Learning to Role-Switch in Multi-Robot Systems

Eric Martinson and Ronald C. Arkin

Mobile Robot Laboratory
Georgia Institute of Technology
College of Computing
Atlanta, GA 30332-0280
{ebeowulf,arkin}@cc.gatech.edu

Abstract

We present an approach that uses Q-learning on individual robotic agents, for coordinating a mission-tasked team of robots in a complex scenario. To reduce the size of the state space, actions are grouped into sets of related behaviors called roles and represented as behavioral assemblages. A role is a Finite State Automata such as Forager, where the behaviors and their sequencing for finding objects, collecting them, and returning them are already encoded and do not have to be relearned. Each robot starts out with the same set of possible roles to play, te same perceptual hardware for coordination, and no contact other than perception regarding other members of the team. Over the course of training, a team of Q-learning robots will converge to solutions that best the performance of a well-designed handcrafted homogeneous team.

Index terms: Multi-Robot Systems, Role-Switching, Q-learning.

1. Introduction

When a team of people is selected to perform a complex task, they usually start by breaking the task up into smaller pieces, and assigning jobs, or roles, to each team member. The reason for using roles is that, while each person on the team may be capable of handling any one of the designated roles, when combined, the task is too difficult for any one person. By selecting a group of people who are able to perform any of the roles, the performance of the team can be maximized for a scenario. If one of the roles proves to be useless at a given time, a team member can switch to another role to speed up the completion time of the entire team's mission.

The same concepts can be applied to a team of homogeneous robots. As with people, the use of a homogeneous team allows robots to switch between roles with no penalty. Robots can switch roles to fill in for critical positions as needed, and switch out of roles that are not being used at that time. The challenge from a robotics perspective is to determine when role switching is advantageous to the team, versus remaining in their current roles. In this paper, the use of Q-learning as a role-switching mechanism in a foraging task is studied. Each robot in the team, with no inter-robot communication, learns when to switch, and what role to switch to, given the perceptual state of the world.

There are several goals to this research. First, we hope to demonstrate that the complexity problems usually associated with Q-learning [17] in complex scenarios can be overcome by using role-switching. Furthermore, despite the apparent oversimplification of the action space that this entails, robot teams using the Q-learning algorithm are shown to demonstrate an advantage over hand-crafted methods, at least for the scenarios studied. The second goal is to explore the dimensions of this multi-agent domain. By using independent Q-learning functions, it can be evaluated how important the reward function is versus the team size versus the environmental complexity.

This research is part of the ongoing DARPA Mobile Autonomous Robot Software (MARS) program. The overall project focuses on multi-level learning in hybrid deliberative/reactive architectures. Other related papers from our laboratory relevant to this effort include [1,8,9,13].

2. Related Work

Reinforcement Learning [16,17], as used today for coordination in behavior-based robotics, has appeared in a variety of tasks aimed at taking known competencies and building more complex behavior or improved performance. Asada et al [2] demonstrated how reinforcement learning could learn primitive behaviors starting from sensory information. Their soccer robot started with little capability and used visual input to learn how to shoot a soccer ball at a goal. Instead of starting from scratch, work by Mahadevan and Connell [11] exploited the success of already developed primitive behaviors to learn a task. Their robot used Q-learning to learn how to push boxes around a room without getting stuck. Martinson et al [13], worked with even higher levels of abstraction, to coordinate high-level behavioral assemblages in their robots to learn finite state automata in an intercept scenario.

By continuing to increase the level of abstraction, potentially an entire finite state automata (FSA) could be coordinated using a reinforcement learning mechanism. Diettrich [5], in his work on Hierarchical Reinforcement Learning with MAXQ, alludes to the possibility, as does the work by Martinson et al [13]. In this paper, each Q-learner is coordinating a set of roles as described by an

FSA, to learn complex tasks with many requisite subtasks.

Work by Balch [2] in his Ph.D. thesis, combined a number of these elements within a system to learn roles. His multi-agent homogeneous teams were used to study reward functions that resulted in the most diverse behaviors, or degree of specialization, among agents. These learned specializations are analogous to roles as defined in this work.

The concept of roles, and role switching, however, is not limited to the machine learning domain. In particular, robotic soccer has been a popular test bed for role playing robots [4,12]. Robots are crafted to play forward, defender, goalie etc. as do human soccer teams. Role switching is relatively common, as robots that end up on the wrong side of the field might assume a new role to maximize their performance at their new location. Work by Stone and Veloso [14] in particular, has robots assuming new roles in the team depending upon the global strategy selected. It requires significant communication among the robots to periodically determine which strategy is selected, and which roles need to be assumed.

3. Overview of Q-learning

Probably the most widely used reinforcement learning method for robotic systems is Q-Learning [16]. This is largely due to its algorithmic simplicity and the ease of transitioning from a state value function to an optimal control policy by choosing in every state the action with the highest value. Following Kaelbling's approach [7], at every time step the robot perceives the perceptual state *s*. Based on this information the robot chooses an action *a* and executes it. The utility of this action is communicated to the robot through a scalar reinforcement value *r*. The goal of the robot is to choose actions that, in the long run, maximize the sum of the reinforcement value.

Let *S* be the set of distinct internal states that the robot can be in and let *A* be the set of actions that the robot can take. Let *T(s,a,s')* be the probability of transitioning from state *s* to state *s'* using action *a*. If we are given a world model defined by the transition probabilities and the reward function *R(s,a)* we can compute an optimal deterministic stationary policy using techniques from dynamic programming (e.g. *Value Iteration* or *Policy Iteration[7]*).

It is usually the case, however, that a world model is not known in advance and the robot needs to learn this model and simultaneously construct an optimal policy. Q-learning is an algorithm that does just that. Let *Q(s,a)* be the expected value of the discounted reinforcement of taking action *a* in state *s*. The value of this quantity can be estimated recursively with the following formula:

$$Q^*(s,a) = R(s,a) + \gamma \sum T(s,a,s') \max_{a'} Q^*(s',a')$$

The optimal policy in this case is:

$$pi^* = \arg\max_a Q^*(s,a)$$

In other words, the best policy is, in each state, to take the action with the largest Q-value. Thus the Q-function makes the actions explicit, which allows us to compute them on-line using the following Q-learning update rule:

$$Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max Q(s',a') - Q(s,a)\right)$$

where α is the learning rate, and γ is the discount factor ($0 \leq \gamma < 1$). It can be proven [17] that this formula converges if each action is executed in each state an infinite number of times and α is decayed appropriately. For a more detailed discussion of Q-learning, the reader is referred to [7,17].

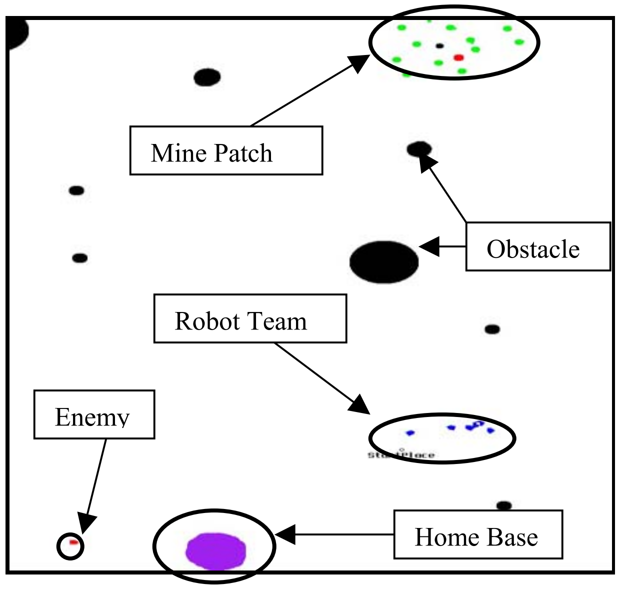

Figure 1. A team of robots in a cluster on the right are trying reach the patch of mines at the top of the screen and return them to the base at the bottom. An enemy robot waits for them on the lower left.

4. Mission Scenario

The scenario chosen is a foraging task in a hostile environment. Anti-Tank mines are scattered about the simulation. A team of robots is expected to collect all of the mines and drop them in a designated storage area. It is assumed that the robots know how to safely handle the explosive ordinance.

However, in addition to the forage task, the robot team is faced with a variety of hazards from the environment. First off, the robots are not perfectly able to navigate within the environment. Unknown terrain can leave robots stuck in shallow locations, or mud pits, preventing them from moving. This is modeled in the simulation as a random occurrence for each robot, where the maximum allowable time between environmental hazards is specified at startup for each map.

The second type of hazard is a mobile enemy hiding in the surrounding environment. If the mobile enemy is not successfully intercepted by one of the team robots, then it will find the nearest robot and "kill" that robot. The "killed" robot becomes stuck in a fixed location until assisted by another robot. When an enemy has "killed" a robot, it retreats to a random location around the edge of the map and waits to attack again. When the robot is either in close proximity (0.1m) to an enemy, or is subject to a random hazard, then it signals the environment that it has died, changes color, and transitions to a STOP action.

All of the programming, and experimental testing described in this paper was performed in the *Missionlab* mission specification [10] environment.

4.1 Q-learning Function

The Q-learning function in *Missionlab* is used on each member of the robot team as a decision-making function for picking the appropriate role (action), based on the perceptual state of the world and the action currently being executed by the Q-learning function. The actions available to each robot are three distinct Finite State Automata that each encapsulate one role: Forager, Soldier, or Mechanic. The perceptual state of the world is represented by a set of four boolean perceptual triggers: Detect_Enemy, Is_Invasion, Detect_Dead, and Is_DieOff. Since a combined state-action pair determines the internal state of the Q-learner, there are 48 possible internal states for the Q-learner. 21 of these 48 internal states are not feasible, because two of the triggers are conditionally dependent on other triggers.

To make decisions, and to learn how successful the robot has been, it is rewarded when either of two events happen. If the robot successfully drops a mine off at the base, then it is rewarded. If the robot successfully aids a dead robot while in the Mechanic role, then it is also rewarded. For most of the experiments, the values of 20 and 10 respectively were used. These reward values were selected for testing after experimenting with different ratios of reward functions.

4.2. Perceptual Triggers

The perceptual triggers used by the Q-learning function all make decisions based on input from a visual sensor. Detect_Enemy and Detect_Dead are represented by the *DETECT_OBJECT* primitive in Missionlab. In simulation, red objects represent Enemies, while yellow objects represent dead or stuck robots. If one of these objects is within 30.0 m of the robot, then the trigger value is TRUE. In the real robot experiments, any object recognized as a dead robot or an enemy, falls well within the 30.0m limit imposed in the simulation.

The remaining two triggers, Is_Invasion, and Is_DieOff, also use different visual inputs. Is_Invasion takes red enemy objects as stimuli, while Is_DieOff reacts to yellow (dead) robots. The purpose of this routine is to trigger TRUE when either an object is very close to the robot, or when there are several such objects within range of detection. The impact of an object on the state of the trigger falls off with *1/r*^{2}. The following code describes the behavior of these triggers:

$$THRESH = 10000 / safety\_m\arg in^2$$

$$sum = \sum_{i=0}^{num\_input} 10000 / dist\_to\_input_i^2$$

$$output = \begin{cases} TRUE, sum > THRESH \\ FALSE, sum \le THRESH \end{cases}$$

The *safety_margin* indicates the range at which any input causes the trigger to become true. For example, if one enemy is detected at a radius less than the safety margin, then the Is_Invasion trigger is guaranteed to be TRUE.

These four triggers are not independent of each other. If Is_Invasion is true, indicating a close enemy or lots of enemies, then Detect_Enemy has to be true as well because at least one enemy has been detected. However, if an enemy is detected at the edge of the sensor range, then Detect_Enemy is true, but Is_Invasion is not. The reason for this decision was to differentiate between distant and immediate concerns.

4.3 Behavioral Actions

The robots can successfully complete each mission, by utilizing a set of Finite State Automata as actions. Each FSA corresponds to a role: SOLDIER, MECHANIC, or FORAGER. Each role contains a fraction of the behaviors necessary for completing the scenario successfully. A robot team can complete foraging, and overcome all of the hazards only by either switching between roles, or by utilizing a heterogeneous team.

All of the movement behaviors described in the following section include an AVIOD_OBSTACLE primitive in the weighted summation.

4.3.1 SOLDIER Role

The SOLDIER (Figure 2) allows a robot to defend itself in the presence of enemies. When no enemies are detected, a robot using the SOLDIER role executes a herd following behavior, HERDING. HERDING is a weighted summation of AVOID_OBSTACLES, WANDER, and MOVE_TO behaviors, where the goal is the center of the group of robots. If an enemy is detected, then a robot in the SOLDIER role transitions to an INTERCEPT behavior, where the robot moves to the nearest distance intercept point with the enemy. When an enemy is successfully intercepted, it is removed by the environment and relocated. The SOLDIER then transitions back to a HERDING behavior. It needs to be noted that a robot in the SOLDIER role is still vulnerable to the random terrain hazards.

4.3.2 MECHANIC Role

The MECHANIC role (Figure 3) allows robots to assist stuck robots (as indicated by changed color), including those killed by an enemy and affected by an environmental hazard. As with the SOLDIER role, when no stuck robots are detected, the MECHANIC executes the HERDING behavior. When a robot in the MECHANIC role detects a stuck robot, it executes a MOVE_TO_OBJECT behavior with the dead robot as the desired object. When the MECHANIC robot is near the stuck robot, it "fixes" the disabled robot. In simulation, this is done through message passing. On this real robot this is done by bumping, or pushing the stuck robot.

4.3.3 FORAGER Role

A robot that has assumed the FORAGER role (Figure [4]) will search for and collect Anti-Tank mines. This is the task with the most number of intermediate states. Robots in the FORAGER role are involved with three different types of visual objects. The first, mines, are located typically in patches around the environment. The motivation for patches was the belief that over very large areas, if you find one mine then there is likely to be others not to far away. The second type of visual object, the base, is statically located and serves as a collection point for the mines. The third, markers, are placed by a FORAGER robot near a found mine to indicate the presence of a possible patch to all robots.

A FORAGER which is not currently holding a mine, and which cannot detect any markers, starts off in the FIND_OBJECT behavior. FIND_OBJECT is a sub-FSA for finding, and placing markers next to the desired object. Within the sub-FSA FIND_OBJECT, if there are no mines visible, then a robot executes an EXPLORE behavior. EXPLORE is a weighted summation of AVOID_PAST, AVOID_OBSTACLES, and WANDER. When the robot detects a mine, it places a visual marker in the environment that can be detected by all robots, and leaves the FIND_OBJECT behavior.

When the FORAGER can detect a marker, it executes a MOVE_TO_OBJECT behavior with the marker as the desired object. When the robot is NEAR the marker (less than 0.1 m), then it leaves the MOVE_TO_OBJECT behavior. If there are no mines visible, then the robot removes the visual marker, and returns to the FIND_OBJECT behavior. If there is a mine, then the robot transitions to a COLLECT_OBJECT behavior.

The COLLECT_OBJECT behavior is a sub-FSA for picking up mines and returning them to the base. The sub-FSA includes in the following succession: MOVE_TO_OBJECT (object = mine), PICKUP_OBJECT, and MOVE_TO_GOAL, with the goal being the last known location of the base. Every time the base is detected visually, its location in memory is updated.. At the base, the robot drops off the mine

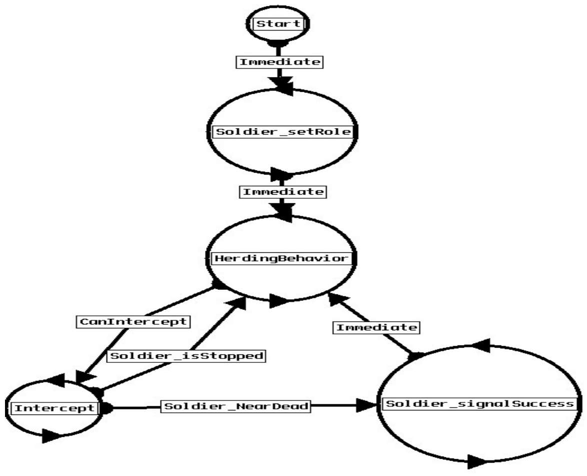

Figure 2. Soldier Role. The robot stays close to the group center, but moves to the nearest distance intercept when it detects an interceptable enemy

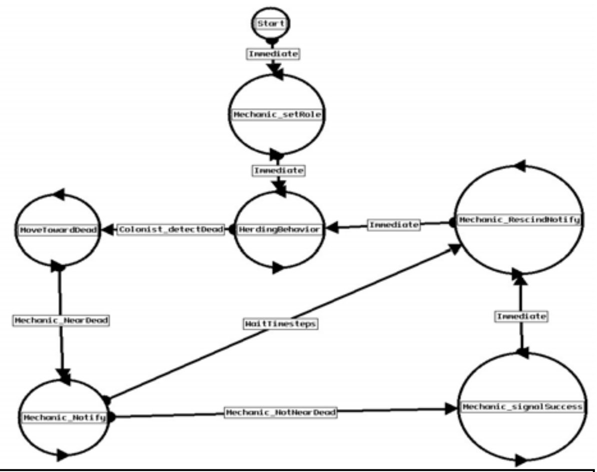

Figure 3. Mechanic Role. The robot stays close to the group center, but moves to dead robots when one becomes visible and fixes it.

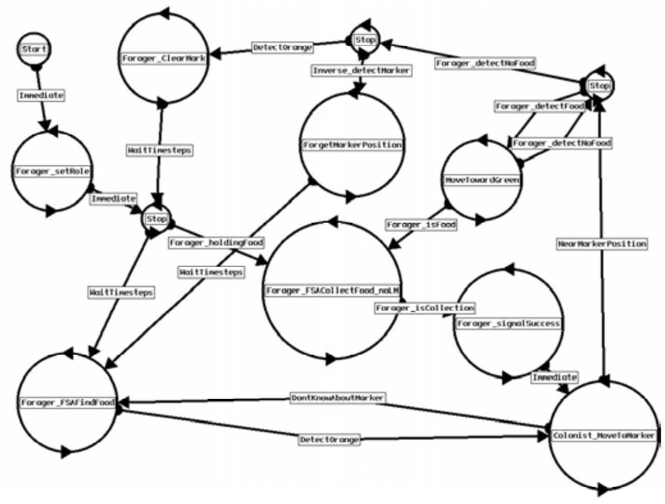

Figure 4. Forager Role. A robot explores the environment using Avoid_Past until it finds a mine. Then it drops a marker, collects the mine, and returns to the base. After dropping off the mine, it moves to the nearest marker and begins looking for mines again. If no mines are present, it removes the marker, and returns to exploring.

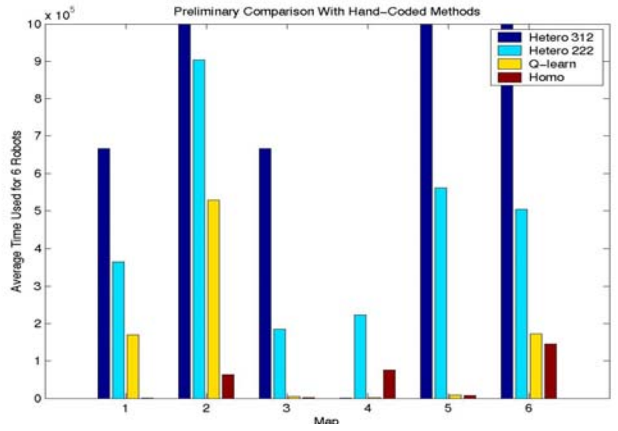

Figure 5. Comparison of Hand Crafted T eams vs. Q Teams.

(DROP_IN_BASKET) and leaves the COLLECT_OBJECT behavior. If no marker is detected, then the robot transitions to FIND_OBJECT, otherwise it moves to the marker.

5. Simulation Results

To test the robots, the Q-learning function is located within another FSA for each individual robot. When the robot is initially started, it signals the *MissionLab* console that it is active and loads the parameters for random hazards. When the robot is either touched by an enemy, or is subject to a random hazard, then it signals the environment that it has died, changes color, and transitions to a STOP action. At this point, the Q-learning function is no longer executing. When the robot is aided by another robot touching it, then it changes color to blue, signals that it is again active, and loads the Q-learning function again from the beginning. The Q-learner does not have to select the last role it was executing before it died.

5.1 Performance Metric

The success of a robot team is judged by the number of iterations the simulation steps through, before all of the mines are removed from a map. The faster a team collects all of the mines, the better the team is judged to have performed.

Sometimes, however, every robot on the team has died before removing the last mine on a map. In this case, the simulation is allowed to run up to 1,000,000 iterations before stopping the test. Since most runs complete before 300,000 iterations have passed, it is highly unlikely that a team will take up to 1,000,000 iterations to complete a map. Therefore in the case of failure, the simulation is judged to have taken the full 1,000,000 steps.

5.2 Hand Coded vs. Q-learn

Teams of 6 Q-learning robots were tested against two types of handcrafted teams. The first type of team used fixed role assignments. Robots were selected to play a particular role for the duration of the scenario. Two different teams were tested. The first team was composed of 3 Foragers, 1 Soldier, and 2 Mechanics. The second

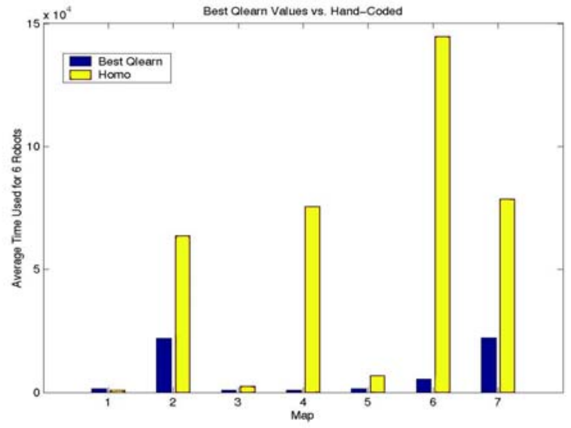

Figure 6. By choosing the optimal values of the Q-learning team for each map, the performance of the Q-learning team can be an order of magnitude greater than that of the homogeneous team.

team tested used 2 of each role. This second team was found to successfully complete more scenarios than the first.

The second type of handcrafted team was a homogenous team, with handcrafted rules for role-switching. In this case, the default role was FORAGER. However, a robot would switch to SOLDIER in the presence of an enemy, and would switch to MECHANIC whenever a dead robot was detected.

Table 1. Map Information

Map	# of Mines	Marked?	# of Patches
1	21	Y*	4
2	146	N	4
3	11	N	1
4	21	Y	1
5	51	Y	3
6**	16	N	1
7	35	N	35

(*) - One patch is marked, while others have to be found
(**) - Environment is an obstacle field

Table [1] shows the information for each of the maps used in this testing. The maps varied in the number of mines present, whether the location of those mines was initially marked, and how many patches existed on the screen. The mines were not guaranteed to be evenly distributed among the patches.

On all but one of these maps, the heterogeneous teams required more iterations to complete the scenario than the other teams. Map 4 was an exception, for the heterogeneous team with 3 foragers. In that map, the mines are all marked, and not far from the base so it was able to clear the mines quickly and efficiently.

The homogeneous team, however, demonstrated exceptional performance on all of the maps. It required fewer iterations on average to complete the scenarios than either the Q-learning team, or the Heterogeneous teams.

The success of the Homogeneous team does not indicate failure for the Q-learning team. The values of the learning rate, exploration rate, and exploration decay were held static for the Q-learning team in Figure [5], but Q-learning depends on selecting the best values for the task. Just using the arbitrarily chosen values, the Q-learning team completed the task in fewer iterations than the Heterogenous teams, and except on map 1, in less than 10 times the number of iterations required by the Homogeneous team. This includes the maps on which the Q-learning algorithm was trained.

Map	Learning Rate	Exploration Rate	Exploration Decay
1	.3	.1	.98
2	.1	.3	.99
3	.3	.2	.1
4	.1	.3	.98
5	.1	.2	.9995
6	.1	.5	.9995
7	.1	.3	.99

Table 2. Q-learning parameters which achieved the best performance on each of the maps.

A second batch of tests were performed to tune the values for each of the environments. The results are displayed in Figure [6]. They show that given the optimal values, the Q-learning team can ultimately match or beat the performance of the Homogeneous team. Table [2] contains the values which achieved the best performance for each map.

5.3 Reward vs. Team Size

The second battery of tests focused on exploring reward function and team size in a variety of environments. 5 different reward function variations were tested with 6 team sizes on 7 different maps.

5.3.1 Number of Robots

The first study probed for the ideal number of robots in colony undertaking this task. The intuitive reaction is the more the better. However, at some point adding more robots is not going to improve the performance of the entire colony. Mataric's analysis of interference has previously explored this phenomenon [6]. This is demonstrated by the graph in Figure [7]. The dashed line displays the averages observed during the testing. The error bars indicate the minimum and maximum values observed for each team size. The solid line is a simple exponent of the form:

$$f(x) = C * K^{Z*X^M}$$

Where:

$$C = 99.6, K = 10091, M = -.16, Z = 1.1$$

The purpose of this curve is just to demonstrate the exponential properties of the performance vs. team size relation in this scenario for these numbers of robots. At some point, the performance of the team does not increase by adding more robots.

5.3.2 Exploring the Reward Space

The second interesting phenomenon demonstrated by the graph in Figure [7], is the unimportance of the rewards applied relative to the size of the team.

Each team was tested with 5 different reward functions. The different reward functions were designed to compare the importance of being rewarded for fixing robots, vs. being rewarded for collecting mines. It was expected that the performance of the team was strongly dependent on the selection of an appropriate reward function. Such a result would be consistent with other work in the area of heterogeneous reward functions[14]. Table [3] has the actual reward ratios used for each function.

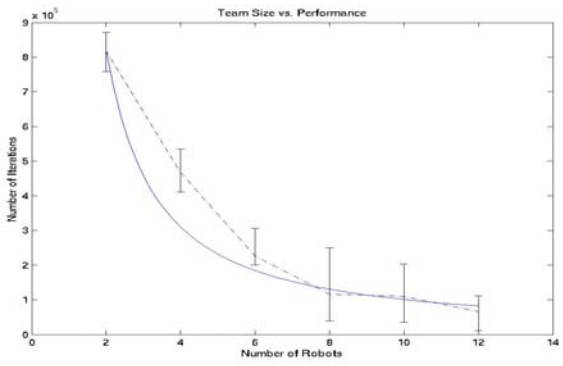

Figure 7. Average Performance of Q-learning team as team size is changed. Dashed line indicates actual measurements, with the error bars indicating variations due to different reward functions within each group. The solid line indicates the values predicted by an exponential decay.

Fixed Robot	Collected Mine
50	10
20	10
20	20
10	20
10	50

Table 3. Reward values used in each reward function.

The reward function variation is displayed as error bars in Figure[7]. The minimum and maximum values seen by each size team, are indicated by the endpoints of the error bars. The size of the team causes an exponential drop-off in the number of iterations required to complete the map, while the change in the reward function only makes local changes about this curve.

For teams up to 12 robots, the importance of selecting the right relationship between the dual rewards is not as important as selecting a larger team. This may change with larger team sizes as the robots begin to interfere with each other.

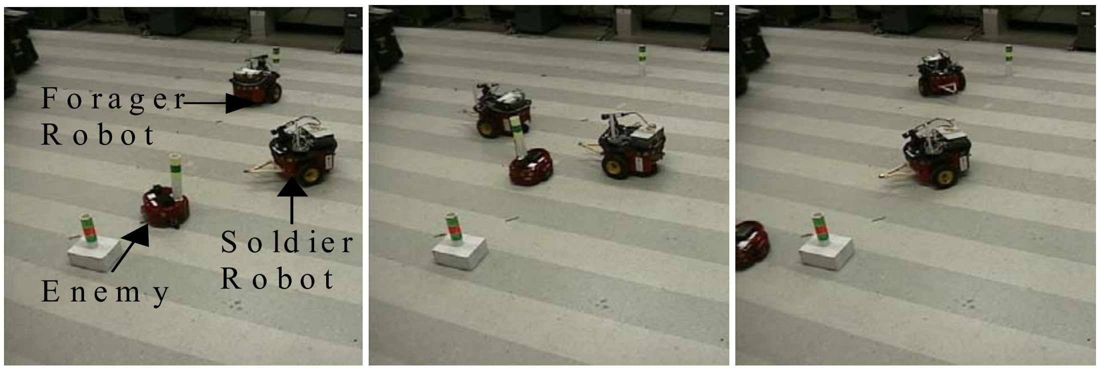

Figure 8. (left) A small enemy robot approaches the two robot team. (middle) The robot on the right is a soldier and begins to intercept, while the second robot changes to MECHANIC and wanders away. (right) The soldier continues to chase the enemy, while the second robot resumes foraging.

6. Robotic Results

The robots used in this work are a pair of 2 Pioneer2-DXE robots made by ActivMedia. Each robot is equipped with an onboard computer and forward and rear sonar for obstacle avoidance. The vision system consists of a Sony XC999 camera with a wide angle lens, connected to a NewtonLabs Cognachrome system for color blob detection. Finally, an electromagnet was attached in the front for picking up metal objects for the foraging task.

For visual object detection, each of the relevant objects (enemies, home base, dead robots, and mines) was identified using a unique 3-color bar code mounted on a vertical pole. Markers, as used in the simulation, were not used in the real experiments because the robot could see a “mine” from anywhere inside the 12’x16’ testing area. Using the cognachrome vision system, a robot could determine the bearing to an object and estimate its distance for use with the perceptual triggers.

For the FORAGER role, robots retrieved metal objects from a platform around the “mine” pole (see Figure [8]). These were then dropped off at a base identified by another pole. Enemies were portrayed by an AmigoBot robot, also made by ActivMedia, remotely controlled by a human operator. When a SOLDIER detects the enemy, it moves to intercept it. The human operator does not let the two robots actually touch, but removes the enemy after an intercept. Finally, a robot in the MECHANIC role moves to and bumps robots displaying the “dead robot” pole. A human operator is in charge of placing and removing the “dead robot” pole. The goal behind the experimentation with real robots was to verify that the results learned from the simulation environment could be transferred directly to a team of real robots.

The simulation results from a two robot training session were selected for running on the real robot team. The learned policy was as follows for the first robot:

Role	% Chosen	Only Common States
Forager	18.5	25.0
Mechanic	37.0	41.7
Soldier	44.5	33.3

And for the second robot:

Role	% Chosen	Only Common States
Forager	51.9	66.7
Mechanic	22.2	8.3
Soldier	25.9	25.0

The first column, % Chosen, indicates the number of states in which the robot would select that role over all feasible states. The second column, Only Common States, is the percentage of feasible states where isInvasion, and isDieOff, are false where that role is selected. The reasoning for the second column is that the isInvasion and isDieOff triggers occur infrequently as they require a close proximity to the interesting object. In general, the first robot stays in the SOLDIER and MECHANIC roles, while the second robot stays in the FORAGER role.

When placed onto the real robots, the simulation code worked as predicted. The second robot stuck to the forager role, collecting and returning objects to the base, while the first robot stayed in the soldier role and watched for enemies. When an enemy became visible, the foraging robot became a MECHANIC and began searching for dead robots, while the SOLDIER robot intercepted the enemy. Figure [8] shows a demonstration of this particular sequence of actions of the robot team.

The next step would be to learn a policy on the real robots themselves. Nonetheless, this simple example confirmed the simulation results indicating that more robots need to be deployed to ensure a successful mission. With the vision system, enemies can remain outside a robot’s field of view for a long time, and can “kill” foraging robots. By adding more robots, there will be more cameras watching a larger area of the field for potential dangers and dead robots.

7. Conclusion

First and foremost, we have demonstrated the extension of our previous Q-learning work [13] to a significantly more complicated action space. In the first paper, it was put forward that Q-learning could be used at any level of the control hierarchy. It could be used to control low-level primitives as demonstrated in work by Asada [2]. It could be used to control behavioral assemblages as demonstrated in the intercept scenario. This form of Q-learning can also be used, as postulated by Diettrich [5], to control complex actions such as Finite State Automata or Roles.

The second aspect that was demonstrated by this work is the success of multiple distinct Q-learning algorithms in a multi-robot scenario. The robot team is not using a global reward algorithm, and it is using no direct communication between the robots. However, the team is still converging to a useful global output. Even just by using the best guess values for the Q-learn function, the results are far better than the performance of a heterogeneous team of robots. If the values are selected specifically for each map, then the performance of the team can even outperform the homogeneous hand coded solution.

Finally, this work demonstrated some interesting aspects about a complex reward function. As suspected, the ideal reward function should be selected on a task-by-task basis with terrain and team information included. However, the results indicate that the reward ratio is not as critical to the success of the team as initially believed. Provided that both functions are being rewarded, a best guess for the reward ratio is good enough to clear the mines. More important to the selection process is getting enough robots onto the field to survive the hazards inherent to the environment.

Acknowledgments
This research was supported under DARPA's Mobile Autonomous Robot Software Program under contract
# DASG60-99-C-0081. Approved for Public Release, distribution unlimited.

References

1) Atrash, A. and Koenig, S., "Probabilistic Planning for Behavior-based Robots", (2001). *Proc. FLAIRS-01*, Key West, FL., pp.531-535.
2) Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1995). "Vision-Based Reinforcement Learning for Purposive Behavior Acquisition", *Proc. IEEE International Conference on Robotics and Automation*, pp.146-153.
3) Balch, T. (1998). "Behavioral Diversity in Learning Robot Teams", Ph.D. Dissertation, College of Computing, Georgia Tech.
4) D'Andrea, R. et al. (2000) "Big Red: The Cornell Small League Robot Soccer Team", in *Lecture Note in Computer Science*, v. 1856, p 657-660
5) Dietterich, Thomas G. {2000}. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", *Journal of Artificial Intelligence Research, 13,* pp. 227-303
6) Dani Goldberg and Maja J Mataric´, "Interference as a Tool for Designing and Evaluating Multi-Robot Controllers", *Proceedings, AAAI-97,* Providence, Rhode Island, Jul 27-31, 1997, 637-642.
7) Kaelbling, Leslie P., Littman, Michael L., and Moore, Andrew W. (1996). "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, Volume 4., p. 237-285
8) Lee, J.B. and Arkin, R.C., (2001). "Learning Momentum: Integration and Experimentation", *IEEE International Conference on Robotics and Automation*, Seoul, Korea. May 2001. pp. 1975-1980
9) Likhachev, M. and Arkin, R.C., (2001). "Spatio-Temporal Case-based Reasoning for Behavioral Selection", *IEEE International Conference on Robotics and Automation*, Seoul, Korea. pp. 1627-1634
10) MacKenzie, D. and Arkin, R., "Evaluating the Usability of Robot Programming Toolsets", (1998). *International Journal of Robotics Research*, Vol. 4, No. 7, pp.381-401
11) Mahadevan, S. and Connell, J., (1991). "Automatic Programming of Behavior-Based Robots Using Reinforcement Learning", *Proc. AAAI-91*, pp. 768-73.
12) Marsalla, S. et. al. (1999) "On being a teammate: Experiences acquired in the design of RoboCup teams" Proceedings of the Third International Conference on Autonomous Agents (Agents'99)
13) Martinson, E., Stoychev, A. , and Arkin, R. (2002) "Robot Behavioral Selection Using Q-learning", to be appear *Proc. of IROS 2002*, Lausanne, Switzerland, October 2002.
14) Mataric´, Maja J. "Reward Functions for Accelerated Learning" in *Machine Learning: Proceedings of the Eleventh International Conference,* William W. Cohen and Haym Hirsh, eds., Morgan Kaufmann Publishers, San Francisco, CA, 1994, 181-189.
15) Stone, P. and Veloso, M. (1995). "Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork", *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages* (ATAL-98), Heidelberg, Germany. pages 293-308
16) Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass.
17) Watkins, C. (1989). "Learning from Delayed Rewards", Ph.D. Thesis, King's College, Cambridge, UK.