Autonomous Robots, ??, 1?? (??)


# Lifelong Adaptation in Heterogeneous Multi-Robot Teams: Response to Continual Variation in Individual Robot Performance

**LYNNE E. PARKER**

*ParkerLE@ornl.gov*
*Center for Engineering Science Advanced Research, Computer Science and Mathematics Division*
*Oak Ridge National Laboratory, P. O. Box 2008, Oak Ridge, TN 37831-6355 USA*

**Abstract.**

Generating teams of robots that are able to perform their tasks over long periods of time requires the robots to be responsive to continual changes in robot team member capabilities and to changes in the state of the environment and mission. In this article, we describe the L-ALLIANCE architecture, which enables teams of heterogeneous robots to dynamically adapt their actions over time. This architecture, which is an extension of our earlier work on ALLIANCE, is a distributed, behavior-based architecture aimed for use in applications consisting of a collection of independent tasks. The key issue addressed in L-ALLIANCE is the determination of which tasks robots should select to perform during their mission, even when multiple robots with heterogeneous, continually changing capabilities are present on the team. In this approach, robots monitor the performance of their teammates performing common tasks, and evaluate their performance based upon the time of task completion. Robots then use this information throughout the lifetime of their mission to automatically update their control parameters. After describing the L-ALLIANCE architecture, we discuss the results of implementing this approach on a physical team of heterogeneous robots performing proof-of-concept box pushing experiments. The results illustrate the ability of L-ALLIANCE to enable lifelong adaptation of heterogeneous robot teams to continuing changes in the robot team member capabilities and in the environment.

## 1. Introduction

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.

One important consequence of dynamic changes in lifelong multi-robot systems is that continuing drift in individual robot capabilities creates a team of heterogeneous robots, even if the original team was designed to be homogeneous. This may actually be true at the outset of the application. Experienced

roboticists are aware that several copies of the same model of robot, even direct from the manufacturing line, can vary in capabilities due to differences in sensor tuning, etc. Over time, the minor initial differences among robots will grow, since sensors and effectors may degrade or break on certain robots, or for learning robots, some robot team members may learn faster than others. Thus, mechanisms for generating lifelong multi-robot teams must of necessity deal with the issue of heterogeneity among robot team members.

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. Even in the case of full global knowledge, however, the problem of optimally assigning tasks to robots that have different, but overlapping capabilities can be easily shown to be NP-hard. Thus, the multi-robot team control mechanism must have some effective approximate means of distributing tasks so that an acceptable level of efficiency is achieved without sacrificing the desirable features of fault tolerance and robustness.

In previous work [31], 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. In [30], we provided a brief overview of the L-ALLIANCE approach. In this article, we develop this approach more fully, and show detailed results of its implementation in a physical multi-robot team in the context of L-ALLIANCE. Thus, the key issue addressed in this article is:

> *Given a team of robots and a mission that they are to perform that consists of several independent tasks*[1]*, 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?*

In section 2, we show that the general heterogeneous multi-robot action selection problem is NP-hard, leading to the need for an approximate approach that works well in practice. We provide a brief overview of the ALLIANCE approach in section 3, followed in section 4 with an overview of the L-ALLIANCE mechanism for lifelong adaptation in heterogeneous multi-robot teams. In section 5, we discuss the selection of the appropriate action selection strategy, and show in section 6 how this action selection strategy is implemented in the L-ALLIANCE formal model. In section 7, we report and discuss the results of the implementation of this approach in a box pushing application. In section 8, we describe the related work in this area. Finally, in section 9, we offer concluding remarks and future directions for this research.

## 2. NP-Hardness of The Heterogeneous Robot Action Selection Problem

In[30], we argued (without giving the proof) that even a simplified version of the action selection problem for heterogeneous robots is NP-hard. Here, we provide the details of this proof.

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 $task_j$ 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 $q(a_{ix}) = q(a_{jy})$. For the simplified case in this section, we will assume that these robot performance measurements are known in advance.

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: $max_i(\sum_{a_{ik} \in U_i} q_{time}(a_{ik}))$.

The first two constraints of the above problem definition ensure that each task in the mission is assigned to some robot that can actually accomplish that task. The final constraint is an example optimization function for one possible quality metric – time. Since robot team members usually perform their actions in parallel during a mission, the total mission completion time is the time at which the last robot finishes its final task. Thus, we want to minimize the maximum amount of time any robot will take to perform its set of actions.

We note here that for reasons of robustness, fault tolerance, and flexibility, distributed decisionmaking very often gives better results than a centralized decisionmaker. Thus, while this efficiency 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 [15]. The PARTITION problem is as follows: given a finite set $W$ and a "size" $s(w) \in Z^+$ for each $w \in W$, determine whether there is a subset $W' \subseteq W$ such that $\sum_{w \in W'} s(w) = \sum_{w \in W - W'} s(w)$. We then have the following:

THEOREM 1 *The HRASP efficiency problem is NP-hard in the number of tasks required by the mission.*

*Proof:*
By restriction to PARTITION:

Allow only instances of HRASP where:

- $n = 2$ (i.e., $R = (r_1, r_2)$)
- $A_1 = A_2 = W$
- $T = \bigcup_{r_i \in R, a_{ij} \in A_i} (h_i(a_{ij}))$
- $\forall(r_i \in R).\forall(a_{ij} \in A_i).(h_i(a_{ij}) = task_j)$

- $\forall(a_{ij} \in A_i).(q(a_{1j}) = q(a_{2j}) = s(w_j))$, for $w_j \in W$.

Then since PARTITION is a special case of HRASP, HRASP must be NP-hard.

□

Since the PARTITION problem is stated in terms of finding two equally-sized subsets of tasks $W$ and $W'$, the proof of this theorem restricts HRASP to those instances involving two robots with identical capabilities and qualities of capabilities. Furthermore, each robot has the same one-to-one mapping of behavior sets to tasks, meaning that all robots use the same behavior set to accomplish the same task, and all behavior sets are needed to accomplish the mission. These HRASP instances are then instances of PARTITION, so that, if we could solve HRASP, we could solve PARTITION. In other words, even this simpler problem involving homogeneous robots with equal costs is NP-hard. The real-world problems involving heterogeneous robots with unequal costs are certainly no easier, and thus this action selection problem is shown to be NP-hard.

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. Brief Overview of ALLIANCE

The foundation of our approach to achieve lifelong heterogeneous multi-robot adaptation is the ALLIANCE architecture (see Figure 1), which we developed in previous work [31]. 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 distributed, 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

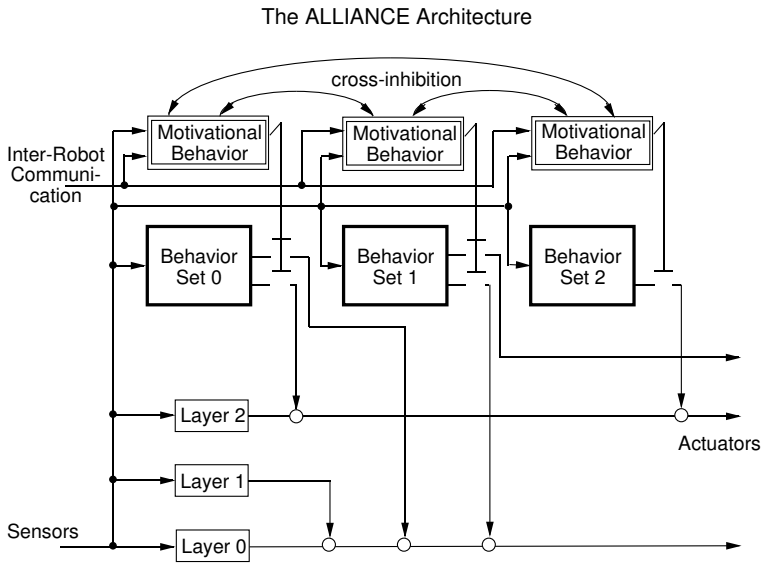


*Fig. 1.* The ALLIANCE architecture defines the design of an individual robot, such that when the robot is brought together with other robots also designed using ALLIANCE, they exhibit dynamic action selection.

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.

More details of the ALLIANCE approach are embedded in the following section, which discusses the L-ALLIANCE extension to ALLIANCE. 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 [29], [31].

## 4. Overview of L-ALLIANCE

In [30] we described the derivation of the L-ALLIANCE mechanism, explaining the experimental process by which the L-ALLIANCE action selection methodology was derived. For clarity, we summarize the main approach in this and the following section. Then, in section 6, we provide the details of how this control approach is implemented into the L-ALLIANCE formal model. Note that the L-ALLIANCE formal model, while built upon the ALLIANCE formal model reported in [31], [33], is a significant extension to the ALLIANCE framework, since it defines how the parameters of the formal model can be set automatically, enabling the robot team to autonomously adapt throughout their mission.

Many efficiency issues are of importance in designing lifelong heterogeneous multi-robot cooperative teams. These issues include:

- How do we ensure that robots attempt those tasks for which they are best suited, even as team member capabilities evolve over time?

- Can we enable the robot team to increase its performance over time as it becomes familiar with team member capabilities?
- Does failure at one task imply total robot failure?
- How does a robot select a method of performing a task if it has more than one way to accomplish that task?
- How do we minimize robot idle time?

The L-ALLIANCE extension to ALLIANCE addresses these 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:

- How to give robots the ability to obtain knowledge about the quality of team member performances, and
- How to use team member performance knowledge to select a task to pursue.

Solutions to the first problem require a robot to learn not only about the abilities of its teammates, but also about its own abilities. Although each robot "knows" the set of behaviors that it has been programmed to perform, it may perform poorly at certain tasks relative to other robots on the team. Robots must thus learn about these relative performance differences as a first step toward efficient mission execution and adaptation over time. However, learning these relative performance quality differences is only a first step towards lifelong adaptation. The next, more major, question is how robots use the performance knowledge to efficiently adapt their own actions.

### *4.1. Assumptions in L-ALLIANCE*

Two key assumptions are made in L-ALLIANCE, as follows:

- A robot's average performance in performing a specific task over a few recent trials is a reasonable indicator of that robot's expected performance in the future.
- If robot $r_i$ is monitoring environmental conditions $C$ to assess the performance of another robot $r_k$, and the conditions $C$ change, then the changes are attributable to robot $r_k$.

Without the first assumption, it is quite difficult for robots to learn about their own expected performance, or the performance of their teammates, since past behavior provides no clues to the expected behavior in the future. The challenge is to determine those aspects of a robot's performance that are good predictors of future performance. Time of task completion is one indicator that is useful in many applications.

The second assumption deals with the well-known credit assignment problem, which is concerned with determining which process should receive credit (or punishment) for the successful (or unsuccessful) outcome of an action. The assumption in L-ALLIANCE is that the only robot that affects the properties of the world that robot $r_i$ is interested in are the robots that $r_i$ is monitoring. Thus, if a robot $r_k$ declares it is performing some task, and that task becomes complete, then the monitoring robot will assume that $r_k$ caused those effects. This assumption is certainly not always true, since external agents can intrude on the robots' world. However, since this issue even causes problems for biological systems, which often have difficulty in correctly assigning credit, this oversimplification seems acceptable.

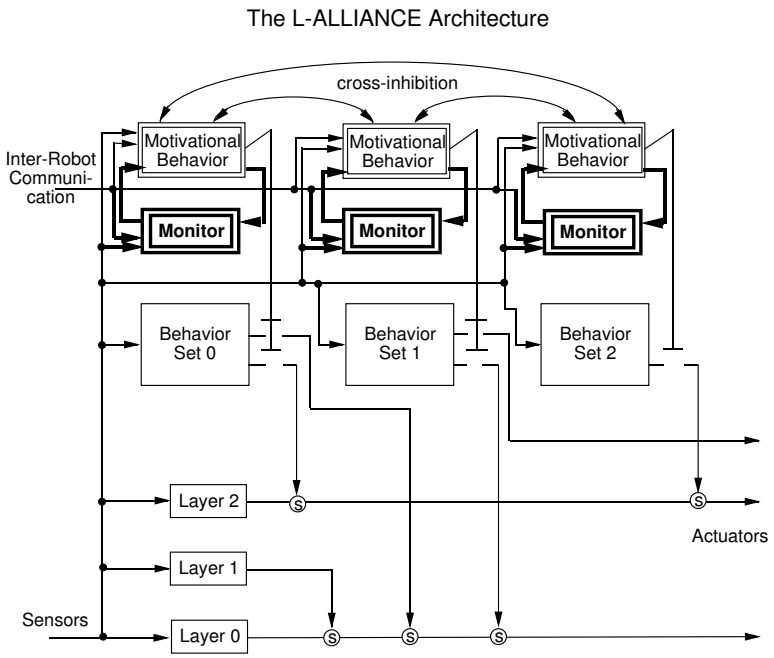


*Fig. 2.* The L-ALLIANCE architecture extends ALLIANCE by adding performance monitors that measure the performance of robot team members for certain tasks.

### *4.2. L-ALLIANCE Performance Monitors*

Of central importance to the learning mechanism used in L-ALLIANCE is the ability of robots to monitor, evaluate, and catalog the performance of team members in executing certain tasks. Without this 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. In either case, 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 2 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 6 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.

### *4.3. 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 is 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.

**Active Learning Phase.** Clearly, the only way robots can independently learn about their own abilities and the abilities of their teammates is for the robots to activate as many of their behavior sets as possible during a mission, and to monitor their own progress and the progress of team members during task execution. Of course, on any given mission, not all of the available behavior sets may be appropriate, so it is usually not possible to learn complete information about robot capabilities from just one mission scenario. However, the *active learning* phase allows the team to obtain as much information as possible through the active exploration of robot abilities. In this 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. Since robots at the beginning stages of learning do not yet know how long it may take them to perform their tasks, this maximal patience/minimal acquiescence feature allows them to try as long as needed to accomplish their tasks. Of course, if a robot has the ability to detect failure with certainty, then it can give up failed tasks to another team member.

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. For example, if the quality measure is time, then the robot stores the average time plus one standard deviation in the time required for that robot to perform the task.

**Adaptive Learning Phase.** When a robot team is applied to a "live" mission, it 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.

## 5. Action Selection Strategy

In [30], 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

| Strategy | Impatience with other robots | Acquiescence |
|---|---|---|
| I | own time | own time |
| II | own time | min. time of team |
| III | time of robot performing $h_i(a_{ij})$ | own time |

*Table 1.* The impatience and acquiescence parameters are functions of the values shown in the table, for each of three dynamic task reallocation strategies.

observed by the robots during mission performance. The objective is to allow the robot team to improve its efficiency over time while not sacrificing the fault tolerant characteristics of the behavior-based ALLIANCE architecture. The ultimate 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 obvious implication of this ability to autonomously adapt L-ALLIANCE parameters is the elimination of the need for the human to guess on the appropriate settings for the robot team throughout its mission.

For completeness, we now briefly summarize the results of our earlier investigations to derive an efficient, distributed robot action selection methodology. Refer to [30] for more details on these investigations.

In these studies, we used time of task completion as the metric that determined the quality of robot performance. Thus, each robot monitored other robot performances of tasks it is able to accomplish, recording information on the time of task completion. Note that robots do not need to monitor all tasks by all robots – only those tasks which it, itself, has the ability to perform. Determining the proper conversion of these relative quality measurements (i.e., time of task completion) into the control parameters of L-ALLIANCE requires solutions to two related subproblems:

- *Dynamic task reallocation* — how an individual robot determines whether to interrupt the task currently being performed by another robot (i.e., become impatient), or whether it should acquiesce its own current task, and
- *Task ordering* — how an individual robot selects from among a number of incomplete tasks that no other team member is currently performing.

To study the first of these issues, we explored three dynamic task reallocation strategies. Summarized in table 1, these strategies vary in which quality measurements are used to determine when a robot should (1) take over tasks from other robots that are not demonstrating their ability to successfully accomplish their tasks, and (2) give up tasks that the robot itself is not able to successfully accomplish.

Additionally, for the second issue above, we studied three task ordering strategies to determine how an individual robot should select from among a number of incomplete tasks that no other team member is currently performing. As before, these strategies vary in which quality measurements are used to determine the selection decision. The three strategies based their next action selection decision on either a greedy choice based upon (1) the longest or (2) shortest expected execution time of the tasks it is able to perform, or (3) a random choice of actions.

To determine the relative merits of these various control strategies, we executed a large number of test runs in simulation, comparing the results of all of the combinations of the dynamic task reallocation and task ordering strategies in terms of the time required to complete the mission. We collected performance data by varying the number of robots on the team ($n$) from 2 to 20, the number of tasks the team must perform ($m$) from 1 to 40, the task coverage[2] from 1 to 10, the degree of heterogeneity[3] from 0 percent to 3200 percent, and the value of the *Progress When Working* (PWW) condition[4] as either true or false.

For this study, the robot capabilities to perform various tasks were distributed uniformly across the robots based upon the given task coverage, and the same task coverage was assumed for all tasks in the mission. For each 5-tuple ($n$, $m$, *task_coverage*, *heterogeneity*, PWW) of a given run of the simulation,

which we call a *scenario*, 200 randomly generated test runs were executed. The average over these 200 runs was then considered the characteristic performance of that scenario.

These results were then analyzed to determine the best overall action selection strategy that allows individual robots to make good decisions in light of the varying capabilities of the other robot team members. The goal in comparing these various strategies was to derive a single approach to multi-robot action selection that performs well regardless of the particular instantiation of the robot team and mission. We do not want to require a human system designer to perform extensive analysis of specific multi-robot applications to determine the appropriate multi-robot control strategy.

Thus, the resulting action selection strategy that was found to have the best performance is as follows:

**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 [30].

## 6. Formal Model of L-ALLIANCE

In[30], the appendix listed the formal model for L-ALLIANCE with no discussion as to how the action selection methodology from the previous section converted into the L-ALLIANCE formal model. In this section, we now describe the implementation of the derived action selection methodology into the control parameters of the L-ALLIANCE architecture. Note that this implementation provides a significant extension to the original ALLIANCE architecture, whose formal model is reported in [31]. This extension enables the parameters of the architecture to be set and adapted automatically throughout the robot team's mission.

As discussed earlier, the main mechanism controlling the robot action selection is the motivational behavior, which computes the level of activation of its corresponding behavior set as a function of the current actions of the teammates, the sensory feedback, and the internal levels of impatience and acquiescence. The parameters regulating the effect of these sources of information determine which actions the robots will select during the mission. Thus, these parameters are the key to achieving lifelong adaptation to robot team member capabilities.

In some cases, the appropriate parameter settings for the various categories of information are dependent upon the particular quality metric that is being used to evaluate robot performance. We have elected to use the metric of *time*, since this is a very common measure of interest in multi-robot team applications. (In future work we plan to explore modifications to L-ALLIANCE that are necessary to incorporate other quality metrics.)

The following discussion is organized by first describing, for a given robot $r_i$ that possesses behavior sets $A_i = \{a_{i1}, a_{i2}, ...\}$, the threshold of activation of the behavior sets, followed by a discussion of the parameter settings pertinent to each of the sources of input to a robot's motivational behavior: sensory feedback, inter-robot communication, suppression from active behavior sets, learned robot influence, robot impatience, and robot acquiescence. We then show how the L-ALLIANCE inputs are combined to compute the motivational levels of each behavior set.

### *6.1. Threshold of activation*

A parameter of key importance to the efficiency of the robot team is the threshold of activation, $\theta$. This parameter is used not only to determine the motivational level at which a behavior set is activated, but also as a way of calibrating the impatience and acquiescence rates across motivational behaviors and across robots. It is therefore important for the sake of efficiency for the value of $\theta$ to be uniform across robots and across the motivational behaviors of each robot. This uniformity should be quite easy to achieve: it can be obtained simply by the human designer broadcasting the desired value to all robots at the start of the mission, or by providing the robots with a simple arbitration mechanism that allows the team on its own to come to a consensus on the value of $\theta$.

### *6.2. Sensory feedback*

The sensory feedback provides the motivational behavior with the information necessary to determine whether its corresponding behavior set needs to be activated at a given point during the current mission. This feedback is assumed to be noisy, and can originate from either physical robot sensors or virtual robot sensors. We define a simple function to capture the notion of sensory feedback in the motivational level computation as follows:

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

### *6.3. Inter-robot communication*

The rate at which robots communicate their current actions to their teammates in L-ALLIANCE determines the level of *awareness* robot team members have of the actions of their teammates. This in turn affects the efficiency of the team's selection of actions, since lack of awareness of the actions of teammates can lead to replication of effort and decreased efficiency. Two parameters are utilized in L-ALLIANCE to control the broadcast communication among robots: $\rho_i$ and $\tau_i$.

$\rho_i$ = Rate of messages per unit time that robot $r_i$ sends concerning its current activity

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

The second parameter, $\tau_i$, provides an additional level of fault tolerance, and is important for allowing a robot to know which other robots are present and to some extent functioning on the team. While monitoring the communication messages, each motivational behavior $a_{ij}$ of robot $r_i$ must also note when a team member is pursuing task $h_i(a_{ij})$. To refer to this type of monitoring in the formal model, the function *comm_received* is defined as follows:

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

To ensure maximal efficiency in the multi-robot team, it is best to set the communication rates, $\rho_i$, to be fairly frequent relative to the time required to complete each task in the mission. Since the task completion time is usually many orders of magnitude larger than the time required to broadcast a message, it is likely that the communication system capacity easily suffices to meet this requirement.

The proper value of $\tau_i$ is dependent upon each robot team member's $\rho_i$ settings. If team members have different values for these parameters, then they cannot be sure how long to wait on messages from other robots. However, the difficulty should be minor if the $\tau_i$ values are set conservatively — say, to several times a robot's own time delay between messages. Even so, if a robot $r_i$ erroneously assumes a team member $r_k$ is no longer functional, the receipt of just one message from that team member at some point in the future is sufficient to reactivate $r_k$'s influence on $r_i$'s activities.

To refer to the team members that robot $r_i$ thinks are currently present on the team, we define the following set:

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

The $robots\_present(i,t)$ set consists of those robots $r_k$ from which $r_i$ has received some type of communication message in the last $\tau_i$ time units.

### *6.4. Suppression from active behavior sets*

For typical multi-robot missions composed of independent tasks, it is usually reasonable to expect that a robot will only pursue one task at a time. Thus, when a motivational behavior activates its behavior set, it simultaneously begins inhibiting other motivational behaviors within the same robot from activating their respective behavior sets. At this point, a robot has effectively "selected an action". In the formal model, this activity suppression is modeled by the following simple function:

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

This function says that behavior set $a_{ij}$ is being suppressed at time $t$ on robot $r_i$ if some other behavior set $a_{ik}$ is currently active on robot $r_i$ at time $t$.

### *6.5. Learned robot influence*

The variables we classify under *learned robot influence* are those that are responsible for ensuring that $r_i$ categorizes tasks correctly according to the L-ALLIANCE Action Selection Methodology. Recall that each robot should effectively categorize tasks to be performed into two categories, and then select from the first category first. One exception to this approach is during active learning. When a robot is operating in the active learning phase, it makes sense for that robot to select its next task from among those tasks that are not currently being attempted by any other robot. Thus, a task $h_i(a_{ij})$ that robot $r_i$ will consider selecting in the active learning phase is determined by the following function:

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

This function says that a robot $r_i$ considers activating a task $h_i(a_{ij})$ only if $r_i$ has not received a communication message from some robot $r_x$ on the team indicating that $r_x$ is pursuing task $h_i(a_{ij})$.

For the quality metric of *time*, the robots judge the performance of other robots based on the time it takes to complete a given task. In this case, the robots maintain information on the average time plus one standard deviation required for an individual robot to complete a given task over the last $\mu$ trials.

$\mu$ = Number of trials over which task performance averages and standard deviations are maintained

In our experiments, the value of $\mu$ can be fairly small – we found that maintaining information over about 5 trials provided good results. We thus define:

$task\_time_i(k, j, t)$ = The average time over the last $\mu$ trials of robot $r_k$'s performance of task $h_i(a_{ij})$ plus one standard deviation, as measured by robot $r_i$

In the case of robot failure, the time attributed to the failed robot is some penalty factor (greater than 1) times the actual attempted time. This penalty factor in the case of task failure is important for allowing a robot to overcome its failure to achieve one task and go on to perform some other task at which it can succeed. The important point to note is that repeated failures cause the expected completion time of the failed task to monotonically increase, leading to slower rates of impatience for the failed task. If a robot continues to select a task at which it repeatedly fails, the updates to the impatience parameters eventually cause the robot to become more impatient to perform some other task at which it *can* succeed. This, therefore, prevents the robot from getting stuck forever performing a task at which it cannot succeed while it still has some task which it could successfully complete. Of course, the larger the penalty factor, the less likely the robot will repeatedly select a task at which it cannot succeed.

As specified by the L-ALLIANCE Action Selection Methodology, we define two categories of tasks:

$$task\_category_{ij}(t) = \begin{cases} 1 \text{ if } (task\_time_i(i,j,t) = \min\limits_{r_k \in robots\_present(i,t)} task\_time_i(k,j,t)) \\ \quad \text{and } ((\sum\limits_{r_x \in robots\_present(i,t)} comm\_received(i,x,j,t-\tau_i,t)) = 0) \\ 2 \text{ otherwise} \end{cases}$$

This function says that task $h_i(a_{ij})$ belongs to the first category in robot $r_i$ at time $t$ if robot $r_i$'s expected task completion time for task $h_i(a_{ij})$ is the minimum of the robot team members that $r_i$ knows about, and if $r_i$ has not received a message from any other robot on the team, $r_x$, in the last $\tau_i$ time units which indicates that $r_x$ is currently performing task $h_i(a_{ij})$. Otherwise, the task belongs to the second category.

We note that due to the distributed implementation of L-ALLIANCE within an individual robot, the motivational behavior corresponding to a given task has no information about other tasks that may be required by the mission. Thus, a motivational behavior for a category 2 task cannot evaluate the state of the mission to make the decision as to whether all of the category 1 tasks are already completed. Thus, we implement the selective choice of all category 1 tasks first by incorporating a *boredom* factor. This boredom factor becomes active within a motivational behavior whenever the robot is idle. A robot will only select a category 2 task when its boredom reaches a certain level. Thus, this enables the robot to first activate any behavior set corresponding to a category 1 task before any of the category 2 tasks are executed.

We define the function that indicates the level of boredom of robot $r_i$ as follows. Given a boredom threshold, $boredom\_threshold_i$, and a rate of boredom, $boredom\_rate_i$, the boredom function is as follows:

$boredom\_threshold_i$ = 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 $r_i$

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

This function says that robot $r_i$'s level of boredom is 0 at time 0 and whenever some behavior set $a_{ij}$ is active on $r_i$. Otherwise, the level of boredom increments linearly over time according to the rate $boredom\_rate_i$.

The following function indicates whether the robot will consider the activation of the current behavior set:

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

The function says that robot $r_i$ considers activating a task $h_i(a_{ij})$ at time $t$ only if $r_i$ believes that it (i.e., $r_i$) is the best performer of this task (in terms of the time metric), or if the robot is bored.

### *6.6. Robot impatience*

Three parameters are used to implement the robot impatience feature of L-ALLIANCE: $\phi_{ij}(k,t)$, $\delta\_slow_{ij}(k,t)$, and $\delta\_fast_{ij}(t)$. The first parameter, $\phi_{ij}(k,t)$, gives the time during which robot $r_i$ is willing to allow robot $r_k$'s communication message to affect the motivation of behavior set $a_{ij}$.

$$\begin{aligned} \phi_{ij}(k,t) &= \text{Time during which robot } r_i \text{ is willing to allow robot } r_k\text{'s communication message to} \\ &\quad \text{affect the motivation of behavior set } a_{ij}. \\ &= \begin{cases} task\_time_i(k,j,t) & \text{if } (strategy = \text{III}) \\ task\_time_i(i,j,t) & \text{if } (strategy = \text{II}) \end{cases} \end{aligned}$$

The next two parameters, $\delta\_slow_{ij}(k,t)$ and $\delta\_fast_{ij}(t)$, give the rates of impatience of robot $r_i$ concerning behavior set $a_{ij}$ either while robot $r_k$ is performing the task corresponding to behavior set $a_{ij}$ (i.e. $h_i(a_{ij})$) or in the absence of other robots performing the task $h_i(a_{ij})$, respectively. We assume that the fast impatience parameter corresponds to a higher rate of impatience than the slow impatience parameter for a given behavior set in a given robot. The reasoning for this assumption should be clear — a robot $r_i$ should allow another robot $r_k$ the opportunity to accomplish its task before becoming impatient with $r_k$; however, there is no reason for $r_i$ to remain idle if a task remains undone and no other robot is attempting that task.

The goal is to set these parameters to cause the motivational behaviors to interact in such a way that each robot selects tasks from category 1 according to the longest task first, and to select from the category 2 tasks according to the shortest task first. Because of the definition of the two task categories, the $\delta\_slow_{ij}(k,t)$ parameters only affect tasks in the second category, which means that $\delta\_slow_{ij}(k,t)$ should grow faster than $\delta\_slow_{ip}(k,t)$ only if robot $r_i$ expects to perform task $h_i(a_{ij})$ faster than it expects to perform task $h_i(a_{ip})$. The $\delta\_slow_{ij}(k,t)$ parameter is therefore updated as follows:

$$\begin{aligned} \delta\_slow_{ij}(k,t) &= \text{Rate of impatience of robot } r_i \text{ concerning behavior set } a_{ij} \text{ after discovering robot } r_k \\ &\quad \text{performing the task corresponding to this behavior set} \\ &= \frac{\theta}{\phi_{ij}(k,t)} \end{aligned}$$

This setting ensures that the time required for the behavior set's motivation to increase from 0 until it exceeds the threshold of activation equals the time of $r_i$'s patience with $r_k$. Since the motivation is reset to 0 when $r_k$ first begins execution of task $h_i(a_{ij})$, but never again, this ensures that $r_i$ does indeed give $r_k$ an opportunity to perform task $h_i(a_{ij})$. However, $r_i$ cannot be fooled by repeated unsuccessful attempts

by $r_k$ to perform task $h_i(a_{ij})$; thus $r_i$ will eventually take over this task if $r_k$ does not demonstrate its ability to accomplish it.

Now let us examine the $\delta\_fast_{ij}(t)$ parameters; these parameters can affect the selection of tasks from either task category 1 or 2, which means they must at times cause tasks to be selected according to the shortest first, and at other times according to the longest first. An additional detail concerning robot idle time between task activations must now be addressed. Any $\delta\_fast_{ij}(t)$ parameter corresponding to a task in the second category could be set the same as $\delta\_slow_{ij}(k,t)$ for some $k$. This would indeed cause the tasks to be selected in ascending order according to the expected task completion time. However, we note that during the time in which the $\delta\_fast_{ij}(t)$ parameters are below the threshold $\theta$, the robot is idle. Thus, setting a $\delta\_fast_{ij}(t)$ parameter the same as its corresponding $\delta\_slow_{ij}(k,t)$ parameter would cause the robot to wait for a period of time $\phi_{ij}(k,t)$ before activating task $h_i(a_{ij})$, which in turn means that the robot would remain idle about as long as it spends performing tasks. This is clearly unacceptable for the sake of efficiency; thus, the $\delta\_fast_{ij}(t)$ parameter must be scaled such that robot idle time is reduced while maintaining the relative impatience rates across motivational behaviors.

One easy way of scaling the $\delta\_fast_{ij}(t)$ parameters is to multiply them by some constant greater than 1. However, while this approach does reduce the idle time and maintain the relative ordering among the tasks, it still does not place an upper bound on how long a robot might remain idle during its mission. A better way of scaling the idle times is to map them to some acceptable range based upon expected task completion time. To do this, we define the notion of a *minimum allowable delay* and a *maximum allowable delay*, which give the range of times a robot can remain idle while waiting on its next behavior set to be activated. The actual values for these allowable delays should be set by the human designer according to the application. The only restriction is that the minimum delay should be greater than 0. Then, the ideal method of scaling the rates to within this range requires the motivational behaviors to ascertain the global minimum and maximum expected task completion times across all tasks of the mission. The reason why the global minimum and maximum times are ideal is because this allows the rates of impatience for a given task to remain calibrated across robots. However, unless outer boundaries for these values are provided by the human designer in advance, this requirement violates the distributed nature of L-ALLIANCE across robots. Although it would be possible to provide the robots with the ability to determine these global minimum and maximum task completion times through the broadcast communication system, we decided not to violate the distributed nature of L-ALLIANCE across robots. Instead, we approximate these global minimum and maximum task completion times with the minimum and maximum task completion times known within a given robot. Although this, too, violates the purely distributed nature of L-ALLIANCE within an individual robot, it can easily be accomplished through message passing between the motivational behaviors or a shared memory location[5]. With these new values, then, the proper settings of the $\delta\_fast_{ij}(t)$ parameters are determined as follows:
Let:

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

Then:

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

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

Thus, in the case of category 2 tasks, the fast impatience rates grow more quickly for the shorter tasks, whereas category 1 task impatience rates grow more quickly for longer tasks. In either case, the maximum delay before task activation is *max_delay*.

The specification of when the impatience rate for a behavior set $a_{ij}$ should grow according to the slow impatience rate and when it should grow according to the fast impatience rate is given by the following function:

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

Thus, the impatience rate will be the minimum slow rate, $\delta\_slow_{ij}(k,t)$, if robot $r_i$ has received communication indicating that robot $r_k$ is performing the task $h_i(a_{ij})$ in the last $\tau_i$ time units, but not for longer than $\phi_{ij}(k,t)$ time units. Otherwise, the impatience rate is set to $\delta\_fast_{ij}(t)$.

The final detail to be addressed is to cause a robot's motivation to activate behavior set $a_{ij}$ to go to 0 the first time it hears about another robot performing task $h_i(a_{ij})$. This is accomplished through the following:

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

This reset function causes the motivation to be reset to 0 if robot $r_i$ has just received its first message from robot $r_k$ indicating that $r_k$ is performing task $h_i(a_{ij})$. This function allows the motivation to be reset no more than once for every robot team member that attempts task $h_i(a_{ij})$. Allowing the motivation to be reset repeatedly by the same robot would allow a persistent, yet failing robot to jeopardize the completion of the mission.

### *6.7. Robot acquiescence*

Two parameters are used to implement the robot acquiescence characteristic of ALLIANCE: $\psi_{ij}(t)$ and $\lambda_{ij}(t)$.

$\psi_{ij}(t)$ = 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

The first parameter, $\psi_{ij}(t)$, is updated according to the current impatience/acquiescence parameter update strategy, as follows:

- For mildly heterogeneous teams in which the *Progress When Working* condition does not hold, $\psi_{ij}(t)$ should be set to $task\_time_i(i,j,t)$ (i.e. the time $r_i$ expects to need to complete task $h_i(a_{ij})$; this is strategy III).

- Otherwise, $\psi_{ij}(t)$ should be set to $\min_{r_k \in robots\_present(i,t)} task\_time_i(k,j,t)$ (i.e. the minimum time $r_i$ expects any robot would need to perform task $h_i(a_{ij})$; this is strategy II).

The second robot acquiescence parameter, $\lambda_{ij}(t)$, gives the time robot $r_i$ wants to maintain behavior set $a_{ij}$ activation before giving up to possibly try another behavior set. The value of the $\lambda_{ij}(t)$ parameter is based upon the time robot $r_i$ expects it requires to perform task $h_i(a_{ij})$. This parameter should be conservatively set, however, so that mild underestimates of expected task time do not cause a robot to give up prematurely. Values for $\lambda_{ij}(t)$ set at two or three times the expected task completion time work well in practice.

The following *acquiescence* function indicates when a robot has decided to acquiesce its task:

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

This function says that a robot $r_i$ will not acquiesce behavior set $a_{ij}$ until one of the following conditions is met:

- $r_i$ has worked on task $h_i(a_{ij})$ for a length of time $\psi_{ij}(t)$ and some other robot has taken over task $h_i(a_{ij})$
- $r_i$ has worked on task $h_i(a_{ij})$ for a length of time $\lambda_{ij}(t)$

### *6.8. Motivation calculation*

All of these inputs can now be combined into a simple motivational behavior calculation. During the active learning phase, 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* goes to 0), some other behavior set activates first (*activity_suppression* is 0), or some other robot has taken on that task (*learning_impatience* equals 0).

DURING ADAPTIVE PHASE:

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:

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

Robot $r_i$'s motivation to perform any given task during the adaptive phase thus increments at the proper impatience rate (based upon the activities of other robots) until it crosses the threshold, unless the task becomes complete (*sensory_feedback*), some other behavior set activates first (*activity_suppression*), some other robot has taken over that task (*impatience_reset*), the robot decides to acquiesce the task (*acquiescence*), or some other robot is present that should be able to accomplish the task better than $r_i$ (*learned_robot_influence*).

In either the active or the adaptive learning phases, when behavior set $a_{ij}$ is operational in robot $r_i$, the corresponding motivational behavior broadcasts $r_i$'s current activity to its teammates at a rate of $\rho_i$.

## 7. Robot Experimental Results and Discussion

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 on physical robots include a "mock" hazardous waste cleanup task [28], [31], cooperative manipulation [32], cooperative observation of multiple moving targets [35], and a cooperative box pushing task, which is described in this section. Over 50 logged physical robot runs of the hazardous waste cleanup and the cooperative manipulation tasks were completed, as well as hundreds of runs of the cooperative observation of multiple moving targets. In addition, over 30 physical robot runs of the box pushing task were completed to elucidate the important issues in heterogeneous robot cooperation. Many runs of each of these physical robot applications are available on videotape. The applications using simulated mobile robots include a janitorial service mission and a bounding overwatch mission (reminiscent of military surveillance), which involved dozens of runs each. Details of these implementations are reported in [28], [29], [31].

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., [11], [42], [20], [25], [37]), 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 [33], we reported on the basic box pushing experiment from the ALLIANCE perspective. In this section, we provide extended results using the L-ALLIANCE mechanisms, showing the L-ALLIANCE parameter settings and learned quality measurements, as well as typical motivational traces of the experimental runs.

### *7.1. Physical Robot Team*

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*. All of these robots were designed and manufactured by IS Robotics, Inc. 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 [17] was used in our physical robot implementations to allow the robots to communicate their current actions to each other. This system consisted of a radio modem attached to each robot, plus a base station that is responsible for preventing message interference by time-slicing the radio channel among robots. The design of the radio system limits the frequency of messages between robots to only one message every three seconds. All of the results described in this article, therefore, involve communication between robots at no more than about $\frac{1}{3}$ Hertz.

### *7.2. 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. We therefore describe the L-ALLIANCE box pushing software of these robots separately.

#### *R-2 Control*

Figure 3 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 3 by the *pushed-at-right* arrow entering the *push_left* motivational behavior. 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. The pushing is initiated at the beginning of the application by programming the control code so that each robot believes that the opposite end of the box has just been pushed.

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. At the beginning of the application, we would ideally like the R-2 to be able to locate one end of the box on its own. However, since this is beyond the scope of these proof of concept experiments, an implicit assumption is made in the R-2 control that at the beginning of the task, the R-2 is facing into a known end of the box.

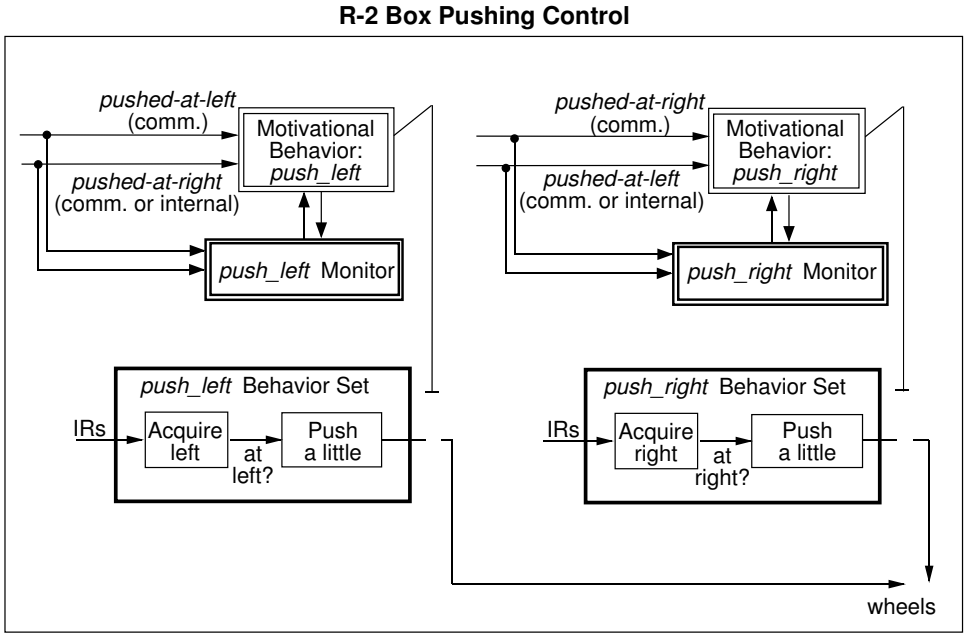


*Fig. 3.* The L-ALLIANCE design of the R-2 software for the box pushing application.

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, 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 4 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 4 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

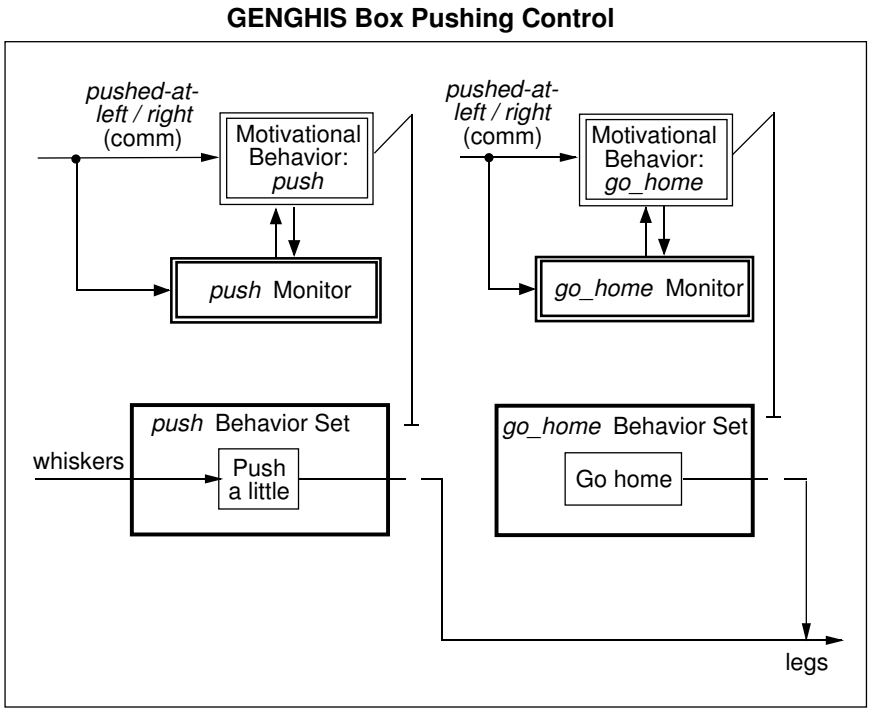


*Fig. 4.* The L-ALLIANCE design of the *GENGHIS* software for the box pushing application.

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.

### *7.3. Initial Learning of Parameter Settings*

We began the experiments by allowing the robots to perform their box pushing tasks while teamed with one of the other available robots during the L-ALLIANCE active learning phase. During this period, the robots monitored the performance of their teammates in performing tasks of interest to them. This learning time was quite simple, only requiring robots to note the task completion times. The L-ALLIANCE performance monitors then instantiated the various parameters of the system according to the learned task times. Once the active learning phase was complete (5 trials), the robots were ready to perform their tasks in the adaptive learning phase, as described in the next subsections.

We now summarize the parameters of these experiments. First, the definition of the task team is:

$$\begin{aligned} R &= \{GREEN, BLUE, GENGHIS\} \\ T &= \{push\_left, push\_right, go\_home\} \\ A_{BLUE} = A_{GREEN} &= \{push\_left, push\_right\} \end{aligned}$$

| $i$ | $k$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|---|
| | | $task\_time_i(k, j, t)$ | | | |
| *BLUE* | *BLUE* | $13 + 2$ | $13 + 2$ | | |
| *BLUE* | *GREEN* | $13 + 2$ | $13 + 2$ | | |
| *BLUE* | *GENGHIS* | $0 + 30$ | $0 + 30$ | | |
| *GREEN* | *BLUE* | $13 + 2$ | $13 + 2$ | | |
| *GREEN* | *GREEN* | $13 + 2$ | $13 + 2$ | | |
| *GREEN* | *GENGHIS* | $0 + 30$ | $0 + 30$ | | |
| *GENGHIS* | *BLUE* | | | $\infty$ | $\infty$ |
| *GENGHIS* | *GREEN* | | | $\infty$ | $\infty$ |
| *GENGHIS* | *GENGHIS* | | | $0 + 30$ | 17 |

*Table 2.* The learned values for $task\_time_i(k, j, t)$.

$$
\begin{aligned}
A_{GENGHIS} &= \{push, go\_home\} \\
h_{GREEN}(push\_left) &= push\_left \\
h_{GREEN}(push\_right) &= push\_right \\
h_{BLUE}(push\_left) &= push\_left \\
h_{BLUE}(push\_right) &= push\_right \\
h_{GENGHIS}(push) &= \begin{cases} push\_left & \text{if } GENGHIS \text{ is on left side} \\ push\_right & \text{if } GENGHIS \text{ is on right side} \end{cases} \\
h_{GENGHIS}(go\_home) &= go\_home \\
\theta &= 1 \\
\rho_{GREEN} = \rho_{BLUE} = \rho_{GENGHIS} &= 1/3 \\
\tau_{GREEN} = \tau_{BLUE} = \tau_{GENGHIS} &= 5 \\
min\_delay &= 0.1 \\
max\_delay &= 1
\end{aligned}
$$

Table 2 gives the values learned for $task\_time_i(k, j, t)$. For the two R-2 robots, the task times are stored as two components – one for acquiring the opposite end of the box, and one for pushing the box. From these task times, the parameters corresponding to robot impatience and robot acquiescence are derived, as shown in Tables 3 through 7.

### *7.4. Experiments and Results*

To demonstrate the fault tolerant, adaptive nature of the L-ALLIANCE architecture due to changes in the robot team capabilities, 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 5.

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 back into the team. In the second experiment, we again seized one of the R-2 robots (*GREEN*), but this time we replaced it

| $i$ | $k$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|---|
| | | $\phi_{ij}(k,t) = task\_time_i(k,j,t)$ | | | |
| *BLUE* | *GREEN* | 15 | 15 | | |
| *BLUE* | *GENGHIS* | 30 | 30 | | |
| *GREEN* | *BLUE* | 15 | 15 | | |
| *GREEN* | *GENGHIS* | 30 | 30 | | |
| *GENGHIS* | *BLUE* | | | $\infty$ | $\infty$ |
| *GENGHIS* | *GREEN* | | | $\infty$ | $\infty$ |

*Table 3.* The learned values for $\phi_{ij}(k,t)$.

| $i$ | $k$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|---|
| | | $\delta\_slow_{ij}(k,t) = \theta/\phi_{ij}(k,t)$ | | | |
| *BLUE* | *GREEN* | $1/15 = 0.07$ | $1/15 = 0.07$ | | |
| *BLUE* | *GENGHIS* | $1/30 = 0.03$ | $1/30 = 0.03$ | | |
| *GREEN* | *BLUE* | $1/15 = 0.07$ | $1/15 = 0.07$ | | |
| *GREEN* | *GENGHIS* | $1/30 = 0.03$ | $1/30 = 0.03$ | | |
| *GENGHIS* | *BLUE* | | | $1/\infty = 0$ | $1/\infty = 0$ |
| *GENGHIS* | *GREEN* | | | $1/\infty = 0$ | $1/\infty = 0$ |

*Table 4.* The learned values for $\delta\_slow_{ij}(k,t)$.

| $i$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|
| | $\delta\_fast_{ij}(t) = \frac{\theta}{max\_delay-(task\_time_i(i,j,t)-low)\times scale\_factor}$ | | | |
| *BLUE* | 1 | 1 | | |
| *GREEN* | 1 | 1 | | |
| *GENGHIS* | | | 1 | 1 |

*Table 5.* The learned values for $\delta\_fast_{ij}(t)$.

| $i$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|
| | $\psi_{ij}(t) = task\_time_i(i,j,t)$ | | | |
| *BLUE* | 15 | 15 | | |
| *GREEN* | 15 | 15 | | |
| *GENGHIS* | | | 30 | 17 |

*Table 6.* The learned values for $\psi_{ij}(t)$.

| $i$ | $a_{ij} = push\_left$ | $a_{ij} = push\_right$ | $a_{ij} = push$ | $a_{ij} = go\_home$ |
|---|---|---|---|---|
| | $\lambda_{ij}(t) = 3 * task\_time_i(i,j,t)$ | | | |
| *BLUE* | 45 | 45 | | |
| *GREEN* | 45 | 45 | | |
| *GENGHIS* | | | 90 | 51 |

*Table 7.* The learned values for $\lambda_{ij}(t)$.

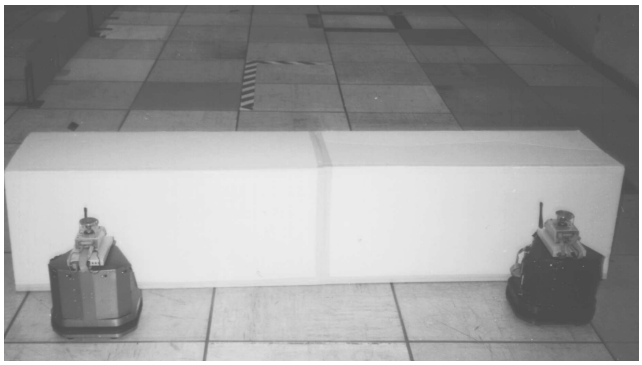

*Fig. 5.* The beginning of the box pushing application. Two R-2s are pushing the box across the room. (The R-2 robot *GREEN* is on the left, the R-2 robot *BLUE* is on the right.)

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, it should be clear that the L-ALLIANCE approach can continually adapt the actions of robots over time to achieve the goal of lifelong implementations of heterogeneous multi-robot teams.

**Experiment 1: Robot "failure".** As we have emphasized, a primary goal of our architecture is to allow robots to dynamically reselect their actions in response to changes in their situation, such as failures of robot team members. Thus, 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 6, 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 7 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 $\theta$.

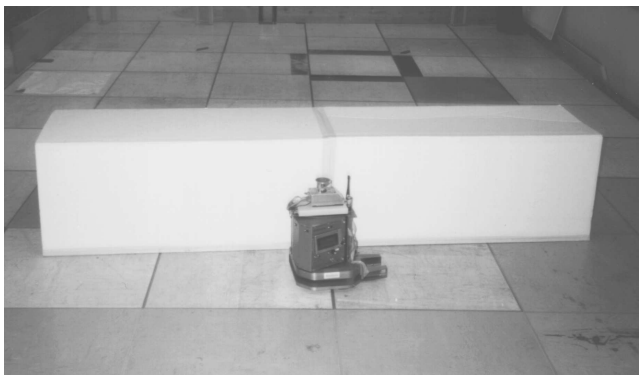

*Fig. 6.* Fault tolerant action selection. In the first experiment, we seize *GREEN* and turn it off. This causes *BLUE* to have to perform both tasks of the box pushing application: pushing at the right end of the box, and pushing at the left end of the box. Here, *BLUE* is acquiring the right end of the box.

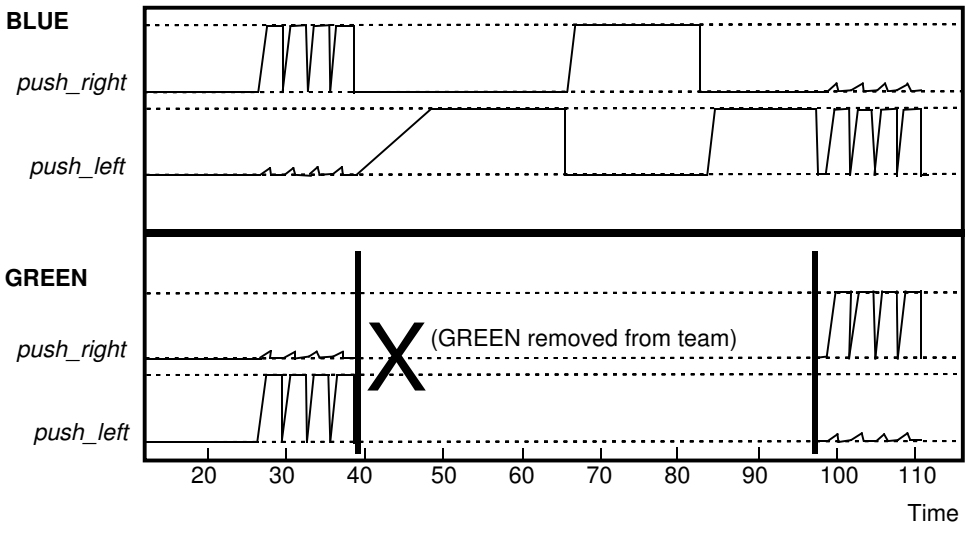


*Fig. 7.* Motivational levels for behavior sets during experiment with two R-2 robots. During the middle of the experiment, we removed *GREEN*, during which time *BLUE* had to push at both sides of the box. We later added *GREEN* back into the application, which resulted in *BLUE* adapting again to only push at one end of the box.

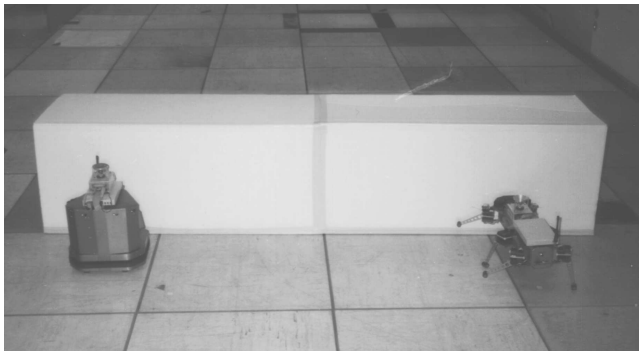

*Fig. 8.* Adaptivity due to heterogeneity. In the second experiment, we again seize one of the R-2 robots, but this time we replace it with *GENGHIS*. Since *GENGHIS* cannot push as powerfully as an R-2, the remaining R-2 robot adapts its actions by pushing less frequently.

**Experiment 2: Increased heterogeneity.** Another goal of our architecture is to allow heterogeneous robot teams to work together efficiently. As we have noted earlier, robots can be heterogeneous in two obvious ways. First, robots may differ in which tasks they are able to accomplish, and second, robots may differ in how well they perform the same task. In this experiment, we deal primarily with the second type of heterogeneity, in which *GENGHIS* and the R-2 robots use different mechanisms for pushing the box. 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 8 illustrates *BLUE* and *GENGHIS* pushing together. Figure 9 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 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.

### *7.5. Discussion*

The types of situations that L-ALLIANCE is designed to handle over the lifetime of a mission include:

- New team composition

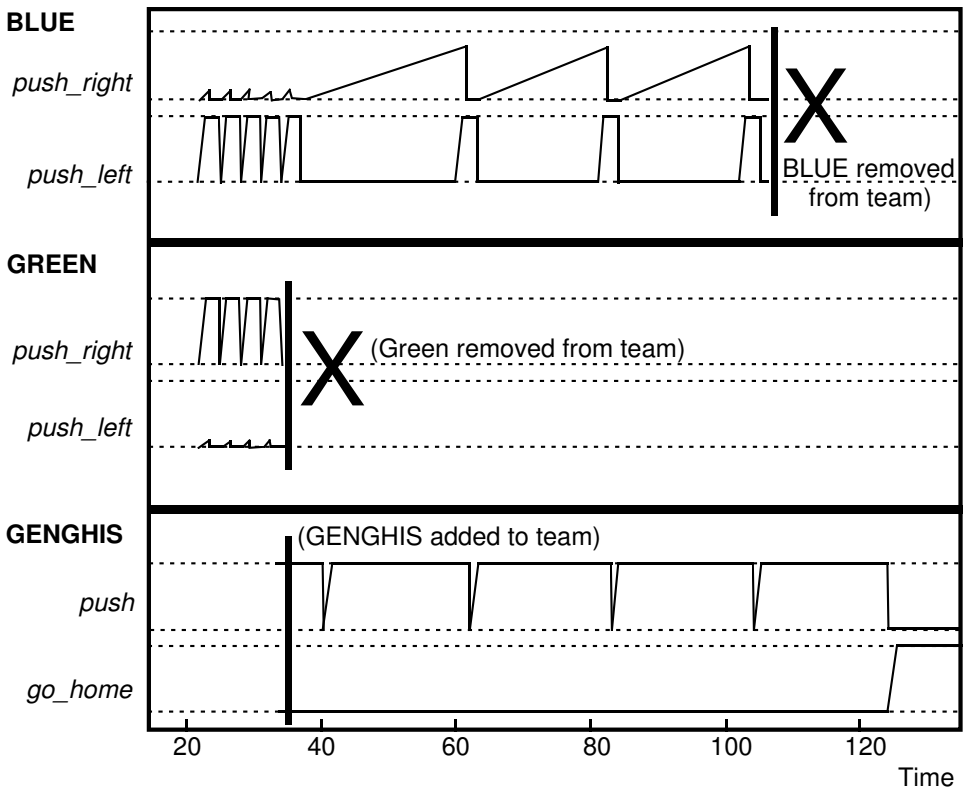


*Fig. 9.* The trace of the activation levels of the behavior sets during the second experiment.

- Improved individual robot capabilities
- Degraded individual robot capabilities
- New individual robot capabilities
- Environmental and mission changes

The experiments reported in this section illustrate how the L-ALLIANCE architecture allows robots to learn about their teammates' capabilities and to adapt their action selections during a mission in response to varying robot capabilities and team composition. While we do not report additional experiments here, it should be clear that the L-ALLIANCE mechanism allows the robot team to adapt to a variety of situations that occur over time – with the caveat that the underlying assumptions of L-ALLIANCE are true in the given application. (See section 9 for a summary of the limitations of L-ALLIANCE.)

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 $\mu$ – 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 $\mu$), 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.

## 8. Related Work

The amount of research in the field of cooperative mobile robotics has grown substantially in recent years. This work can be broadly categorized into two groups: swarm-type cooperation and "intentional" cooperation. A number of researchers have studied the issues of swarm robotics. Deneubourg *et al.* [10] describe simulation results of a distributed sorting algorithm. Theraulaz *et al.* [45] extract cooperative control strategies, such as foraging, from a study of Polistes wasp colonies. Steels [41] presents simulation studies of the use of several dynamical systems to achieve emergent functionality as applied to the problem of collecting rock samples on a distant planet. Drogoul and Ferber [12] describe simulation studies of foraging and chain-making robots. In [24] Mataric describes the results of implementing group behaviors such as dispersion, aggregation, and flocking on a group of physical robots. Beni and Wang [4] describe methods of generating arbitrary patterns in cyclic cellular robotics. Kube and Zhang [20] present the results of implementing an emergent control strategy on a group of five physical robots performing the task of locating and pushing a brightly lit box. Stilwell and Bay [42] present a method for controlling a swarm of robots using local force sensors to solve the problem of the collective transport of a palletized load. Arkin *et al.* [1] present research concerned with sensing, communication, and social organization for tasks such as foraging. The CEBOT work, described in [14] and many related papers, has many similar goals to other swarm-type multi-robotic systems; however, the CEBOT robots can be one of a number of robot classes, rather than purely homogeneous.

The primary difference between these approaches and the problem addressed in this article is that the above approaches are designed strictly for homogeneous robot teams, in which each robot has the same capabilities and control algorithm. Additionally, issues of efficiency are largely ignored. However, in heterogeneous robot teams such as those addressed in this article, not all tasks can be performed by all team members, and even if more than one robot can perform a given task, they may perform that task quite differently. Thus the proper mapping of subtasks to robots is dependent upon the capabilities

and performance of each robot team member. This additional constraint brings many complications to a workable architecture for robot cooperation, and must be addressed explicitly to achieve the desirable level of cooperation.

The second primary area of research in cooperative control deals with achieving "intentional" cooperation among a limited number of typically heterogeneous robots performing several distinct tasks. In this type of cooperative system, the robots often have to deal with some sort of efficiency constraint that requires a more directed type of cooperation than is found in the swarm approach described above. Furthermore, this second type of mobile robotic mission usually requires that several distinct tasks be performed. These missions thus usually require a smaller number of possibly heterogeneous mobile robots involved in more purposeful cooperation. Although individual robots in this approach are typically able to perform some useful task on their own, groups of such robots are often able to accomplish missions that no individual robot can accomplish on its own. Key issues in these systems include robustly determining which robot should perform which task so as to maximize the efficiency of the team and ensuring the proper coordination among team members to allow them to successfully complete their mission.

Most of the existing work on heterogeneous physical robots uses a traditional artificial intelligence approach, which breaks the robot controller into modules for sensing, world modeling, planning, and acting (hence, the *sense-model-plan-act* paradigm), rather than the functional decomposition of behavior-based approaches. Noreils [26] describes one such *sense-model-plan-act* control architecture which includes three layers of control: the planner level, which manages coordinated protocols, decomposes tasks into smaller subunits, and assigns the subtasks to a network of robots; the control level, which organizes and executes a robot's tasks; and the functional level, which provides controlled reactivity. He reports on the implementation of this architecture on two physical mobile robots performing convoying and box pushing. In both of these examples, one of the robots acts as a leader, and the other acts as a follower.

Caloud *et al.* [6] describe another *sense-model-plan-act* architecture which includes a task planner, a task allocator, a motion planner, and an execution monitor. Each robot obtains goals to achieve either based on its own current situation, or via a request by another team member. They use Petri Nets for interpretation of the plan decomposition and execution monitoring.

In [2] and elsewhere, Asama *et al.* describe their decentralized robot system called ACTRESS, addressing the issues of communication, task assignment, and path planning among heterogeneous robotic agents. Their approach revolves primarily around a negotiation framework which allows robots to recruit help when needed. They have demonstrated their architecture on mobile robots performing a box pushing task.

Wang [46] addresses a similar issue to that addressed in here — namely, dynamic, distributed task allocation when more than one robot can perform a given task. He proposes the use of several distributed mutual exclusion algorithms that use a "sign-board" for inter-robot communication. These algorithms are used to solve problems including distributed leader finding, the N-way intersection problem, and robot ordering. However, their research does not address issues of dynamic reallocation due to robot failure and efficiency issues due to robot heterogeneity.

Cohen *et al.* [8] propose a hierarchical subdivision of authority to address the problem of cooperative fire-fighting. They describe their Phoenix system, which includes a generic simulation environment and a real-time, adaptive planner. The main controller in this architecture is called the Fireboss, which maintains a global view of the environment, forms global plans, and sends instructions to agents to activate their own local planning.

Ohko *et al.* [27] describe a learning system, called LEMMING, which learns knowledge quite similar to that learned in L-ALLIANCE. In their system, however, this knowledge is used by a case-based reasoner for reducing the communication flow between distributed agents. These distributed agents use the Contract Net Protocol [38] to negotiate the allocation of tasks. With LEMMING, the agents can often use point-to-point communication rather than broadcast communication to recruit help directly from those agents known to have the capabilities to perform a given task, thus reducing the overall

communication traffic. They present results from a simulation application involving the movement of objects from one location to another by a team of distributed agents.

Much theoretical work has been accomplished for intentional agent control by the Distributed Artificial Intelligence (DAI) community ([5] contains many examples). In most of this work, the issue of task allocation has been the driving influence that dictates the design of the architecture for cooperation, since the selected approach to task allocation invariably restricts the potential solutions to other issues of cooperation, such as conflict resolution.

Typically, the DAI approaches use a distributed, negotiation-based mechanism to determine the allocation of tasks to agents, using a variety of proposed protocols (e.g., [9], [39], [13], [19], [36], [49]). Under these negotiation schemes, no centralized agent has full control over which tasks individual team members should perform. Instead, many agents know which subtasks are required for various portions of the mission to be performed, along with the skills required to achieve those subtasks. These agents then broadcast a request for bids to perform these subtasks, which other agents may respond to if they are available and want to perform these tasks. The broadcasting agent then selects an agent from those that respond and awards the task to the winning agent, who then goes on to perform that task, recruiting yet other agents to help if required.

However, although DAI work has demonstrated success in a number of domains (e.g. distributed vehicle monitoring [21] and distributed air traffic control [7]), the proposed solutions have rarely been demonstrated as directly applicable to *situated* agent (i.e. robotic) teams, which have to live in, and react to, a dynamic and uncertain environment using noisy sensors and effectors, and a limited bandwidth, noisy communication mechanism. They typically rely on unrealistic "black boxes" to provide high-level, perfect sensing and action capabilities. Furthermore, as with the approaches of the previous subsection, these DAI approaches typically ignore or only give brief treatment to the issues of robot performance of those tasks after they have been allocated. Such approaches usually assume the robots will eventually accomplish the task they have been assigned, or that some external monitor will provide information to the robots on dynamic changes in the environment or in robot performance. However, to realistically design a cooperative approach to robotics, we must include mechanisms within the software control of each robot that allow the team members to recover from dynamic changes in their environment or in the robot team.

Researchers have recognized that an approach with more potential for the development of cooperative control mechanisms is autonomous learning. Hence, much current work is ongoing in the field of multi-agent learning (e.g., [48]). The types of applications that are typically studied vary considerably in their characteristics. Some of the applications include air fleet control [40], predator/prey [3], [18], [16], [44], and multi-robot soccer [43], [23].

In section 7, we mentioned previous works that have investigated various issues using the box-pushing application domain, which include [11], [42], [20], [25], [37].

In [47], Weiss describes a system with similar goals to the learning described in this article — namely, how agents can learn to coordinate their actions. In his work, Weiss makes a number of assumptions that are also made in our work: (1) each agent has limited knowledge about its environment, (2) cooperation is required to solve tasks, (3) agent actions can conflict with each other, (4) agent skills and knowledge differ, and (5) agent behavior is tightly coupled with the environment. Weiss' approach involves the use of two algorithms for collective learning, called ACE and AGE (these acronyms stand for "ACtion Estimation" and "Action Group Estimation"). In the ACE algorithm, the multi-agent system learning results from the repeated execution of three steps, which he calls *action determination*, *competition*, and *credit assignment*. However, the application domain that Weiss addresses is somewhat different from the domain addressed here; whereas Weiss' work is addressed at learning sequences of applicable actions that accomplish a given mission, we are interested in having agents select actions that allow them to robustly accomplish a set of independent subtasks in a minimal amount of time.

Bing Liu [22] describes a heuristic, centralized method for coming up with a good job shop schedule. The centralized scheduler has complete information on the tasks to be done and the capabilities of the

"agents" (which in this case are machines). The system strives to deal with many constraints imposed by the organization, by precedence requirements, by resource requirements, or by preferences. It is assumed that the scheduler will always be operational and is not itself an agent, and that it fully knows the state of each agent and the state of the tasks to be performed. Sensory and effector uncertainty is not an issue. This type of system is not designed for real mobile robotic systems, since sensing and action are quite noisy, and since no single agent can be expected to know the full state of the environment (which includes the task states and the agent states) at all times.

## 9. Conclusions: Summary of Advantages and Limitations

The dream of many robotics researchers is to generate robotic systems that can perform real-world missions over long periods of time, even while the environment or the robotic system itself changes. One important component of these robotic systems is a control strategy that enables the robots to adapt their actions throughout their lifetime, without the requirement of human intervention. The L-ALLIANCE architecture we have presented in this article offers an approach to achieving lifelong dynamic action selection in heterogeneous mobile robot teams. This architecture is a behavior-based, distributed control approach that is built upon our earlier ALLIANCE architecture. The L-ALLIANCE architecture extends ALLIANCE by enabling robot teams to automatically update their control parameters during their mission as they learn more about the capabilities of their teammates. The parameter updates are based upon robot measurements of the quality of task performance of their teammates in performing common tasks. We described the method used in L-ALLIANCE to achieve automatic adaptation over time. We then described in detail the results of the implementation of the architecture in a team of heterogeneous mobile robots performing a box pushing application.

While the primary purpose of the L-ALLIANCE mechanism as presented in this article is to enable the lifelong application of multi-robot teams, a number of other benefits result from providing robots with the ability to automatically adjust their own parameter settings to improve efficiency. A summary of the advantages of the L-ALLIANCE mechanism is as follows:

1. **Relieve humans of the parameter adjusting task:**
   ALLIANCE alone requires human programmer tuning of motivational behavior parameters to achieve desired levels of robot performance. Although finding good parameter settings is often not difficult in practice, the cooperative architecture is much simpler to use when the human is relieved of the responsibility of having to tune numerous parameters.
2. **Improve the efficiency of the mission performance:**
   Related to the previous item is the issue of the efficiency of the robot team's performance of its mission. As human designers, it is often difficult to evaluate a given robot team performance to determine how best to adjust parameters to improve efficiency. However, if the robots were controlled by an automated action selection strategy that has been shown to result in efficient group action selection in practice, then the human designer can have confidence in the robot team's ability to accomplish the mission autonomously, and thus not feel the need to adjust the parameters by hand.
3. **Facilitate custom-designed robot teams:**
   Providing the ability for robot teams to carry over their learned experiences from trial to trial allows human designers to successfully construct unique teams of interacting robots from a pool of heterogeneous robot types for any given mission without the need for a great deal of preparatory work. Although ALLIANCE allows newly constructed teams to work together acceptably the first time they are grouped together, the automated parameter adjusting mechanisms of L-ALLIANCE allow the team to improve its performance over time by having each robot learn how the presence of other specific robots on the team should affect its own behavior.
4. **Enable lifelong heterogeneous multi-robot teams:**
   During a mission, a robot team's environment and the ability of its members may change dynamically. However, in the basic ALLIANCE architecture, parameter settings do not change after the start

of the mission. Thus, these robot teams would be quite vulnerable to calibration problems due to drifts in the environment and in robot capabilities. The ability to automatically update parameters during a mission, as provided by L-ALLIANCE, is therefore of critical importance for real-world implementations of multi-robot teams.

Of course, the L-ALLIANCE architecture does not address every type of heterogeneous multi-robot application, and has other limitations that are important to note. These limitations, which will be the subjects of future work, are as follows:

1. **Independent subtasks:**
   L-ALLIANCE is limited to heterogeneous multi-robot applications that are composed of independent subtasks. While the domain of multi-robot applications that fit this description is quite large, there are other types of cooperation that require a tighter coupling of robot tasks. One application of this type that we have been studying [35], [34] is the cooperative observation of multiple moving targets. This application requires a team of robots to continually adjust their movements based upon the locations of targets moving through an area of interest, as well as the movements of other robots on the team. We are exploring other approaches to lifelong adaptation in this application domain, so that robots can adjust their control strategy over time based upon the sensory capabilities of their teammates and the current level of success of the robot team.
2. *Time* **Quality metric:**
   The current instantiation of L-ALLIANCE is defined only for the quality metric of *time*. Thus, L-ALLIANCE is not directly useful for applications in which other quality metrics are of interest. In future work, we will examine the modifications that are needed in L-ALLIANCE to deal with other application-specific performance measures. Additionally, it is assumed that: (1) the time of task performance in prior trials is indicative of performance in the future, (2) a sufficient number of trial outcomes is available upon which to base the action selection decisions, and (3) the standard deviation of task performance of any given task is small relative to the average task completion time. If these assumptions are not accurate in a given application, then the L-ALLIANCE approach will not result in an efficient selection of actions.
3. **Number of trials:**
   The quickness of response of a robot team to changes in team member capabilities is dependent upon the value of $\mu$ – the number of trials over which quality measures are maintained. If $\mu$ is small, then the robot team will respond rapidly to changes in team member capabilities. If $\mu$ is large, then the team will exhibit more hysteresis, thus responding more slowly to changes in the system, while exhibiting more stability. In some applications, it may be more appropriate for the team as a whole to undergo a state change under certain conditions, allowing more abrupt responses to system events.

## Acknowledgements

This research was funded in part by the Engineering Research Program, Office of Science, of the U.S. Dept. of Energy. This article has been authored by a contractor of the U.S. government under Contract DE-AC05-96OR22464. 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.

## Notes

1. We note that while the multi-robot missions addressed in this article must consist of tasks that can be executed independently by one robot at a time, an extremely wide variety of applications can be designed to meet these independent task constraints (e.g., see [29], [35], [32] and the application described in this article). Nevertheless, we note that no mechanism for protecting subgoals is provided in L-ALLIANCE, which will be an important consideration for some multi-robot tasks.

2. The *task coverage* is a measure of the total number of capabilities on the robot team that may allow a team member to achieve a given task — see [28] for a further discussion of this measure.
3. In this context, *degree of heterogeneity* refers to a relative comparison of $q(a_{ij})$ and $q(a_{kl})$, in which $h_i(a_{ij}) = h_k(a_{kl})$; in other words, $r_i$ and $r_k$ can both perform the same task, but potentially with different levels of performance.
4. We define the *Progress When Working* condition as follows (see also [28]). Let $z$ be the finite amount of work remaining to complete a task $w$. Then, whenever robot $r_i$ activates a behavior set corresponding to task $w$, either (1) $r_i$ remains active for a sufficient, finite length of time $\epsilon$ such that $z$ is reduced by a finite amount which is at least some constant $\delta$ greater than 0, or (2) $r_i$ experiences a failure with respect to task $w$. Additionally, if $z$ ever increases, the increase is due to an influence external to the robot team.
5. If this, too, is undesirable, then the motivational behaviors can be provided with the expected minimum and maximum task completion times at the beginning of the mission, and then approximate the proper scaling. Small changes in the definition of $\delta\_fast_{ij}(t)$ would then be necessary to ensure that the rates do not drop below zero.

## References


1. Ronald C. Arkin, Tucker Balch, and Elizabeth Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *Proceedings of the 1993 International Conference on Robotics and Automation*, pages 588–594, 1993.
2. H. Asama, K. Ozaki, A. Matsumoto, Y. Ishida, and I. Endo. Development of task assignment system using communication for multiple autonomous robots. *Journal of Robotics and Mechatronics*, 4(2):122–127, 1992.
3. M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources. Technical Report BCS-G2010-28, Boeing AI Center, August 1985.
4. Gerardo Beni and Jing Wang. On cyclic cellular robotic systems. In *Japan – USA Symposium on Flexible Automation*, pages 1077–1083, Kyoto, Japan, 1990.
5. Alan Bond and Less Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.
6. Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape, and Mark Yim. Indoor automation with many mobile robots. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, pages 67–72, Tsuchiura, Japan, 1990.
7. Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. In *Proceedings of 8th International Joint Conference on Artificial Intelligence*, pages 767–770, 1983.
8. Paul Cohen, Michael Greenberg, David Hart, and Adele Howe. Real-time problem solving in the Phoenix environment. COINS Technical Report 90-28, University of Massachusetts at Amherst, 1990.
9. Randall Davis and Reid Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983.
10. J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, and P. Dario. Self-organizing collection and transport of objects in unpredictable environments. In *Japan-U.S.A. Symposium on Flexible Automation*, pages 1093–1098, Kyoto, Japan, 1990.
11. 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.
12. Alexis Drogoul and Jacques Ferber. From Tom Thumb to the Dockers: Some experiments with foraging robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 451–459, Honolulu, Hawaii, 1992.
13. Edmund Durfee and Thomas Montgomery. A hierarchical protocol for coordinating multiagent behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 86–93, 1990.
14. T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organizing robots based on cell structures — CEBOT. In *Proceedings of 1988 IEEE International Workshop on Intelligent Robots and Systems (IROS '88)*, pages 145–150, 1988.
15. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
16. 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.
17. IS Robotics, Inc., Somerville, Massachusetts. *ISR Radio Communication and Positioning System*, October 1993.
18. R. Korf. A simple solution to pursuit games. In *Working Papers of the 11th International Workshop on Distributed Artificial Intelligence*, pages 183–194, 1992.
19. Thomas Kreifelts and Frank von Martial. A negotiation framework for autonomous agents. In *Proceedings of the Second European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds*, pages 169–182, 1990.
20. C. Ronald Kube and Hong Zhang. Collective robotic intelligence. In *Proceedings of the Second International Workshop on Simulation of Adaptive Behavior*, pages 460–468, Honolulu, Hawaii, 1992.
21. Victor R. Lesser and Daniel D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, pages 15–33, Fall 1983.
22. Bing Liu. A reinforcement approach to scheduling. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 580–585, 1988.

23. S. Marsella, J. Adibi, Y. Al-Onaizan, G. Kaminka, I. Muslea, and M. Tambe. On being a teammage: 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. Designing emergent behaviors: From local interactions to collective intelligence. In J. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 432–441, Honolulu, Hawaii, 1992. MIT Press.
25. Maja Mataric, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box pushing. In *Proceedings of IEEE International Confereence on Intelligent Robots and Systems (IROS)*, pages 556–561, 1995.
26. Fabrice R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The International Journal of Robotics Research*, 12(1):79–98, February 1993.
27. Takuya Ohko, Kazuo Hiraki, and Yuichiro Anzai. Lemming: A learning system for multi-robot environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1141–1146, Yokohama, Japan, 1993.
28. 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).
29. L. E. Parker. On the design of behavior-based multi-robot teams. *Journal of Advanced Robotics*, 1996.
30. L. E. Parker. L-ALLIANCE: Task-oriented multi-robot learning in behavior-based systems. *Journal of Advanced Robotics*, 1997.
31. L. E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.
32. 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.
33. L. E. Parker. Adaptive heterogeneous multi-robot teams. *Neurocomputing, special issue of NEURAP '98: Neural Networks and Their Applications*, 28:75–92, 1999.
34. L. E. Parker. A case study for life-long learning and adaptation in cooperative robot teams. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, pages 92–101, 1999.
35. 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.
36. Jeffery Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 91–99, 1985.
37. S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of AAAI-94*, pages 426–431, 1994.
38. Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.
39. Reid G. Smith and Randall Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1):61–70, January 1981.
40. 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.
41. Luc Steels. Cooperation between distributed agents through self-organization. In Yves Demazeau and Jean-Pierre Muller, editors, *Decentralized A.I.* Elsevier Science, 1990.
42. Daniel Stilwell and John 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.
43. P. Stone and M. Veloso. A layered approach to learning client behaviors in the robocup soccer server. *Applied Artificial Intelligence*, 12:165–188, 1998.
44. M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.
45. Guy Theraulaz, Simon Goss, Jacques Gervet, and Jean-Louis Deneubourg. Task differentiation in Polistes wasp colonies: a model for self-organizing groups of robots. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 346–355, Paris, France, 1990.
46. Jing Wang. DRS operating primitives based on distributed mutual exclusion. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1085–1090, Yokohama, Japan, 1993.
47. Gerhard Weiss. Collective learning and action coordination. Technical Report FKI-1662-92 (revised), Technische Universität München, October 1992.
48. Gerhard Weiss and Sandip Sen, editors. *Adaption and Learning in Multi-Agent Systems.* Springer, 1996.
49. Gilad Zlotkin and Jeffery Rosenschein. Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the Eighth National Conference on AI*, pages 100–105, 1990.