when used by each robot team member locally. The human-generated approach expresses this higher-level knowledge in the form of a weight, w_{lk}, that reduces robot r_l's attraction to a nearby target o_k if that target is within the field of view of another nearby robot. Using these weights helps reduce the overlap of robot sensory areas toward the goal of minimizing the likelihood of a target escaping detection.

The higher-level weight information is combined with the local force vectors to generate the commanded direction of robot movement. This direction of movement for robot v_l is given by: $\sum_{k=1}^{n} w_{lk}\mathbf{f_{lk}} + \sum_{i=1, i\neq l}^{m} \mathbf{g_{li}}$, where $\mathbf{f_{lk}}$ is the force vector attributed to target o_k by robot v_l and $\mathbf{g_{li}}$ is the force vector attributed to robot v_i by robot v_l. To generate an (x, y) coordinate indicating the desired location of the robot corresponding to the resultant force vector, we scale the resultant force vector based upon the size of the robot. The robot's speed and steering commands are then computed to move the robot in the direction of that desired location.

To evaluate the human-generated approach, we ran a series of experiments both in simulation and on physical robots. Figure 1 shows two of the simulation runs (out of over 1,000,000), while Figure 2 shows snapshots of two of the physical robot experiments (out of over 800) in which the the robots perform the task either with no obstacles in the work area or with randomly distributed obstacles. The results of the human-generated approach to CMOMMT vary depending upon a number of factors, including the relative numbers of robots and targets, the size of the work area, the motions of the targets (i.e., whether random or evasive), and the setting of the weights. In general, the *A-CMOMMT* algorithm performed best for a ratio of targets to robots greater than 1/2. We compared the human-generated *A-CMOMMT* approach with a non-weighted local force vector approach, as well as two control cases in which robots either maintained fixed positions or moved randomly. Figure 3 gives a typical set of these comparative results. Complete details of this approach can be found in [35].

2.3 Learning Approach #1: Distributed, Pessimistic Lazy Q-Learning

The first multi-robot learning approach we present in the CMOMMT domain addresses the problem without the assumption of an *a priori* model. This approach uses a combination of reinforcement learning, instance-based (or lazy) learning, and a Pessimistic Algorithm able to compute for each team member a lower bound on the utility of executing an action in a given situation. The challenges in this multi-robot learning problem include a very large search space, the need for communication or awareness of robot team members, and the difficulty of assigning credit in an inherently cooperative problem.

In this learning approach, instance based (or lazy) learning [1] is used to enable robot team members to build a memory of situation-action pairs through random exploration of the CMOMMT problem. A reinforcement function gives the utility of a given situation. The Pessimistic Algorithm for each robot then uses the utility values to select the action that maximizes the lower bound

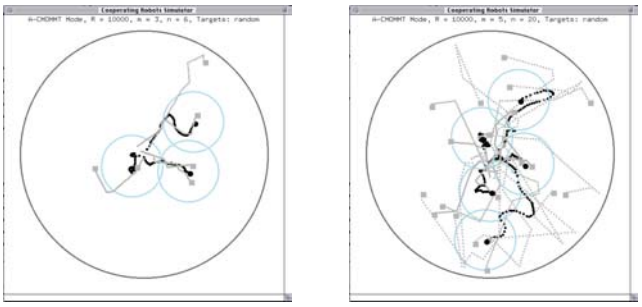


Figure 1: Simulation results of three robots and six targets (first image), and five robots and twenty targets (second image), with the robots using the human-generated solution to CMOMMT, and the targets moving randomly.

Figure 2: Robot team executing human-generated solution to CMOMMT. The first photo shows robots operating in an area with no obstacles. The second photo shows the robots amidst randomly distributed obstacles.

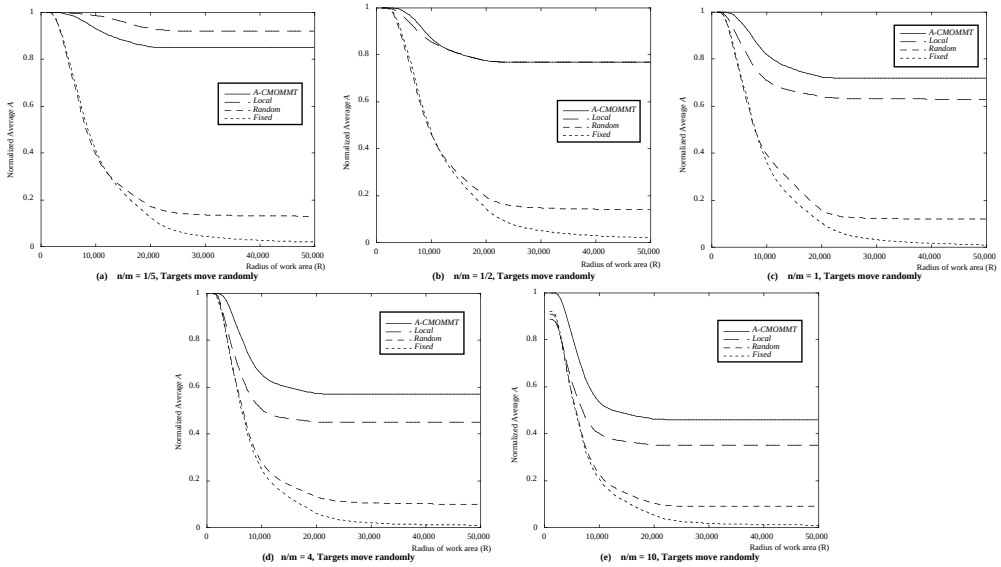


Figure 3: Simulation results of four distributed approaches to cooperative observation, for random/linear target movements, for various ratios of number of targets (n) to number of robots (m).

on utility. The resulting algorithm is able to perform considerably better than a random action policy, although it is still significantly inferior to the human-generated algorithm described in the previous section. However, even with a performance less than that of the human-generated solution, this approach makes an important contribution because it does not assume the existence of a model (as is the case in the Partially Observable Markov Decision Process (POMDP) domain), the existence of local indicators that help individual robots perform their tasks, nor the use of symbolic representations. The following subsections describe this approach and its results in more detail.

2.3.1 Q-Learning and Lazy Learning

Q-learning [44] is one of the most-used reinforcement learning algorithms for an agent to learn an action policy. It is based in the use of a state-action table containing the gain that the agent obtains by executing an action from a state. This table represents a function Q that tells the agent which action it should execute in order to obtain a maximum gain. The algorithm is summarized in Table 1.

Q-learning algorithm (S, A)

For each pair ($s \in \mathcal{S}$, $a \in \mathcal{A}$), initialize the table entry $Q(s, a)$ to 0.
Observe the current state s.
Do forever:

- Select an action a and execute it.
- Receive immediate reward r.
- Observe the new state s'.
- Update the table entry for $Q(s, a)$ as follows:

$$Q_n(s, a) \leftarrow (1 - \alpha_n) Q_{n-1}(s, a) + \alpha_n \{r + \gamma \max_{a'} Q_{n-1}(s', a')\} \quad (1)$$

- Set s to s'.

Table 1: Q-learning algorithm.

Lazy learning [1] – also called instance-based learning – promotes the principle of delaying the use of the gathered information until the necessity arises (see Figure 4). The same pool of information (i.e., memory) is used for different behavior syntheses. The lazy memory provides a good way of reducing the duration of any robotic learning application. In the context of reinforcement learning, lazy learning provides an instantaneous set of situation-action pairs (after the initial and unique sampling phase). Lazy learning samples the situation-action space according to a random action selection policy, storing the succession of events in memory and, when needed, probes the memory for the best action. The exploration phase is performed only once. By storing situation-action pairs, a lazy memory builds a model of the situation transition function.

In order to express a behavior, the memory must be probed. To do this probing, we use a modified version of the technique proposed in [38]. In [38] the objective is to provide a method for

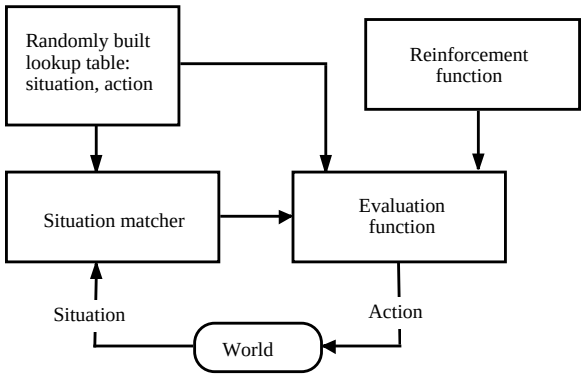


Figure 4: Lazy learning: randomly sampled situation-action pairs in the lookup table are used by the situation matcher to select the action to execute in the current situation. The reinforcement function qualifies the actions proposed, helping to select the best one.

predicting the rewards for state-action pairs without explicitly generating them. For the current real world situation, a situation matcher locates all the states in the memory that are within a given distance. If the situation matcher has failed to find any nearby situations, the action comparator selects an action at random. Otherwise, the action comparator examines the expected rewards associated with each of these situations and selects the action with the highest expected reward. This action is then executed, resulting in a new situation. There is a fixed probability (0.3) of generating a random action regardless of the outcome of the situation matcher. New situation-action pairs are added to the memory, along with a Q-value computed in the classical way [44]. Among similar situation-action pairs in the memory, an update of the stored Q-values is made. However, there is a limit to the generality of this lazy memory because the Q-values associated with the situation-action pairs only apply for a particular behavior. With the desire of reducing the learning time as much as possible, as well as preserving the generality of the lazy memory, we modified the algorithm as follows: (1) the situation matcher always proposes the set of nearest situations – no maximum distance is involved, (2) there is no random selection of actions by the action comparator, and (3) the Q-values are not stored with the situation-action pairs, but are computed dynamically as the need arises.

2.3.2 The Pessimistic Algorithm

We define a Pessimistic Algorithm for the selection of the best action to execute for a given robot in its current local situation as follows: find the lower bounds of the utility value associated with the various potential actions that may be conducted in the current situation, then choose the action with the greatest utility. A lower bound is defined as the lowest utility value associated with a set of similar situations.

The idea behind the Pessimistic Algorithm is that a local robot situation is an incomplete observation of the true state of the system. Thus, instead of trying to solve the observation problem by completing the observation (the usual POMDP approach), we are only interested in ranking the utility of the actions. If we use a unique instance of the memory to obtain the utility of the situation, then chances are that the utility attributed to this local situation is due in fact to other robot's actions. This probability decreases proportionally with the number of similar situations that are taken into account. If a large number of situations are considered, then there must be at least one for which the reward directly depends on the local situation. By taking the minimum

utility value of the set of similar situations, we are guaranteed that, if the value is null, then the situation achieved does not imply loosing target(s).

The Pessimistic Algorithm is then given as follows:

- Let M be the memory, a lookup table of situation-action pairs gathered during an exploration phase: $M = [(s(1), a(1)), ..., (s(t), a(t)), (s(t+1), a(t+1)), ...]$.
- Let sit be the current situation.
- Find $S(sit)$, the set of n situations of M similar to sit.
- Let $S_{follow}(sit)$ be the set of the situations that directly follows each situation of $S(sit)$.
- Compute the lower bound (LB) of the utility value (U) associated with each situation $s(k) \in S_{follow}(sit)$:
  - $LB(s(k)) = \min(U(s(m)))$, for $s(m) \in S(s(k))$, the set of situations similar to $s(k)$.
- Execute the action that should take the robot to the new situation s^*: $s^* = \max(LB(s))$ and $s \in S_{follow}(sit)$.

The utility U associated with a given situation can be computed in many ways. It can be the exact value of the reinforcement function for this particular situation-action pair, or it can be a more elaborate variable. For example, in our experience we store the situation-action pairs, plus the number of targets under observation in the lookup table (M). However, the value that is used as utility is +1 if one or more targets have been acquired compared to the previous situation, -1 if one or more targets have been lost, or 0 otherwise. An exact Q value requires running the Q-learning algorithm with the samples stored in the memory.

2.3.3 Results of Instance-Based Q-Learning

We studied the efficiency of the Pessimistic Algorithm by comparing the performance of a team of robots with a purely random action selection policy, a user-defined non-cooperative policy and *A-CMOMMT*. In these experiments, each robot situation is a vector of two times 16 components. The first 16 components code the position and orientation of the targets, simulating a ring of 16 sensors uniformly distributed around the robot body. Each sensor measures the distance to the nearest target, with the sensor position around the body giving the orientation. The second ring of 16 components code in the same manner the position and orientation of neighboring robots. The maximum range for a target or a robot to be seen is 1000, for an arena radius of 5000. The actions of each robot are rotation and forward movement. The measure of performance is the mean observation time of all targets.

Figure 5 shows the performance of a Pessimistic lazy Q-learning policy versus the size of the lazy memory, from 100 to 900 situation-action pairs. Each point is the average of 10 experiments. The lazy memories are obtained through an initial exploration involving from 15 to 25 targets and a single robot. During the sampling, the targets are fixed and the robot's policy is random action selection (with 5% chance of direction and orientation changes). The reinforcement function returns +1 if the total number of targets under observation increases, -1 if this number decreases, or 0 otherwise.

As we see there is an important performance gain associated with the Pessimistic lazy Q-learning over a purely random selection policy. Even more interestingly, lazy Q-learning performs much better than the human-defined non-cooperative policy (*Local*). It is important to note that neither

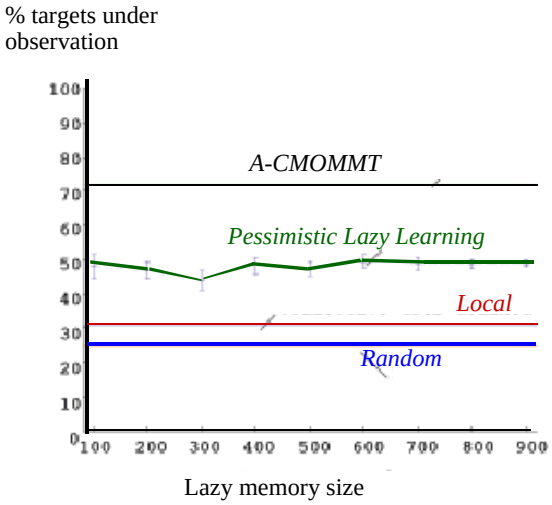


Figure 5: Performances of the Pessimistic lazy Q-learning approach compared to a random action selection policy, a user-defined non-cooperative policy and the human-generated solution *A-CMOMMT*, for 10 robots and 10 randomly moving targets. The results are the mean of 10 different experiments per point for lazy learning policy, and 100 experiments for the other 3 policies. Each experiment duration is 1000 iterations.

policy is aware of the existence of the other robots. Both policies use the same sensory information – i.e., the distance and orientation of nearby targets. It is our opinion that the variation of performance is due to the fact that the lazy Q-learned behavior is somewhat less rigid than the user-defined policy. A lazy Q-learning guided robot will follow a target further than it could be, and, in doing so, will exhibit an erratic path, moving from one side of the target to another, back and forth without losing the target. In doing so, the surface under observation per unit of time is larger than the covered surface by the more rigid center-of-gravity-oriented robot. On the other hand, because it does not take into account the neighboring robots, it is easy to understand why the lazy Q-learned behavior performance cannot reach the level of the *A-CMOMMT* performance. In our current work, we are extending this approach so that it does take into account neighboring robots.

2.4 Learning Approach #2: Q-Learning with VQQL and Generalized Lloyd Algorithm

Our second approach for multi-robot learning of new cooperative behaviors in inherently cooperative domains applies Q-Learning along with the VQQL [11] technique and the Generalized Lloyd Algorithm [21, 20] to address the generalization issue in reinforcement learning. The application of reinforcement learning [13] is useful in robot control tasks, but it usually requires a well-defined state space by which the robot learns to select an adequate action for each current state to accomplish the task. Typical solutions to the state space definition were proposed in [25, 26], where variable resolution discretization of the input data was used in order to solve control problems. In [6] similar ideas were proposed, but most of these techniques assume a deterministic environment, and are thus not easily applicable to a robotic domain, where indeterminism from sensors and actuators is always present. In this sense, several approaches can be found. In [3], hyper ellipsoids were used to cluster stored vectors and to learn a state transition map in terms of actions from which the optimal action sequence is obtained. In [43], situations are learned and maintained in the

continuous state space on the basis of the rewards from the environment, learning the transitions between the situations in order to apply partial planning over the network learned. In [37], another heuristic is presented to split the state space environment on the basis of the rewards. In general, most of these approaches have been studied in low size domains, typically with only two or three attributes for the input state representation.

Other related work shows the utility of using quantization techniques to represent the state space. An adaptive input-space quantization was proposed in [18] based on Voronoi quantizers, where neurons were added and removed based on the behavior of the robots. In [7] the Kohonen LVQ algorithm [16] and Q-Learning [44] work together to locate a pre-defined number of neurons in adequate positions of the environment to achieve the goal. In [42], associative memory [15] and Q-Learning are used to learn a map from an input representation of the space state to actions.

The VQQL technique we describe here allows the application of the Q-Learning algorithm over local information to learn behaviors in large domains by obtaining reduced representations of such domains using an unsupervised learning technique [21, 20]. This technique allows a dramatic reduction in the number of states that would be needed if a uniform representation of the environment of the robots were used, thus allowing a better use of the experience obtained from the environment. In this way, useful information can be stored in the state space representation without excessively increasing the number of states needed.

2.4.1 Reducing the Domain Size

The reinforcement learning model [13] is usually explained in terms of finite and discrete parameters. The usual execution cycle begins with an agent receiving information from the environment. This information is used to represent an individual state s from a finite set $\mathcal{S}$. Given that state, the action policy finds which action a from a finite set $\mathcal{A}$ is useful to achieve a goal. Then the agent executes the action, receiving feedback from the environment about the new state s' and a reinforcement signal, $r \in \mathcal{R}$. This discrete scheme is useful when time is not continuous and the execution of actions generates state transitions. In this case, the discount parameter γ used for learning is an integer parameter that ranges from 0 to infinity.

However, in most domains, such as the CMOMMT application, none of the previous assumptions about discrete environments are true; we therefore need a mechanism to discretize states and actions in a world with continuous time. Thus, a discretized reinforcement learning problem can be defined as follows [9]:

- The continuous state space, $\mathcal{S}$, is mapped to a finite one, $\hat{\mathcal{S}}$. Each discretized new state can be considered as a region or situation in the continuous state space.
- The continuous action space is mapped to a discretized set of high-level macro-actions or skills, $\hat{\mathcal{A}}$.
- State transitions exist only among regions by executing macro-actions. A state transition from any region $\hat{s}$ to other region $\hat{s'} \in \hat{\mathcal{S}}$ with a macro-action $\hat{a} \in \hat{\mathcal{A}}$ is given when, from any continuous state s in region $\hat{s}$, the agent executes macro-action $\hat{a}$ until it arrives in a continuous state s' into a region $\hat{s'}$ different than $\hat{s}$. That is, a state transition is given only when the agent changes regions in the state space. This transition is given in a continuous time t.
- The reinforcement scheme used is a delayed reward scheme, where positive or negative rewards are obtained only at the end of a training session. Then, if while the agent is in a region it arrives at an end state, it receives an immediate reward from the environment and updates

its policy. If the agent arrives in any other region, it receives a null reward and updates its policy taking into account the region in which it arrives and the time that it has taken.

2.4.2 Generalized Lloyd Algorithm (GLA)

The generalized Lloyd algorithm [21, 20] is a clustering technique that consists of a number of iterations, each one recomputing the set of more appropriate partitions of the input states (vectors), and their centroids. The algorithm is shown in Table 2. It takes as input a set T of M input states, and generates as output the set C of N new states (*quantization levels*).

Generalized Lloyd algorithm (T, N)

1. Begin with an initial codebook C_1.
2. Repeat:
   (a) Given a codebook (set of clusters defined by their centroids) $C_m = \{y_i; i = 1, \ldots, N\}$, redistribute each vector (state) $x \in T$ into one of the clusters in C_m by selecting the one whose centroid is closer to x (nearest neighbor rule).
   (b) Recompute the centroids for each cluster just created, R, to obtain the new codebook C_{m+1}, using equation 2:

$$cent(R)[i] = \frac{1}{\|R\|} \sum_{j=1}^{\|R\|} x_j[i] \tag{2}$$

   where $x_j \in R$, $x_j[i]$ is the value of component (attribute) i of vector x_j, and $\|R\|$ is the cardinality of R.
   (c) If an empty cell (cluster) was generated in the previous step, an alternative code vector assignment is made (instead of the centroid computation).
   (d) Compute the average distortion for C_{m+1}, D_{m+1}.

   Until the distortion has only changed by a small enough amount since last iteration.

Table 2: The generalized Lloyd algorithm.

There are three design decisions to be made when using such a technique:

Stopping criterion. Usually, the average distortion of a codebook at cycle m, D_m, is computed and compared to a threshold θ $(0 \leq \theta \leq 1)$ as in equation 3:

$$(D_m - D_{m+1})/D_m < \theta \tag{3}$$

Empty cells. One of the most used mechanisms consists of splitting other partitions, and reassigning the new partition to the empty one. All empty cells generated by the GLA are changed in each iteration by another cell. To define the new one, another non-empty cell, y, with large average distortion is split in two:

$$y_1 = \{y[1] - \epsilon, \ldots, y[K] - \epsilon\}, \text{and}$$

$$y_2 = \{y[1] + \epsilon, \ldots, y[K] + \epsilon\}$$

Initial codebook generation. We have used a version of the GLA [20] as explained in Table 3, that requires a partition split mechanism as the one described above inserted into the GLA in Table 2.

GLA with Splitting (T)
1. Begin with an initial codebook C_1 with N (number of levels of the codebook) set to 1. The only level of the codebook is the centroid of the input.
2. Repeat:
(a) Set N to $N * 2$
(b) Generate a new codebook C_{m+1} with N levels that includes the codebook C_m. The rest of the N undefined levels can be initialized to 0.
(c) Execute the GLA algorithm in Table 2 with the splitting mechanism with parameters (T, N) over the codebook obtained in the previous step.
Until N is the desired level.

Table 3: A version of the generalized Lloyd algorithm that solves the initial codebook and empty cell problems.

2.4.3 Application of *VQ* to *Q-Learning*: VQQL

The use of vector quantization and the generalized Lloyd algorithm to solve the generalization problem in reinforcement learning algorithms requires two consecutive phases:

Learn the quantizer. Design the N-levels vector quantizer from input data obtained from the environment.

Learn the Q function. Once the vector quantizer is designed (i.e., the environment is clustered into N different states), the Q function must be learned, generating the Q table composed of N rows and a column for each action.

To apply vector quantization with the goal of a domain reduction, followed by the application of a discrete reinforcement learning technique, two strategies can be used: an off-line version where all the initial data is obtained in a previous step, or an on-line version, where the learning phase is interactive, and exploitation strategies can be followed. In this work, the on-line version of the model is applied, using the following steps:

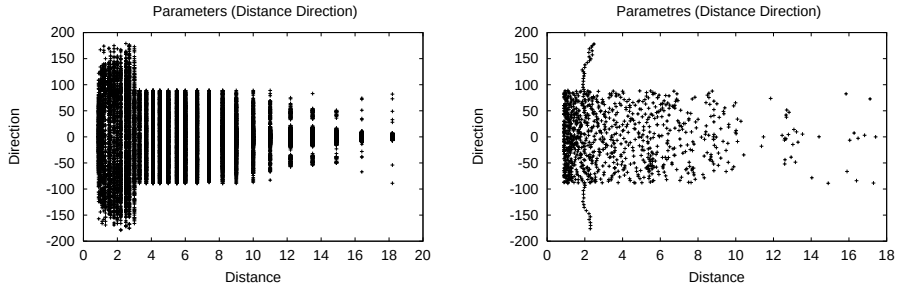
- Obtain a set T of examples of states.
- Design a vector quantizer C using T with the Generalized Lloyd Algorithm.
- Learn the Q function using the following steps:

- Choose an action following an exploration/exploitation strategy. Receive the reward, obtaining an experience tuple $< s, a, s', r >$. (This tuple is obtained by following the discretized reinforcement learning approach introduced at beginning of this section.)
- Quantify the experience tuple, obtaining $< \hat{s}, a, \hat{s'}, r >$.
- Create the Q table following the algorithm in Table 1.

Both phases – the design of the vector quantizer and the reinforcement learning technique – are applied within the VQQL model [11].

2.4.4 Previous Application of VQQL

VQQL was first used in the RoboCup domain [14] in the simulation league. The Vector Quantization technique allows us to take advantage of the statistical characteristics of the domain to reduce its size, taking into account only relevant zones from the complete domain. In [10] the ball interception skill of a goalie is learned in the robosoccer domain. Figure 6 (a) shows examples of the distance and direction parameters from the ball to the goalie – two of the four parameters used for learning the skill. Figure 6 (b) shows the clusters obtained when the GLA algorithm is used.



(a) (b)

Figure 6: Distance and direction parameters from (a) original data, and (b) a codebook obtained by the GLA.

In this case, data obtained from random behavior of the agent was used to learn the clusters. After using Q-Learning to learn the skill, more than 60% of balls are intercepted by the goalie. In Figure 6, we can see how GLA places clusters only in those zones of the environment where there are examples of the original data, with the bias of situating more clusters in those zones with the highest density of examples.

In [9], an iterative version of the algorithm obtains advantages from the behavior that the robot is learning. The main idea is that the state representation might change while the robot is learning so it iteratively repeats both phases of the VQQL model: learning the state space representation and learning the behavior.

2.4.5 Reinforcement Learning in the CMOMMT Application

In our second learning approach, the CMOMMT application has been redefined as a delayed reward reinforcement learning problem. To explain this approach, we discuss several issues in the following paragraphs.

The Input Data. In the CMOMMT domain, relevant input data consists of the locations of the targets and the other robots. However, at each moment, we likely have a partially observable

environment, since not all targets and robots will be generally known to each robot. As an approximation, this approach maintains information on the nearest target and the nearest robot, using a mask when information is not known. In this case, the size of the input data depends on the number of targets and robots used as local information, and will differ in different experiments.

The Actions. Actions are discretized into eight skills, following the cardinal points: go North, go North-East, go East, etc. Additional actions can be introduced if desired. Then, if the agent is in a discretized state $\hat{s}$, and performs the action "go North", it will be moving until it arrives in a discretized state $\hat{s'} \neq \hat{s}$.

The Reinforcement Function. The reinforcement function changes in the experiments, depending on the input data that is received. In most cases, positive rewards are given when targets are observed, so a higher reward is gained with higher numbers of targets in view. As will be described below, this positive reward is counteracted in some experiments when other robots are in view, which is the criterion used to define whether or not the robots are collaborating. Therefore, negative reinforcements may be received if other robots are in the same viewing range. Furthermore, a delayed reinforcement approach has been followed, so reinforcements are only received at the end of each trial.

2.4.6 Results of Q-Learning with VQQL

To evaluate this second approach to multi-robot learning, we conducted a number of experiments. These experiments consist of applying the VQQL model to the CMOMMT application in order to achieve a higher performance, following the performance measure defined in Section 2.1 as the average number of targets under observation in runs of 1000 cycles. The maximum range for a target or a robot to be seen is 1000, for an arena radius of 5000.

In this case, the on-line version of the model has been used. Different experiments have been executed in order to compare the performance of the model based on two main factors – (1) the number of states used for the state space representation, and (2) whether robots use collaboration in order to achieve a higher performance.

Experiment 1

In the first set of experiments, each state is a two component tuple storing the x and y component of the distance vector from the robot to the furthest target in view of the robot. Figure 7 shows the x and y components of the distance vector of the input data obtained in the first step of the VQQL model. In this sense, we can see how this input data already introduces statistical information. For instance, the x component and y component are such that the distance vector is smaller than the range of view of the robots, except for the point $(1000, 1000)$, which is the mask value used when the robot cannot see any target.

When applying the GLA algorithm to obtain a smaller representation of all this data, we can obtain several codebooks of different sizes. Figure 8 shows the distortion evolution of the training data and the test data, and how in both cases, the average distortion obtained is less than 1000 for codebooks of 256 states or more.

Figure 9 shows the state representation obtained with both a codebook of 16 and a codebook of 64 centroids, i.e., a domain with 16 or 64 different states. We can see how this representation is adapted to the input data shown in Figure 7, including a state for the point $(1000, 1000)$.

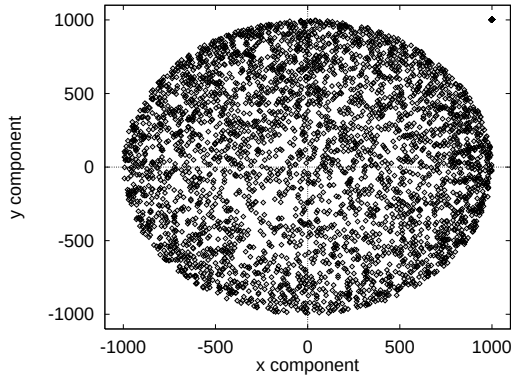


Figure 7: Distance vectors from the robot to the furthest target within viewing range.

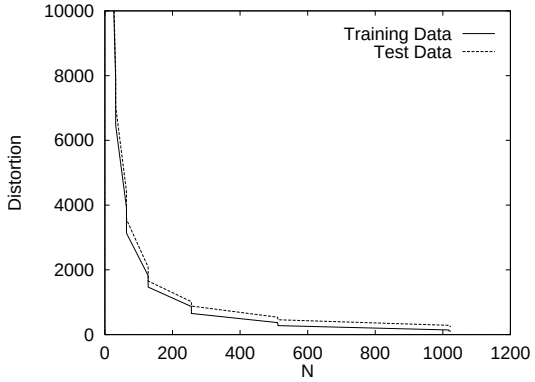


Figure 8: Distortion evolution versus size of the codebook.

Given this representation, the next step is to learn the multi-target observation task. As in the rest of the experiments, eight actions have been defined for the robot, each defined to follow a cardinal point: "go north", "go north-east", "go east", etc. The delayed reinforcement function is the number of targets that the agent sees. The number of targets is 10, while the number of robots is 1 in the learning phase, and 10 in the test phase. In this case, no collaboration strategy has been defined in the experiment among the robots, and positive delayed reward is only given at the end of each trial equal to the number of targets under observation. Furthermore, if the robot loses all the targets, it receives a negative reinforcement, and remains motionless until some target enters its viewing range. The duration of each trial in the learning phase is 100.

Figure 10 shows the results of the learning for different size codebooks and different learning phase lengths. From this figure, we see that the VQQL model achieves a performance of approximately 59% for most of the studied cases. It is an improvement over the first learning approach explained in Section 2.3, where a 50% rate of performance was achieved with the Pessimistic Lazy Q-Learning algorithm.

In this case, with only 16 different states the robots achieve a 59% rate of performance; higher state representations do not provide higher performance improvements. Another important issue is the learning rate. Smaller state space representations (16 or 64 states) learn very fast, requiring only 100-150 trials, while higher numbers of states (256) need at least 200 trials to achieve similar results. The 1024 level model cannot achieve this success within 1000 trials. Finally, we note that

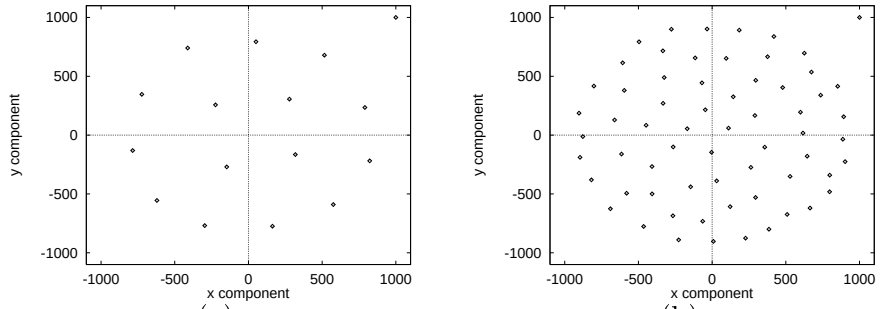


Figure 9: Codebooks representing the state space representation obtained for two different codebook sizes (16 and 64).

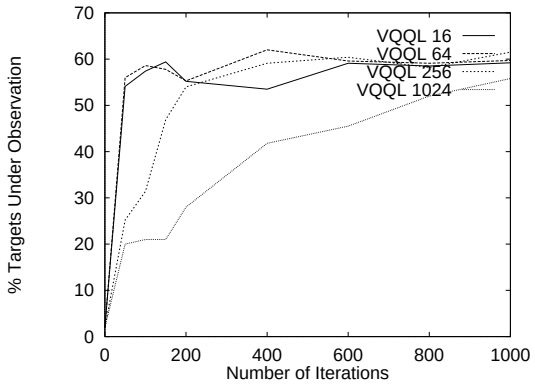


Figure 10: Performance of the VQQL model in the CMOMMT application.

convergence in learning is harder to achieve because of the exploration and exploitation strategy followed in the Q-Learning phase, where the learning rate α is constant (0.05), for a discount parameter of $\gamma = 0.6$. Convergence might be improved with another strategy.

Why cannot the robots achieve a higher performance? The answer to this question can be easily found in Figure 11. In this figure, note how ten robots are grouped into only three clusters. In this case, the same performance could be achieved if only three robots were used instead of ten. Thus, a collaboration strategy must be introduced in order to avoid this sort of behavior.

Experiment 2

The goal of this experiment is to achieve a better performance by introducing collaborative behaviors among the robots. In this sense, given that the only signal that the robots receive in order to learn their behavior is the reinforcement signal, the collaboration must be implicitly introduced. In this case, the state space representation is increased in order to incorporate more information about targets and other robots. Thus, input data is composed of information about the nearest target within view, the furthest target within view, as well as the nearest robot. This increases the input vector from two to six components.

Adding this new information requires a change in the training phase as well, in that now, ten robots simultaneously learn the same policy using the same Q table during learning and testing. The implication of this approach is that the behavior is learned up to ten times faster, due to

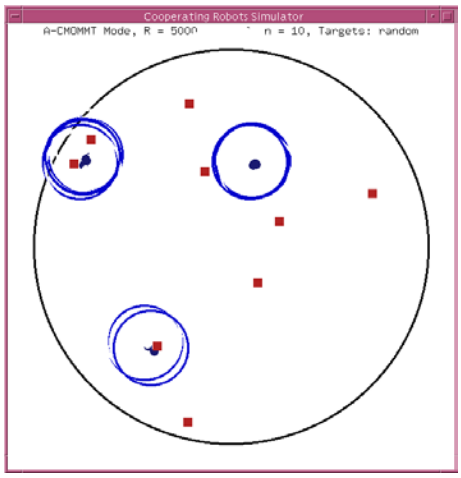


Figure 11: Robots following targets in the CMOMMT application. The behavior is not correct because several robots follow the same target instead of following different targets.

ten robots creating the table simultaneously (although this point has not been studied in depth). Furthermore, the reinforcement signal now incorporates a negative reward that is given at the end of each trial in order to achieve a collaborative behavior. This negative reward is based on whether or not the robot has another robot in its range of view. Thus, the reinforcement function for each robot i at the last moment of the trial, $r_i(T)$ is calculated in this way (following the notation introduced in Section 2.1):

$$r_i(T) = (\sum_{j=1}^{n} b_{ij}(T)) - k(T) \tag{4}$$

where

- $b_{ij}(t) = \begin{cases} 1 & \text{if robot } v_i \text{ is } \textit{observing} \text{ target } o_j(t) \text{ in } \mathcal{S} \text{ at time } t \\ 0 & \text{otherwise} \end{cases}$
- n is the number of robots (10 in this experiment), and
- $k(T)$ is a function whose value is 1 if the robot can see other robots, or 0 otherwise.

The basic idea is that the optimal behavior will be achieved when each robot follows a different target, to the greatest extent possible. This negative reward does not insure this characteristic, but it is easy to understand that it might help.

The results shown in Figure 12 illustrate these prior expectations. In this case, a higher number of states are needed to achieve good behavior because of the increase in the number of input attributes. For state space representations of only 64, around a 40% rate of performance is achieved. For 256 states, the performance is increased up to 50%, and for 1024 states, the 60% level of performance achieved in experiment 1 is also obtained. The real improvement appears with 2048 states, where the percentage of targets under observation is 65% – five percentage points higher than in the previous experiment, and near the best human-generated solution reported in [36].

Figure 13 shows the robots following the targets using the learned behavior over a state representation of 2048 states. In this image, only one target is not followed by any robot, showing how the collaborative behavior has been achieved. Figure 14 shows the results of both experiments, and their comparison with previous works.

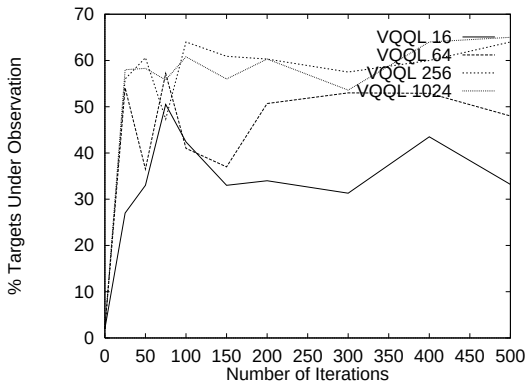


Figure 12: Performance of the VQQL model in the CMOMMT application.

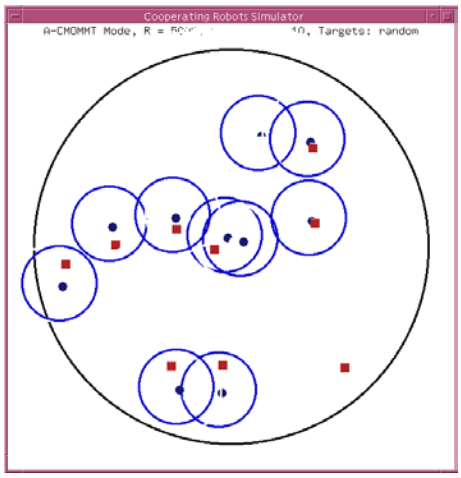


Figure 13: Robots following targets in the CMOMMT application. The behavior is better than in previous results because most of the targets are followed by only one robot.

The primary findings from this research are that the VQQL model is useful for obtaining state space representations that allow the application of the learning model based on discrete representations. In this sense, with only two attributes, the model is able to achieve results of 59% of targets in the robots' viewing area with only 16 states – an improvement over previous research that achieved a 50% rate of performance. Furthermore, the model is able to learn in a higher dimensional environment, where six attributes were introduced to obtain information about more targets and other robots. In this case, collaborative behaviors emerge only by penalizing robots for maintaining other robots within view, achieving a 64% rate of performance, nearer to the 71% achieved by the best human-generated solution. In conclusion, collaborative behaviors have been achieved by only using local information of the environment, obtaining better results than with the non-collaborative version.

In future research for learning cooperative behaviors, primary efforts must be aimed at improving performance by obtaining better representations of the state space. In this sense, guiding the unsupervised technique to use information about the behavior being learned may be useful. Some previous works introduce as information the Q value of the states [42] or the reinforcements received from the environment [43]. On the other hand, the advantages of this approach over the

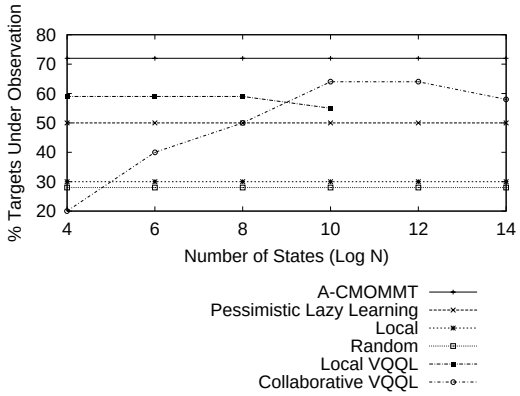


Figure 14: Performance of different approaches to the CMOMMT application

unsupervised techniques are not obvious. The main goal of this technique is to reduce the learning time by not including useless states that lead to inefficiency in learning. However, in some cases, the time and resources spent in learning a more adjusted state space representation might be longer than the penalty for using a less adjusted state representation.

3 Learning for Parameter Adjustment

An important goal in the development of autonomous robot systems is enabling robots to perform their tasks over a long period of time without human supervision. So-called "lifelong" robot systems must be capable of dealing with dynamic changes that will inevitably occur over time, such as changes in the environment or incremental variations in their own performance capabilities. Environmental changes may occur due to shifts in the weather, time-of-day daylight and temperature variation, the effect of the robot system itself in performing its application, or other causes external to the robot system. Incremental variations in robot performance capabilities may occur: (1) as a natural consequence of wear-and-tear on the robots, which may cause incremental or sudden degradations in robot performance, or (2) in more advanced robots, as a consequence of robot learning, which improves an individual robot's performance of a particular task.

The ability to adapt to these types of dynamic changes is especially important in multi-robot applications, since the effects of individual robot actions propagate across the entire team. In most real-world applications, a multi-robot team with static capabilities will not be able to continually achieve its goals over time as the system of robots and the environment drift further and further from the original state. Instead, a successful lifelong multi-robot team will adapt to changes in robot team member capabilities, robot team composition, mission requirements, and the environmental state.

Heterogeneity in multi-robot teams presents a particular challenge to efficient autonomous control when overlap in team member capabilities occurs. Overlap in team member capabilities means that more than one robot may be able to perform a given task, but with different levels of efficiency. In these cases, the robots must continually determine which individual on the team is currently the best suited for a given task in the application. These types of decisions are usually not easy to make, especially when the multi-robot team control is distributed across all team members.

3.1 Learning Approach #3: L-ALLIANCE

In previous work [30], we have developed the ALLIANCE software architecture, and have shown it to be capable of providing robot team members with fault tolerant, dynamic action selection for situations in which robots fail, the mission changes, the robot team composition changes, or the environment changes. ALLIANCE alone, however, does not address the issues of incremental degradations or improvements in individual robot performance. Thus, to address the issue of lifelong adaptation in heterogeneous multi-robot teams, we present the L-ALLIANCE (for *L*earning-*ALLIANCE*) extension to ALLIANCE that enables robot team members to efficiently learn about the capabilities of robot team members through experience with those team members, and to select their actions based upon incremental changes in individual robot team performance. The key issue addressed in this approach is:

> *Given a team of robots and a mission that they are to perform that consists of several independent tasks. how do the robots select their actions to ensure that the mission is completed as efficiently as possible, even when robot team members have overlapping, but heterogeneous, capabilities that may continually vary over time?*

Formally, let $R = \{r_1, r_2, ..., r_n\}$ represent the set of n robots on a cooperative team, and the set $T = \{task_1, task_2, ..., task_m\}$ represent the m independent tasks required in the current mission. Each robot r_i in R has a number of high-level task-achieving functions that it can perform, represented by the set $A_i = \{a_{i1}, a_{i2}, ...\}$. Since different robots may have different ways of performing the same task, we define the set of n functions H, where $H : A_i \rightarrow T$, $H = \{h_1(a_{1k}), h_2(a_{2k}), ..., h_n(a_{nk})\}$, and $h_i(a_{ik})$ returns the task $task_j$ that robot r_i is working on when it performs the high-level function a_{ik}.

We denote the metric evaluation function as $q(a_{ij})$, which returns the "quality" of the action a_{ij} as measured by a given metric. Typically, we consider metrics such as the average time or average energy required to complete a task, although many other metrics could be used. Of course, robots unfamiliar with their own abilities or the abilities of their teammates do not have access to this $q(a_{ij})$ function. Thus, an additional aspect to the robot's learning problem is actually obtaining the performance quality information required to make an "intelligent" action selection choice. Finally, define the tasks a robot r_i elects to perform during a mission as the set $U_i = \{a_{ij} | \text{robot } r_i \text{ performs task } h_i(a_{ij}) \text{ during the current mission}\}$.

In the most general form of this problem, distinct robots may have different collections of capabilities; thus, we do not assume that $\forall i.\forall j.(A_i = A_j)$. Further, if different robots can perform the same task, they may perform that task with different qualities; thus, we do not assume that if $h_i(a_{ix}) = h_j(a_{jy})$, then $q(a_{ix}) = q(a_{jy})$.

The formal Heterogeneous Robot Action Selection Problem can then be stated as follows:

Heterogeneous Robot Action Selection Problem (HRASP):

> Given R, T, A_i, and H, determine the set of actions U_i for all r_i such that $\forall(r_i \in R).U_i \subseteq A_i$ and $\forall(task_j \in T).\exists i.\exists k.((task_j = h_i(a_{ik}))$ and $(a_{ik} \in U_i))$ and the performance metric is optimized, according to the desired performance metric; for example, for the time metric, we seek to minimize: $max_i(\sum_{a_{ik} \in U_i} q_{time}(a_{ik}))$.

We note here that for reasons of robustness, fault tolerance, and flexibility, distributed decision-making very often gives better results than a centralized decisionmaker. Thus, while this efficiency

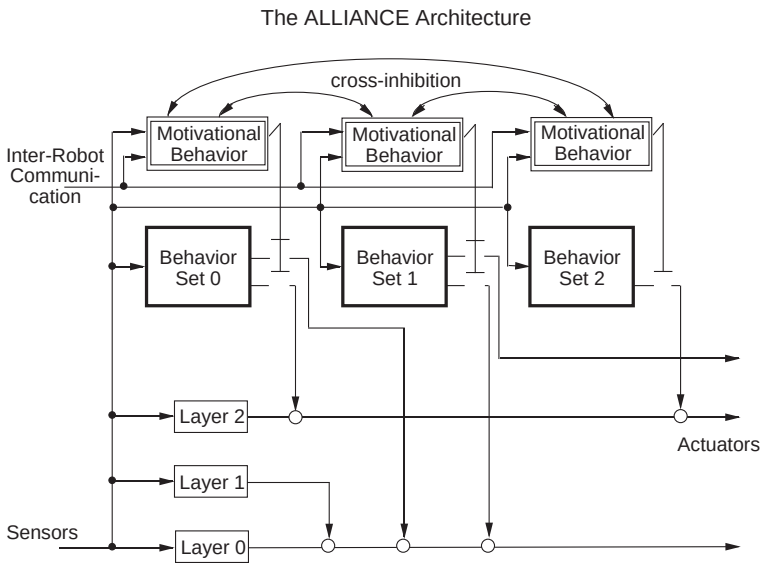


Figure 15: The ALLIANCE architecture defines the design of an individual robot, such that when the robot is brought together with other robots also designed using ALLIANCE, they exhibit dynamic action selection.

problem is formulated as a centralized decision problem, in robust real-world applications, the problem should often be solved by the distributed multi-robot team using incomplete local information. Nevertheless, even if complete global information is assumed, this efficiency problem can be easily shown to be NP-hard by restriction to the well-known NP-complete problem PARTITION (see [34] for the proof). Since this efficiency problem is NP-hard, we cannot expect the robot teams to be able to derive an optimal action selection policy in a reasonable length of time. We look instead to heuristic approximations to the problem that work well in practice.

3.1.1 Brief Overview of ALLIANCE

The foundation of our approach to achieve lifelong heterogeneous multi-robot adaptation is the ALLIANCE architecture (see Figure 15), which we developed in previous work [30]. To understand the details of the L-ALLIANCE mechanisms, we repeat a brief overview of ALLIANCE here. A major design goal in the development of ALLIANCE was to address the real-world issues of fault tolerance and adaptivity when using teams of fallible robots with noisy sensors and effectors. The aim is to create robot teams that are able to cope with failures and uncertainty in action selection and action execution, and with changes in a dynamic environment. Because of these design goals, we developed ALLIANCE to be a fully decentralized, behavior-based software architecture which gives all robots the capability to determine their own actions based upon their current situation. No centralized control is utilized, so that we can investigate the power of a fully distributed robotic system to accomplish group goals. The purpose of this approach is to maintain a purely distributed cooperative control scheme which affords an increased degree of robustness; since individual agents are always fully autonomous, they have the ability to perform useful actions even amidst the failure of other robots.

Unlike typical behavior-based approaches, ALLIANCE delineates several *behavior sets* that are either active as a group or are hibernating. Each behavior set of a robot corresponds to those levels of competence required to perform some high-level task-achieving function. Because of the alternative goals that may be pursued by the robots, the robots must have some means of selecting the

appropriate behavior set to activate. This action selection is controlled through the use of *motivational behaviors*, each of which controls the activation of one behavior set. Due to conflicting goals, only one behavior set is active at any point in time (implemented via cross-inhibition of behavior sets). However, other lower-level competences such as collision avoidance may be continually active regardless of the high-level goal the robot is currently pursuing.

The motivational behavior mechanism is based upon the use of two mathematically-modeled motivations within each robot – impatience and acquiescence – to achieve adaptive action selection. Using the current rates of impatience and acquiescence, as well as sensory feedback and knowledge of other team member activities, a motivational behavior computes a level of activation for its corresponding behavior set. Once the level of activation has crossed the threshold, the corresponding behavior set is activated, and the robot has selected an action. The motivations of impatience and acquiescence allow robots to take over tasks from other team members (i.e., become *impatient*) if those team members do not demonstrate their ability — through their effect on the world — to accomplish those tasks. Similarly, they allow a robot to give up its own current task (i.e., *acquiesce*) if its sensory feedback indicates that adequate progress is not being made to accomplish that task. The rate at which robots become impatient or acquiesce is dependent upon control parameter settings. It is these control parameters that are automatically updated by the L-ALLIANCE mechanism to achieve lifelong adaptations to continual changes in robot team member performance.

In ALLIANCE, it is important for the sake of efficiency that robots be aware of the actions of their teammates. Although it would be more elegant if robots were able to visually observe and understand the actions of their teammates, it is still very difficult to program robots with this capability. Thus, as a substitute for passive action recognition, the robots under ALLIANCE periodically broadcast their current actions to their teammates. For a complete discussion of ALLIANCE, as well as examples of its implementation in several multi-robot applications (e.g. "mock" hazardous waste cleanup, janitorial service, bounding overwatch, etc.), see [28, 30].

3.1.2 Overview of L-ALLIANCE

The L-ALLIANCE extension to ALLIANCE addresses issues of efficiency and lifelong adaptation by incorporating a dynamic parameter update mechanism into the ALLIANCE architecture. This parameter update mechanism preserves the fault tolerant features of ALLIANCE while enabling the robot team to adapt its performance to changes in the environment and robot team capabilities, thus improving or maintaining the efficiency of the robot team performance over time.

Providing a robot team with the ability to automatically update its own motivational behavior parameters requires solutions to two problems:

1. How to give robots the ability to obtain knowledge about the quality of team member performances, and

2. How to use team member performance knowledge to select a task to pursue.

The following subsections describe how these issues are addressed in L-ALLIANCE.

L-ALLIANCE Performance Monitors

The learning mechanism used in L-ALLIANCE requires robots to have the ability to monitor, evaluate, and catalog the performance of team members in executing certain tasks. Without this

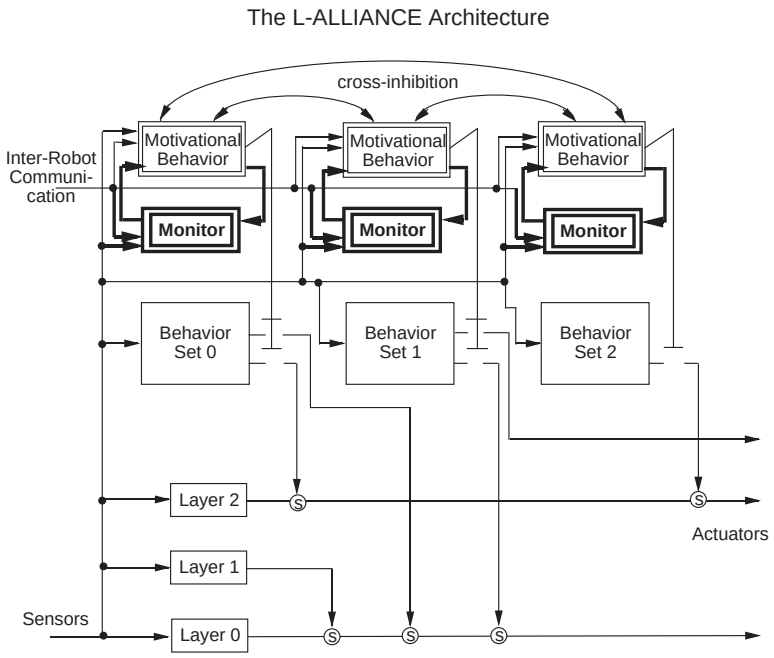


Figure 16: The L-ALLIANCE architecture extends ALLIANCE by adding performance monitors that measure the performance of robot team members for certain tasks.

ability, a robot must rely on human-supplied performance measurements of robot team members that will likely not be responsive to dynamic changes that occur over time. Once these performance measurements are obtained, the robot team members have a basis for determining the preferential activation of one behavior set over any other either for the sake of efficiency and long-term adaptation, or to determine when a robot failure has occurred.

Figure 16 illustrates the L-ALLIANCE architecture. This architecture extends the ALLIANCE architecture by incorporating the use of performance monitors for each motivational behavior within each robot. Each monitor is responsible for observing, evaluating, and cataloging the performance of any robot team member (including itself) whenever it performs the task corresponding to that monitor's respective behavior set. Formally, robot r_i, programmed with the b behavior sets $A = \{a_{i1}, a_{i2}, ..., a_{ib}\}$, also has b monitors $MON_i = \{mon_{i1}, mon_{i2}, ..., mon_{ib}\}$, such that monitor mon_{ij} observes the performance of any robot performing task $h_i(a_{ij})$, keeping track of the appropriate performance quality measure (such as time of task completion) of that robot. Monitor mon_{ij} then uses the mechanism described in section 3.1.3 to update the control parameters of behavior set a_{ij} based upon this learned knowledge. It is important to note that a robot r_i does not keep track of the performance qualities for capabilities of other robots that r_i does not share. This allows the L-ALLIANCE architecture to scale more favorably as the mission size increases.

Two L-ALLIANCE Control Phases

The degree to which robot team members can actively pursue knowledge concerning team member abilities depends on the type of mission in which they are engaged. If they are on a *training* mission, whose sole purpose is to allow robots to become familiar with themselves and with their teammates, then the robots have more freedom to explore their capabilities without concern for possibly not completing the mission. On the other hand, if the robots are on a *live* mission that

may continue for a long period of time, then the team has to ensure that the mission gets completed as efficiently as possible, while continuing to adapt their performance over time as the capabilities of their teammates change. Thus, one of two high-level control phases are utilized for robot team members under L-ALLIANCE. During training missions, the robots enter the *active learning* phase, whereas during live missions, they enter the *adaptive learning* phase.

In the active learning phase, the robots' motivational behaviors interact to cause each robot to select its next action randomly from those actions that are: (1) currently undone, as determined from the sensory feedback, and (2) currently not being executed by any other robot, as determined from the broadcast communication messages. While they perform their tasks, the robots are maximally patient and minimally acquiescent, meaning that a robot neither tries to preempt another robot's ongoing task, nor does it acquiesce its own current action to another robot. During this active learning phase, each monitor mon_{ij} in each robot r_i monitors and catalogs the performance of all robots r_k that are performing task $h_i(a_{ij})$, and then uses this information to update the appropriate control parameters.

During the adaptive learning phase, robots are working on a "live" mission. Thus, the team cannot afford to allow members to attempt to accomplish tasks for long periods of time with little or no demonstrable progress. The team members have to make a concerted effort to accomplish the mission with whatever knowledge they may have about team member abilities, and must not tolerate long episodes of robot actions that do not have the desired effect on the world. Thus, in the *adaptive learning* phase, the robots acquiesce (give up tasks) and become impatient (take over tasks) according to their learned knowledge and the control strategies described below, rather than being maximally patient and minimally acquiescent as they are in the active learning phase. However, the monitors of each robot continue to observe the robot performances during this phase, and update the control parameters appropriately, as described in the following two sections.

Action Selection Strategy

In [29], we reported on a series of experiments we conducted in simulation to determine the most efficient means for the robot team members to select their actions, based upon the relative quality measurements observed by the robots during mission performance. The goal is to implement the best action selection approach into the parameters of the L-ALLIANCE framework. By defining the parameter settings as a function of the quality measures of robot performances, the parameters can then be set and updated automatically over time during the robot team's mission. The results from our earlier experiments showed that the following action selection strategy was found to have the best performance:

L-ALLIANCE Action Selection Methodology:

1. At each iteration, robot r_i divides the remaining tasks into two categories:
   (a) Tasks meeting the intersection of the following conditions: (1) Tasks that r_i expects to be able to perform better than any other team member, and (2) tasks that no other robot is currently performing.
   (b) All other tasks r_i can perform.
2. Robot r_i repeats the following until sensory feedback indicates that no more tasks are left:
   (a) Select tasks from the first category according to the longest task first approach, unless no more tasks remain in the first category.

(b) Select tasks from the second category according to the shortest task first approach.

If a robot has no learned knowledge about team member capabilities, all of its tasks fall into the second category.

We note that the task division that occurs in the first step is a very simple process of using sensory feedback and the current system state to determine which tasks are applicable at the current point in the mission. For each applicable task, the robot looks up the stored performance measurements to divide the tasks into the categories noted. This is an $O(nm)$ operation. More discussion on the significance and derivation of this action selection strategy can be found in [29].

3.1.3 Formal Model of L-ALLIANCE

The derived action selection policy has been formally modeled into the control parameters of the L-ALLIANCE architecture. Due to space limitations, we simply state the formal model here. Refer to [34] for a detailed explanation of this model.

Given θ, which is the threshold of activity of a behavior set, and *strategy*, which is the current impatience/acquiescence update strategy, seven sources of input affect the motivation to perform a particular behavior set. These inputs are defined as:

Sensory feedback:

$$sensory\_feedback_{ij}(t) = \begin{cases} 1 & \text{if sensory feedback in robot } r_i \text{ at time } t \text{ indicates that} \\ & \text{behavior } a_{ij} \text{ is applicable} \\ 0 & \text{otherwise} \end{cases}$$

Inter-robot communication:

ρ_i = Rate of messages per unit time that robot r_i sends concerning its current activity

$$comm\_received(i,k,j,t_1,t_2) = \begin{cases} 1 & \text{if robot } r_i \text{ has received message from robot } r_k \text{ concerning} \\ & \text{task } h_i(a_{ij}) \text{ in the time span } (t_1,t_2), \text{ where } t_1 < t_2 \\ 0 & \text{otherwise} \end{cases}$$

τ_i = Maximum time robot r_i allows to pass without hearing from a particular robot before assuming that that robot has ceased to function

$$robots\_present(i,t) = \{k | \exists j.(comm\_received(i,k,j,t-\tau_i,t) = 1)\}$$

Suppression from active behavior sets:

$$activity\_suppression_{ij}(t) = \begin{cases} 0 & \text{if another behavior set } a_{ik} \text{ is active, } k \neq j, \text{ on robot } r_i \text{ at time } t \\ 1 & \text{otherwise} \end{cases}$$

Learned robot influence:

$$learning\_impatience_{ij}(t) = \begin{cases} 0 & \text{if} \\ & \left(\sum_{x \in robots\_present(i,t)} comm\_received(i,x,j,0,t) \right) \\ & \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

μ = Number of trials over which task performance averages and standard deviations are maintained

$task\_time_i(k,j,t)$ = (average time over last μ trials of r_k's performance of (task $h_i(a_{ij})$)
+ one standard deviation of these μ attempts, as measured by r_i

$$task\_category_{ij}(t) = \begin{cases} 1 & \text{if } (task\_time(i,j,t) = \\ & \quad \min_{k \in robots\_present(i,t)} task\_time(k,j,t)) \\ & \quad \text{and} \\ & \quad ((\sum_{x \in robots\_present(i,t)} comm\_received(i,x,j,t-\tau_i,t)) = 0) \\ 2 & \text{otherwise} \end{cases}$$

$boredom\_threshold_i$ = level of boredom at which robot r_i ignores the presence of other robots able to perform some task not currently being executed

$boredom\_rate_i$ = Rate of boredom of robot r_i

$$boredom_i(t) = \begin{cases} 0 & \text{for } t = 0 \\ (\prod_j activity\_suppression_{ij}(t)) \times & \\ (boredom_i(t-1) + boredom\_rate_i) & \text{otherwise} \end{cases}$$

$$learned\_robot\_influence_{ij}(t) = \begin{cases} 0 & \text{if } (boredom_i(t) < boredom\_threshold_i) \\ & \quad \text{and } (task\_category_{ij}(t) = 2) \\ 1 & \text{otherwise} \end{cases}$$

Robot impatience:

$\phi_{ij}(k,t)$ = Time during which robot r_i is willing to allow robot r_k's communication message to affect the motivation of behavior set a_{ij}.

$$= \begin{cases} task\_time_i(k,j,t) & \text{if } (strategy = \text{III}) \\ task\_time_i(i,j,t) & \text{if } (strategy = \text{II}) \end{cases}$$

$\delta\_slow_{ij}(k,t)$ = Rate of impatience of robot r_i concerning behavior set a_{ij} after discovering robot r_k performing the task corresponding to this behavior set

$$= \frac{\theta}{\phi_{ij}(k,t)}$$

min_delay = minimum allowed delay
max_delay = maximum allowed delay
$high = \max_{k,j} task\_time_i(k,j,t)$
$low = \min_{k,j} task\_time_i(k,j,t)$
$scale\_factor = \frac{max\_delay - min\_delay}{high - low}$

$\delta\_fast_{ij}(t)$ = Rate of impatience of robot r_i concerning behavior set a_{ij} in the absence of other robots performing a similar behavior set

$$= \begin{cases} \frac{\theta}{min\_delay + (task\_time_i(i,j,t) - low) \times scale\_factor} & \text{if } task\_category_{ij}(t) = 2 \\ \frac{\theta}{max\_delay - (task\_time_i(i,j,t) - low) \times scale\_factor} & \text{otherwise} \end{cases}$$

$$impatience_{ij}(t) = \begin{cases} \min_k(\delta\_slow_{ij}(k,t)) & \text{if } (comm\_received(i,k,j,t-\tau_i,t) = 1) \text{ and} \\ & \quad (comm\_received(i,k,j,0,t-\phi_{ij}(k,t)) = 0) \\ \delta\_fast_{ij}(t) & \text{otherwise} \end{cases}$$

$$impatience\_reset_{ij}(t) = \begin{cases} 0 & \text{if } \exists k.((comm\_received(i,k,j,t-\delta t,t)=1) \text{ and} \\ & \quad (comm\_received(i,k,j,0,t-\delta t)=0)), \\ & \quad \text{where } \delta t = \text{time since last communication check} \\ 1 & \text{otherwise} \end{cases}$$

Robot acquiescence:

$\psi_{ij}(t)$ = Time robot r_i wants to maintain behavior set a_{ij}'s activity before yielding to another robot.

$$= \begin{cases} task\_time_i(i,j,t) & \text{if } (strategy = \text{III}) \\ \min\limits_{k \in robots\_present(i,t)} task\_time_i(k,j,t) & \text{if } (strategy = \text{II}) \end{cases}$$

$\lambda_{ij}(t)$ = Time robot r_i wants to maintain behavior set a_{ij}'s activity before giving up to try another behavior set

$$acquiescence_{ij}(t) = \begin{cases} 0 & \text{if [(behavior set } a_{ij} \text{ of robot } r_i \text{ has been active} \\ & \quad \text{for more than } \psi_{ij}(t) \text{ time units at time } t) \text{ and} \\ & \quad (\exists x.comm\_received(i,x,j,t-\tau_i,t)=1)] \\ & \quad \text{or} \\ & \quad (\text{behavior set } a_{ij} \text{ of robot } r_i \text{ has been active} \\ & \quad \text{for more than } \lambda_{ij}(t) \text{ time units at time } t) \\ 1 & \text{otherwise} \end{cases}$$

Motivation calculation:

The motivation of robot r_i to perform behavior set a_{ij} at time t is calculated as follows:

DURING ACTIVE LEARNING PHASE:

$$\begin{aligned} random\_increment &\leftarrow \theta \times (\text{a random number between 0 and 1}) \\ m_{ij}(0) &= 0 \\ m_{ij}(t) &= [m_{ij}(t-1) + random\_increment] \times sensory\_feedback_{ij}(t) \\ &\quad \times activity\_suppression_{ij}(t) \times learning\_impatience_{ij}(t) \end{aligned}$$

The motivation to perform any given task thus increments at some random rate until it crosses the threshold, unless the task becomes complete (*sensory_feedback*), some other behavior set activates first (*activity_suppression*), or some other robot has taken on that task (*learning_impatience*).

When the robots are working on a "live" mission, their motivations to perform their tasks increment according to the robots' learned information. The motivations are thus calculated as follows:

DURING ADAPTIVE PHASE:

$$\begin{aligned} m_{ij}(0) &= 0 \\ m_{ij}(t) &= [m_{ij}(t-1) + impatience_{ij}(t)] \times sensory\_feedback_{ij}(t) \\ &\quad \times activity\_suppression_{ij}(t) \times impatience\_reset_{ij}(t) \\ &\quad \times acquiescence_{ij}(t) \times learned\_robot\_influence_{ij}(t) \end{aligned}$$

3.1.4 Results of L-ALLIANCE

The ALLIANCE and L-ALLIANCE architectures have been successfully implemented in a variety of proof of concept applications on both physical and simulated mobile robots. The applications implemented include "mock" hazardous waste cleanup [27, 30], simple cooperative manipulation [31], cooperative observation of multiple moving targets [33], cooperative box pushing (described in this section), janitorial service [28], and bounding overwatch [28]. The cooperative box pushing task offers a simple and straight-forward illustration of the key characteristics of the ALLIANCE and L-ALLIANCE architectures: fault tolerant action selection and adaptive control due to dynamic changes in the robot team, including increased heterogeneity. This box pushing task requires a long box to be pushed across a room; the box is sufficiently heavy and long that one robot cannot push at the fulcrum of the box to move it across the room. Thus, the box must be pushed at both ends in order to accomplish this task. To synchronize the pushing at the two ends, the task is defined in terms of two recurring subtasks — (1) push a little on the left end, and (2) push a little on the right end — neither of which can be activated (except for the first time) unless the opposite side has just been pushed. Note that our emphasis in these experiments is on issues of fault tolerant cooperation and adaptation rather than in the design of the ultimate box pusher. Others have demonstrated the ability to cooperatively push objects in various ways (e.g., [8, 40]), but without demonstrating capabilities for dynamic action selection due to changes in the robot team, including increased heterogeneity. Thus, we are not concerned here with issues such as robots pushing the box into a corner, obstacles interfering with the robots, how robots detect box alignment, and so forth. In [32], we reported on the basic box pushing experiment from the ALLIANCE perspective. In this chapter, we describe the results using the L-ALLIANCE mechanisms.

The box pushing application was implemented using three mobile robots of two types — two R-2 robots and one Genghis-II, which we will call *BLUE*, *GREEN*, and *GENGHIS*. The first type of robot, the R-2, has two drive wheels arranged as a differential pair, and a pair of gripper fingers in front. Its sensor suite includes eight infrared sensors and seven bump sensors evenly distributed around the front, sides, and back of the robot. The second type of robot, Genghis-II, is a legged robot with six two-degree-of-freedom legs. Its sensor suite includes two whiskers and force detectors on each leg.

A radio communication system was used in our physical robot implementations to allow the robots to communicate their current actions to each other. All of the results described in this section involve communication between robots at no more than about $\frac{1}{3}$ Hertz.

Robot Software Design

Since the capabilities of the R-2 and Genghis-II robots differ, the software design of the box pushing application for these robots varies somewhat.

R-2 Control. Figure 17 shows the L-ALLIANCE implementation of the box pushing application for the R-2 robots *BLUE* and *GREEN*. As shown in this figure, the R-2 is controlled by two behavior sets — one for pushing a little on the left end of the box (called *push_left*), and one for pushing a little on the right end of the box (called *push_right*). As specified by the L-ALLIANCE architecture, the activation of each of these behavior sets is controlled by a motivational behavior. We now examine the design of the *push_left* motivational behavior and the *push_left* behavior set of a robot r_i in more detail; the *push_right* design is symmetric to that of *push_left*.

The sensory feedback required before the *push_left* motivational behavior within r_i can activate its behavior set is an indication that the right end of the box has just been pushed. This requirement is indicated in Figure 17 by the *pushed-at-right* arrow entering the *push_left* motivational behavior.

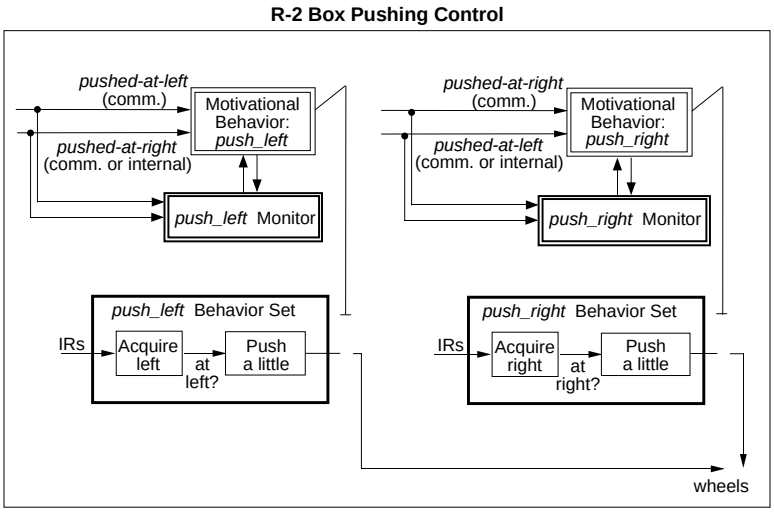


Figure 17: The L-ALLIANCE design of the R-2 software for the box pushing application.

The right end of the box can be pushed either by some robot other than r_i, or it can be pushed by r_i itself. If r_i is the robot doing the pushing, then the *pushed-at-right* feedback comes from an internal message from r_i's *push_right* motivational behavior. However, if some robot other than r_i is pushing, then r_i must detect when that other robot has completed its push. Since this detection is impossible for the R-2s with their current sensory suites, the robots are provided with this capability by having the team members broadcast a message after each push that indicates the completion of their current push.

When the sensory feedback is satisfied, the *push_left* motivational behavior grows impatient at either a rate $\delta\_fast_{i,push\_left}(t)$ (the i subscript stands for either *BLUE* or *GREEN*) if no other robot is performing the *push_left* task, or at a rate $\delta\_slow_{i,push\_left}(k,t)$ when robot r_k is performing the *push_left* task. When the *push_left* motivation grows above threshold, the *push_left* behavior set is activated. The *push_left* behavior set involves first acquiring the left end of the box and then pushing a little on that end. If the robot is already at the left end of the box, then no acquiring has to take place. Otherwise, the R-2 assumes it is at the right end of the box, and moves to the left end of the box by using the infrared sensors on its right side to follow the box to the end, and then backing up and turning into the box. As we shall see below, this ability to acquire the opposite end of the box during the application is important to achieving fault tolerant cooperative control. An implicit assumption is made that at the beginning of the task, the R-2 is facing into a known end of the box.

As the R-2 pushes, it uses the infrared sensors at the ends of its gripper fingers to remain in contact with the box. The current push is considered to be complete when the R-2 has pushed for a prescribed period of time. After the *push_left* task is completed, the motivation to perform that task temporarily returns to 0. However, the motivation begins growing again as soon as the sensory feedback indicates the task is needed.

Genghis-II Control. *GENGHIS* and the R-2s are different in two primary ways. First, *GENGHIS* cannot acquire the opposite end of the box, due to a lack of sensory capabilities, and second, *GENGHIS* cannot push the box as quickly as an R-2, due to less powerful effectors. The first difference means that *GENGHIS* can only push at its current location. Thus, implicit in the control of *GENGHIS* is the assumption that it is located at a known end of the box at the beginning of the task. The second difference with the R-2s implies that if an R-2 pushes with the same duration,

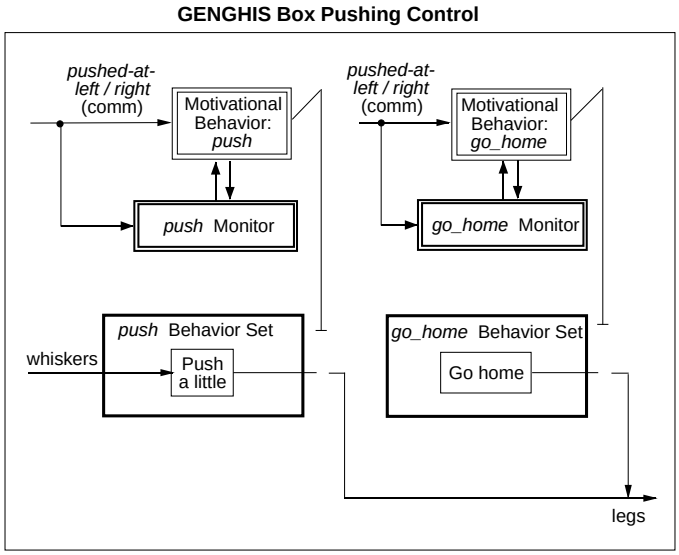


Figure 18: The L-ALLIANCE design of the *GENGHIS* software for the box pushing application.

speed, and frequency when teamed with *GENGHIS* as it does when teamed with another R-2, the robot team will have problems accomplishing its task due to severe box misalignment.

Figure 18 shows the organization of *GENGHIS*'s box pushing software. As this figure shows, *GENGHIS* is controlled by two behavior sets, each of which is under the control of a motivational behavior. *GENGHIS*'s pushing at its current location is controlled by the *push* behavior set. The only sensory feedback which satisfies the *push* motivational behavior is that which indicates that some other robot is pushing the opposite end of the box. This requirement is shown in Figure 18 as the *pushed-at-left/right* arrow going into the *push* motivational behavior. Once the sensory feedback is satisfied, *GENGHIS* becomes impatient to perform the *push* behavior at a rate $\delta\_fast_{GENGHIS,push}(t)$. Once the motivation crosses the threshold of activation, the *push* behavior set is activated, causing *GENGHIS* to push the box by walking into it while using its whiskers to maintain contact with the box. Once *GENGHIS* has pushed a given length of time, the motivation to perform *push* returns to 0, growing again whenever the sensory feedback is satisfied.

The sensory feedback required for the *go_home* behavior set to be activated is the opposite of that required for the *push* behavior set — namely, that no other robot is pushing at the opposite end of the box. When the sensory feedback for *go_home* is satisfied, the motivation to activate *go_home* grows at the rate $\delta\_fast_{go\_home}(t)$, with the behavior set being activated as soon as the motivation crosses the threshold. The *go_home* behavior set causes *GENGHIS* to walk away from the box; in effect, *GENGHIS* gives up on this task when no other robot is present to push at the opposite end of the box.

Experiments

To analyze the L-ALLIANCE architecture, we undertook two basic experiments using the box pushing application. Both of these experiments began with two R-2s (*GREEN* and *BLUE*) pushing the box — one at each end of the box — as illustrated in Figure 19.

After the two R-2s push the box for a while we dynamically altered the capabilities of the robot team in two ways. In the first experiment, we altered the team by seizing one of the R-2 robots (*GREEN*) during the task and turning it off, mimicking a robot failure; we then later added it

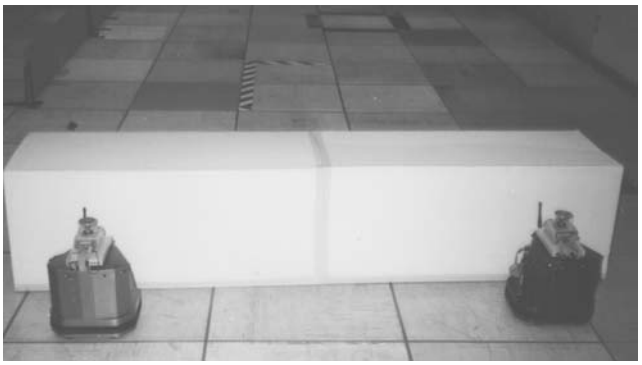


Figure 19: The beginning of the box pushing application. Two R-2s are pushing the box across the room. (The R-2 robot *GREEN* is on the left, the R-2 robot *BLUE* is on the right.)

back into the team. In the second experiment, we again seized one of the R-2 robots (*GREEN*), but this time we replaced it with *GENGHIS*, thus making the team much more heterogeneous; we then later seized the remaining R-2 robot (*BLUE*), leaving *GENGHIS* as the sole team member. While the experiments reported here are for a relatively short snapshot of time, the L-ALLIANCE approach is designed to continually adapt the actions of robots over time to achieve the goal of lifelong implementations of heterogeneous multi-robot teams.

Experiment 1: Robot "failure". By seizing an R-2 (in this case, *GREEN*) and turning it off, we test the ability of the remaining R-2 (*BLUE*) to respond to that "failure" and adapt its action selection accordingly. In this experiment, what we observe after the seizure is that after a brief pause (which is dependent upon the setting of the $\delta\_slow_{BLUE,j}$(k,t) parameter), *BLUE* begins acquiring the opposite end of the box, as shown in Figure 20, and then pushes at its new end of the box. *BLUE* continues its back and forth pushing, executing both tasks of pushing the left end of the box and pushing the right end of the box as long as it fails to hear through the broadcast communication mechanism that another robot is performing the push at the opposite end of the box. When we add *GREEN* back in, however, *BLUE* adapts its actions again, now just pushing one side of the box, since it is satisfied that the other end of the box is also getting pushed. Thus, the robot team demonstrates its ability to recover from the failure of a robot team member.

Figure 21 shows the motivational trace of these robot during this experiment. This figure shows the two robots, *BLUE* and *GREEN*, each with two behavior sets, *push_right* and *push_left*. In these traces, the dashed lines show the extent of the activation levels – from 0 to the threshold θ.

Experiment 2: Increased heterogeneity. In the second experiment, by substituting robots during the middle of a task, we test the ability of the remaining team member to respond to the dynamic change in the heterogeneity of the team.

What we observe in this experiment is that the remaining R-2 (here, *BLUE*) begins pushing much less frequently as soon as it hears that *GENGHIS*, rather than an R-2, is the robot pushing the opposite end of the box. Thus, the robots remain more or less aligned during their pushing. Figure 22 illustrates *BLUE* and *GENGHIS* pushing together. Figure 23 shows the trace of the levels of activation of the behavior sets during this experiment.

The reduced rate of pushing in *BLUE* when *GENGHIS* is added is caused by the following. First, *BLUE*'s $\delta\_slow_{BLUE,j}$(*GREEN*,t) and $\delta\_slow_{BLUE,j}$(*GENGHIS*) parameters differ somewhat since *GENGHIS* is slower at pushing the box than the R-2s. In addition, once *GENGHIS* is swapped into the team, it takes longer to complete its pushing than *GREEN* did. This in turn causes the sensory feedback of *BLUE*'s *push_left* motivational behavior to not be satisfied as frequently, and

Figure 20: Fault tolerant action selection. In the first experiment, we seize *GREEN* and turn it off. This causes *BLUE* to have to perform both tasks of the box pushing application: pushing at the right end of the box, and pushing at the left end of the box. Here, *BLUE* is acquiring the right end of the box.

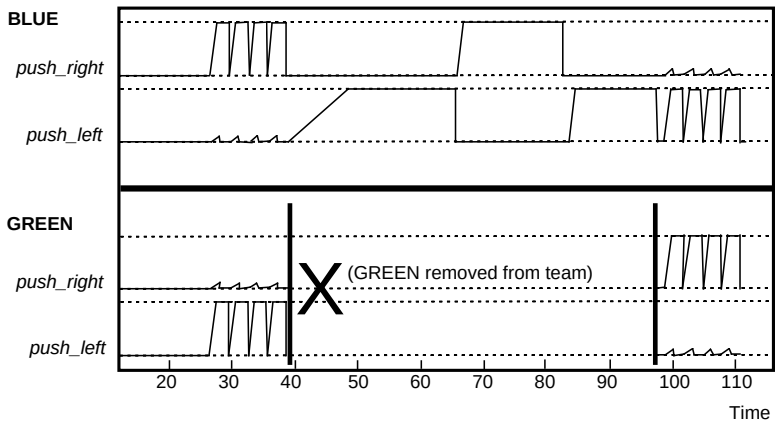


Figure 21: Motivational levels for behavior sets during Experiment 1.

Figure 22: Adaptivity due to heterogeneity. In the second experiment, we again seize one of the R-2 robots, but this time we replace it with *GENGHIS*. Since *GENGHIS* cannot push as powerfully as an R-2, the remaining R-2 robot adapts its actions by pushing less frequently.

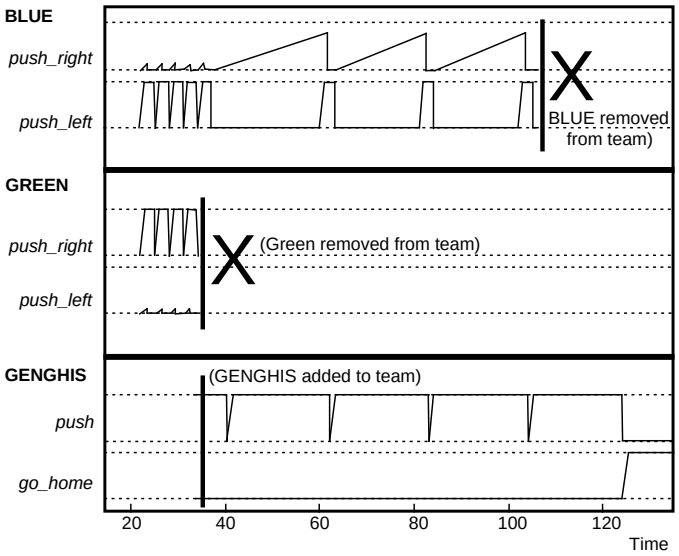


Figure 23: The trace of the activation levels of the behavior sets during Experiment 2.

thus *BLUE*'s *push_left* behavior set cannot be activated as frequently. In the meantime, *BLUE*'s *push_right* motivational behavior is becoming more impatient to activate the *push_right* behavior set since it is not hearing that any other robot is accomplishing its task. However, since the *push_right* motivation is now growing at a reduced rate of impatience, $\delta\_slow_{BLUE,push\_right}(GENGHIS)$, the motivation to activate the *push_right* behavior set does not cross the threshold of activation before *GENGHIS* announces its completion of the task. This in turn prevents *BLUE* from taking over the push of the right side of the box as long as *GENGHIS* continues to push. In this manner, *BLUE* demonstrates its ability to adapt to a dynamic change in team heterogeneity.

We complete this experiment by removing *BLUE* from the team. This causes *GENGHIS* to activate its *go_home* behavior, since it cannot complete the box pushing task on its own. Thus, *GENGHIS* also demonstrates its adaptive action selection due to the actions and failures of robot team members.

Discussion

The types of situations that L-ALLIANCE is designed to handle over the lifetime of a mission include new team composition, changes in individual robot capabilities, environmental changes, and mission changes. When the composition of the team changes, the immediate effect on the action selection decisions is dependent upon the previous knowledge teammates have of the new team members. If prior experience is available, stored in the form of the quality measurements of the robots performing tasks pertinent to the current mission, then the effect on action selections is immediate and efficient. If no prior knowledge is available, then the team will improve its performance over time as the new robot performs tasks under the observation of other robot team members. The speed of this improvement is dependent upon the value of μ – the number of trials over which performance data is maintained.

Since the robot team members are continually monitoring the performance of their teammates and updating the performance measurements accordingly, the response to improved or degraded capabilities is also automatic (again, depending upon the value of μ), regardless of the length of the mission. Thus, by simply incorporating the ongoing observation of the quality of performance

(here, time of task completion) into the parameter update mechanism, the parameters upon which action selections are made will continually adapt over time as the situation changes.

4 Summary and Conclusions

The field of multi-robot learning is still in its infancy, and much work still remains. The problem is particularly challenging due to very large state spaces, uncertain credit assignment, limited training time, uncertainty in sensing and shared information, nondeterministic actions, difficulty in defining appropriate abstractions for learned information, and difficulty of merging information learned from different robot experiences. However, the potential benefits are significant, and include increased robustness, reduced programming complexity, and increased ease of adding new assets to the team.

This chapter has not made an attempt to provide a complete summary of multi-robot learning techniques that have been explored by the field as a whole, although we have mentioned several related approaches. Instead, we have focused on three specific techniques we have developed for learning in multi-robot teams. The first two deal with the learning of inherently cooperative tasks – tasks that cannot be broken into distinct subtasks to be executed independently by robot team members. We compared these two learning approaches to a human-generated solution to the CMOMMT (Cooperative Multi-Robot Observation of Multiple Moving Targets) task and found that, while they do not reach or surpass the human-generated solution, they achieve significant improvements over naive approaches. These developments are important because they are some of the first techniques developed to enable robot teams to learn in the very challenging domain of inherently cooperative tasks. The third multi-robot learning approach we present addresses the issue of automatic parameter updates to enable robot team members to adapt their performance over time as they gain experience working with their teammates. This approach is significant because it gives the robots the ability to automatically adapt to the heterogeneous nature of their teammates over the lifetime of their work together.

Our results show that the learning approaches, while not yet superior to human-generated solutions, offer much promise for eventually achieving the goal of enabling robot teams to learn cooperative control approaches autonomously. The ultimate goal of this research is to develop techniques for multi-robot learning and adaptation that will generalize to cooperative robot applications in many domains, thus facilitating the practical use of multi-robot teams in a wide variety of real-world applications.

Acknowledgements

This research was funded in part by the Engineering Research Program, Office of Science, of the U.S. Dept. of Energy. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the U.S. Dept. of Energy under contract DE-AC05-00OR22725.

References

[1] D. Aha, editor. *Lazy Learning.* Kluwer Academic Publishers, 1997.

[2] M. Asada, E. Uchibe, S. Noda, S. Tawaratsumida, and K. Hosoda. Coordination of multiple behaviors acquired by a vision-based reinforcement learning. In *Proceedings of IEEE RSJ GI*

International Conference on Intelligent Robots and Systems, pages 917–924, Munich, Germany, 1994.

[3] Minoru Asada, Shoichi Noda, and Koh Hosoda. Action-based sensor space categorization for robot learning. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1502–1509, 1996.

[4] M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, August 1985.

[5] Rodney A. Brooks and Maja J. Mataric. Real robots, real learning problems. In Jonathan H. Connell and Sridhar Mahadevan, editors, *Robot Learning*. Kluwer Academic Publishers, 1993.

[6] David Chapman and Leslie P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.

[7] C. Claussen, S. Gutta, and H. Wechsler. Reinforcement learning using funtional approximation for generalization and their application to cart centering and fractal compression. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1362–1367, Stockholm, Sweden, August 1999.

[8] Bruce Randall Donald, James Jennings, and Daniela Rus. Towards a theory of information invariants for cooperating autonomous mobile robots. In *Proceedings of the International Symposium of Robotics Research*, Hidden Valley, PA, October 1993.

[9] Fernando Fernández and Daniel Borrajo. Iterative VQQL for learning skills. In *Proceedings of Learning'00*, Leganés, Madrid, Spain, 2000.

[10] Fernando Fernndez and Daniel Borrajo. Vector quantization applied to reinforcement learning. In Manuela M. Veloso, editor, *Proceedings of the Third International Workshop on RoboCup*, pages 97–102, Stockholm, Sweden, August 1999. IJCAI'99.

[11] Fernando Fernndez and Daniel Borrajo. VQQL. Applying vector quantization to reinforcement learning. In *RoboCup-99: Robot Soccer World Cup III*. Springer Verlag, 2000.

[12] Thomas Haynes and Sandip Sen. Evolving behavioral strategies in predators and prey. In Gerard Weiss and Sandip Sen, editors, *Adaptation and Learning in Multi-Agent Systems*, pages 113–126. Springer, 1986.

[13] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Int. J. of Artificial Intelligence Research*, pages 237–285, 1996.

[14] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Learning Robots*, pages 19–24, December 1995.

[15] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer, Berlin, Heidelberg, 1984. 3rd ed. 1989.

[16] Teuvo Kohonen. Learning vector quantization for pattern recognition. Report TKK-F-A601, Helsinki University of Technology, Espoo, Finland, 1986.

[17] R. Korf. A simple solution to pursuit games. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194, 1992.

[18] Ben J.A. Krose and Joris W.M. van Dam. Adaptive state space quantization for reinforcement learning of collision-free navigation. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'92)*, volume 2, pages 1327–1332, 1992.

[19] Masao Kubo and Yukinori Kakazu. Learning coordinated motions in a competition for food between ant colonies. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 487–492. MIT Press, 1994.

[20] Yoseph Linde, André Buzo, and Robet M. Gray. An algorithm for vector quantizer design. In *IEEE Transactions on Communications, Vol1. Com-28, N 1*, pages 84–95, 1980.

[21] S. P. Lloyd. Least squares quantization in pcm. In *IEEE Transactions on Information Theory*, number 28 in IT, pages 127–135, March 1982.

[22] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *Proceedings of AAAI-91*, pages 8–14, 1991.

[23] S. Marsella, J. Adibi, Y. Al-Onaizan, G. Kaminka, I. Muslea, and M. Tambe. On being a teammate: Experiences acquired in the design of RoboCup teams. In O. Etzioni, J. Muller, and J. Bradshaw, editors, *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 221–227, 1999.

[24] Maja Mataric. Learning in multi-robot systems. In Gerhard Weiss and Sandip Sen, editors, *Adaption and Learning in Multi-Agent Systems*. Springer, 1996.

[25] Andrew W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued spaces. *Proceedings in Eighth International Machine Learning Workshop*, 1991.

[26] Rmi Munos and Andrew Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In Thomas Dean, editor, *Proceedings of Sixteenth International Joint Coference on Artificial Intelligence*, volume 2, pages 1348–1355, Stockholm, Sweden, August 1999.

[27] L. E. Parker. *Heterogeneous Multi-Robot Cooperation*. PhD thesis, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, February 1994. MIT-AI-TR 1465 (1994).

[28] L. E. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 1996.

[29] L. E. Parker. L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Journal of Advanced Robotics*, 1997.

[30] L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[31] L. E. Parker. Distributed control of multi-robot teams: Cooperative baton-passing task. In *Proceedings of the 4th International Conference on Information Systems Analysis and Synthesis (ISAS '98)*, volume 3, pages 89–94, 1998.

[32] L. E. Parker. Adaptive heterogeneous multi-robot teams. *Neurocomputing, special issue of NEURAP '98: Neural Networks and Their Applications*, 28:75–92, 1999.

[33] L. E. Parker. Cooperative robotics for multi-target observation. *Intelligent Automation and Soft Computing, special issue on Robotics Research at Oak Ridge National Laboratory*, 5(1):5–19, 1999.

[34] L. E. Parker. Lifelong adaptation in heterogeneous teams: Response to continual variation in individual robot performance. *Autonomous Robots*, 8(3), July 2000.

[35] L. E. Parker. Distributed algorithms for multi-robot observation of multiple moving targets. *under revision for publication in Autonomous Robots*, 2001.

[36] L. E. Parker and C. Touzet. Multi-robot learning in a cooperative observation task. In *Distributed Autonomous Robotic Systems 4*, pages 391–401. Springer, 2000.

[37] Tsutomu Sawada, Sumiaki Ichikawa, and Fumio Hara. Autonomous action-mode change in a two-mobile robotics system. s-temperature based on-line learning. In *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'99)*, volume 1, pages 393–399, 1999.

[38] J.W. Sheppard and S.L. Salzberg. A teaching strategy for memory-based control. In D. Aha, editor, *Lazy Learning*, pages 343–370. Kluwer Academic Publishers, 1997.

[39] Randall Steeb, Stephanie Cammarata, Frederick Hayes-Roth, Perry Thorndyke, and Robert Wesson. Distributed intelligence for air fleet control. Technical Report R-2728-AFPA, Rand Corp., 1981.

[40] D. Stilwell and J. Bay. Toward the development of a material transport system using swarms of ant-like robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 766–771, Atlanta, GA, 1993.

[41] P. Stone and M. Veloso. A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.

[42] Claude Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics and Autonomous Systems*, 1997.

[43] A. Ueno, K. Hori, and S. Nakasuda. Simultaneous learning of situation classification based on rewards and behavior selection based on the situation. In *Proceedings of the 1996 IEEE/RSJ International Conference on Intelligence Robots and Systems (IROS'96)*, volume 3, pages 1510–1517, 1996.

[44] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

[45] Gerhard Weiss and Sandip Sen, editors. *Adaption and Learning in Multi-Agent Systems*. Springer, 1996.