MULTISTRATEGY LEARNING METHODS FOR MULTIROBOT SYSTEMS

R.C. Arkin, Y. Endo, B. Lee, D. MacKenzie, and E. Martinson
Mobile Robot Laboratory, College of Computing, Georgia Tech, Atlanta, GA 30332

Abstract: This article describes three different methods for introducing machine learning into a hybrid deliberative/reactive architecture for multirobot systems: learning momentum, Q-learning, and CBR wizards. A range of simulation experiments and results are reported using the Georgia Tech *MissionLab* mission specification system.

Key words: Machine learning, Multirobot systems

1. INTRODUCTION

As part of our research under DARPA's Mobile Autonomous Robot Software Program, we have been investigating the use of multiple levels of machine learning within a hybrid deliberative reactive robotic architecture. Of late we have been focusing on several multirobot team mission scenarios, where we have explored the use of two different forms of reinforcement learning (learning momentum and Q-learning) as well as the application of case-based assistance for complex multirobot mission specification within our *MissionLab* software system.

This paper will present a summary and overview of the results to date obtained for this project in the context of multirobot missions and scenarios. Specifically:

- Learning momentum (LM), which has already been shown to have value in the context of single robots, has now been applied to the multirobot domain. While less than promising results were initially obtained when using LM to switch between discrete roles that a

robot could perform, greater success resulted by allowing the LM module to alter the influences of components of a behavioral assemblage on the robot's final output. The components, when mixed in different proportions, can lead to qualitatively different behaviors. If other robots in the team influence these proportions, appropriate roles for each robot can emerge without the need for inter-robot communication.

- Q-learning is applied as a coordination mechanism for multiple robot teams. Using well-formulated roles, a group of robots with no direct communication can learn to perform complex jobs with multiple subtasks, and each robot is able to select and switch between these roles within an overall team context.
- A CBR mission planning wizard assists users during the pre-mission phase; by retrieving and adapting previously stored mission plans in its memory, whereby even novice users can create complex multi-agent missions without substantial training. Formal usability testing has recently demonstrated its power and utility.

2. LEARNING MOMENTUM

We have previously demonstrated that multiple behavior-based robots can work together to complete a task [1] using statically defined behaviors. If robots within a team are able to alter their behavior at runtime to better suit the current situation, they may be able to further improve their performance. Learning Momentum (LM) is a method by which a robot may change its behavioral parameters in response to perceived situations at run time, and has previously been utilized to enhance obstacle navigation [4]. We now focus on multirobot teams to assess if it can also help improve performance.

A test scenario was invented in which a team of robots is required to protect a fixed target object from attacking enemy robots. The object is placed in an open area, and one or more soldier robots are placed around it. Enemy robots, whose overall goal is to attack the target object, are periodically created at the fringes of the mission area. The soldier team's goal is to intercept and destroy enemies at an optimal distance from the target object based on various attack strategies. A soldier is always able to kill an enemy when intercepted. An enemy's goal is to reach and attack the target object. In a successful attack, the enemy destroys itself when it reaches the target, which remains intact. For the purposes of this experiment, the target is invincible; this allows us to see how many enemies make it to the target.

For these experiments, the mission specification and simulation capabilities of *MissionLab* were used [5]. Two enemy and three solider types were created and pitted against each other in *MissionLab*'s simulator. The first type of enemy (normal) moves directly towards any target object in sight. The second type (decoy) wanders about where it first appeared in an attempt to draw soldiers away so that normal enemies can attack from another direction.

The three soldier types each use a single behavioral assemblage to control their actions. The component behaviors include: *MoveToTarget*, *InterceptEnemies, AvoidSoldiers,* and *AvoidObstacles*. Space limitations prevent a complete description of these behaviors here. Each soldier performs a weighted vector summation of the above behaviors. The difference between each type of soldier comes from how the different behaviors are weighted via their gains.

Two soldier types (*Static1* and *Static2)* are non-learning and use static weights for the vector summation. The third type (*LM)* uses learning momentum to dynamically change the weights. The only difference between the two static types is the *MoveToTarget* weight. The lower weight for the *Static1* types allows more freedom to intercept, but they are unstable in the absence of any enemies as the *AvoidSoldiers* behavior may keep them apart. The *Static2* types are stable without enemies, but they don't have "as long of a reach" when intercepting.

The *LM* soldiers check to see which one of five possible situations it is in:

- *All Clear* – There are no visible enemies.
- *Clear to Enemy* – The soldier is between the target object and an enemy group that is not occluded by another soldier.
- *Flanking Enemy* – The solider sees an enemy group that is not occluded by any soldier, and it is not between that group and the target object.
- *Soldier Needs Help* – All enemy groups are being intercepted by soldiers, but at least one soldier group is overwhelmed by an enemy group. "Overwhelmed" is defined to mean that S/E < T, where S is the size of the intercepting soldier group, E is the size of the enemy group being intercepted, and T is a threshold (set to 0.5 for these experiments).
- *No Soldiers Need Help* – Enemy groups exist, but they are all being intercepted by soldier groups of appropriate size.

Table 1. Behavioral weight adjustments for different situations

	Move To Target	Intercept	Avoid Soldiers
All Clear	0.05	-0.05	0.05
Flanking Enemy	-0.1	0.1	-0.1
Clear To Enemy	-0.1	0.1	0.1
Needs Help	-0.1	0.1	-0.1
No Help	0.1	-0.1	-0.1

The behaviors' weight changes using learning momentum parametric adjustment rules [4] for each situation (Table 1). Groups of 1, 2, 3, and 6 soldiers of each type were run against five different enemy attacking strategies. The first strategy had enemies approaching from north, south, east, and west. The second had enemies approaching from northwest, west, and southwest. The third had decoy enemies to the west while groups of five normal enemies approached from the northeast. The fourth was similar to the third, except decoys also appeared in the south. The last had enemies coming from all directions, but always in pairs from opposite sides of the target object. The target object was in the middle of a 200m x 200m mission area with 3% obstacle coverage, where obstacles ranged from 1 to 5 meters in radius. When 1, 2, or 3 soldiers were present, preliminary results were averaged over 10 runs per enemy strategy, and when 6 soldiers were present, results were averaged over 100 runs per enemy strategy.

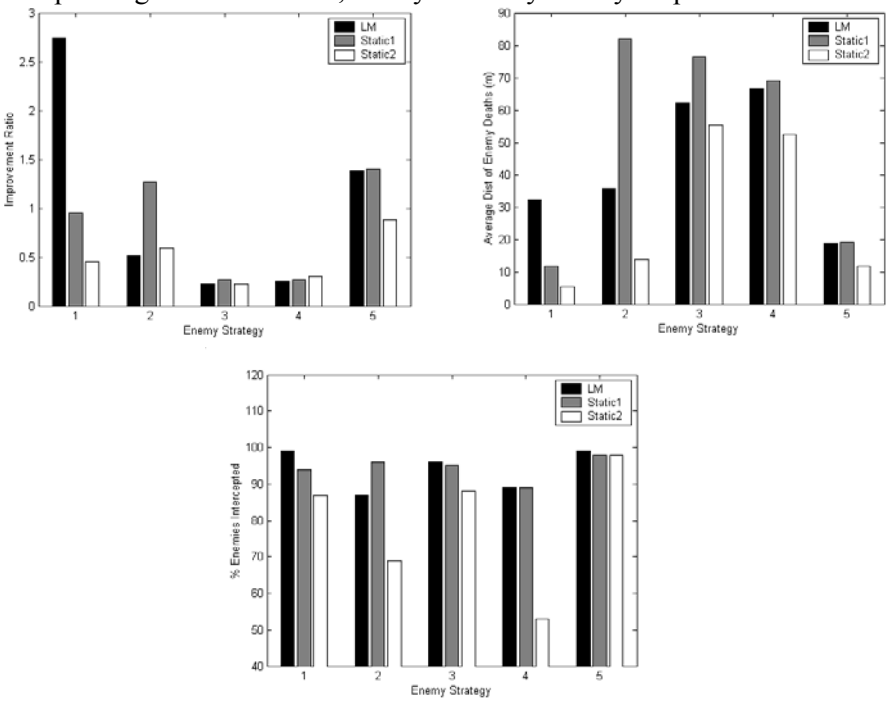
Statistics were gathered on enemy creations, successes, and intercepts. Several metrics were used to gauge the performance of groups of learning (LM) robots versus groups of (static) non-learning ones. From this information, the average distance from the target of an enemy's death and the percentage of enemies intercepted can be determined. A useful metric, similar to "speedup" [1], is the ratio P(n) / P(1)*n, where P(n) is the average distance of enemy death (by intercepts or enemy successes) when n soldiers are present. A ratio value > 1 indicates a *superlinear* improvement, whereas a value < 1 indicates a *sublinear* improvement over a single robot. Figures 1-3 present results for three different teams of six soldiers, each of which has faced the five different enemy strategies.

Comparisons of the results show that *Static1* soldiers nearly always perform better than the *Static2* soldiers for these conditions. Thus if the *LM* soldiers perform as well or better than the *Static1* soldiers, then they also will likely outperform the *Static2* soldiers. *Static2* soldiers might be better for more realistic scenarios, as attacks tend to be intermittent.

The *LM* soldiers, in most cases, performed as well or better than the *Static1* soldiers. The only case where the non-learning robots outperformed the learning ones was for the second enemy strategy. For the first strategy, the learning soldiers had a clear performance advantage. For enemy strategies 3, 4, and 5, the learning and non-learning soldiers performed similarly, with the non-learning getting slightly better numbers for distances of enemy deaths, and the learning soldiers getting slightly better numbers for intercept percentages.

In summary, this experiment shows that Learning Momentum can improve the performance of a multirobot team guarding a single object against multiple enemies, if required that the team be stable (as used in the description of the static soldiers) in the absence of enemies. If that

requirement is not present, however, a static set of weights may be found that allows performance comparable to dynamic weights, although depending on the situation, one system may clearly outperform the other.



Figures 1, 2, and 3. These show the improvement ratios, average distances from the target of enemy deaths, and the interception percentages for each enemy strategy for teams of six soldiers.

3. Q-LEARNING

It is usually the case 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 [8,9] 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 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 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 $(0 \leq \gamma < 1)$.

The multirobot scenario chosen to demonstrate the value of Q-learning of behavioral assemblages [6] is a foraging task in a hostile environment. Anti-Tank mines are scattered about the simulation where 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 ordnance. The robot team is also faced with a variety of hazards. Firstly, 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. The second 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 a robot is either in close proximity 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. The programming, and experimental testing was performed in the *Missionlab* [5] system.

Q-learning is used on each team member 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 encapsulates one robot team member role:

- *Forager:* designed to locate, pick-up, and return anti-tank mines found in the environment to the designated storage area.
- *Soldier:* defends the team from enemies by safely intercepting detected enemies before they reach other robots on the team.
- *Mechanic:* searches for and assists those robots that have been disabled either by the environment or by enemies.

The perceptual state of the world is represented by a set of four Boolean perceptual triggers:

- *Detect_Enemy*: true only if an enemy (red object) is visible within the robot's field of view.

- *Is_Invasion*: true only if an enemy is detected within 5 meters from the robot.
- *Detect_Dead*: true only if a disabled robot (yellow object) is visible within the robot's field of view.
- *Is_DieOff*: true only if a disabled robot is detected within 5 meters of the robot.

Since a combined state-action pair determines the internal state of the Q-learner, there are 48 possible internal states for the Q-learner. 24 of these 48 internal states are not feasible, because two of the triggers, *Is_Invasion* and *Is_DieOff,* are conditionally dependent on other triggers.

To make decisions, and to learn how successful the robot has been, the Q-learner is rewarded when either of two events happens. If the robot successfully drops a mine off at the base, then it receives a reward. If the robot successfully aids a dead robot while in the Mechanic role, then it is also rewarded. For most of the experiments, reward values of 20 and 10 respectively were used and were selected after experimenting with different ratios of reward functions. The third role, Soldier, was not explicitly rewarded, because a successful intercept allowed the robot to continue gathering other rewards.

To test the robots, the Q-learning function is located within a separate 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 to yellow, 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, determines the current state, and queries the existing Q-table to select the proper action. The Q-learner does not have to select the last role it was executing before it died.

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. 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. Therefore in the case of failure, the simulation is judged to have taken the full 1,000,000 steps.

In testing the team's performance, we began with a twofold hypothesis. Firstly, the more robots participating in completing this task, the better the performance. Mataric's analysis of interference [7], however, suggests that

there is a limit in multiagent systems to the level of improvement gained by adding additional robots.

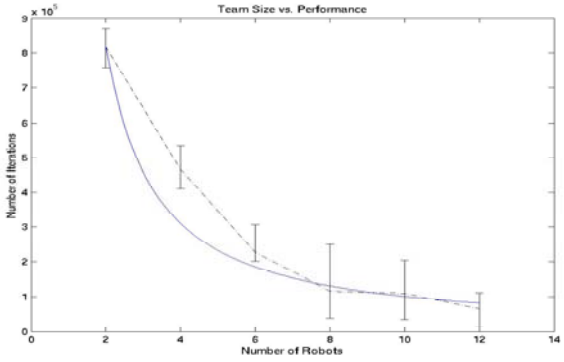


Figure 4. Average Performance of Q-learning team as team size is changed. Dashed line indicates actual measurements, with the error bars indicating variations due to different reward functions within each group. The solid line indicates the values predicted by an exponential decay.

The second hypothesis was that the selection of an ideal reward function was critical to the performance of the team. As such, reward functions were designed to compare the importance of being rewarded for fixing robots versus being rewarded for collecting mines. This hypothesis would be consistent with other work involving heterogeneous reward functions [7].

These hypotheses were then tested using 5 different reward function variations and 6 team sizes in 7 different environments (Fig. 4). Performance is graphed versus team size, with the reward function variation displayed as error bars about the average team size performance. The minimum and maximum values seen by each size team are indicated by the endpoints of said bars. What was demonstrated was that while the size of the team causes an exponential drop-off in the number of iterations required to complete the map, 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.

More important to the performance of the overall team than the reward values are the learning rate and exploration properties. The second battery of testing focused on finding which parameters used during learning could best a handcrafted control strategy for coordinating the team. Position and number of the mines, as well as the proximity of the mines to each other varied in the maps used for this testing.

Teams of 6 Q-learning robots were tested against two types of handcrafted teams. The first type of team used a heterogeneous strategy of

fixed role assignments and in general performed very poorly on all of the maps. 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 switches to SOLDIER in the presence of an enemy, and switches to MECHANIC whenever a dead robot was detected.

In general, the homogeneous handcrafted team performed strongly on all of the maps, and would outperform an arbitrary set of parameters for the Q-learning team. However, by varying the learning rate, exploration rate, and the decay of the exploration rate used for learning the policy on each robot, the Q-learning team could outperform the homogeneous solution on all of the maps.

Real robot experiments are ongoing. We have performed initial work in transferring policies learned in simulation to Pioneer2-dxe robots. Given that a Q-learning algorithm has stabilized in simulation, and that all of the same roles and perceptual triggers exist on the real robot, we can successfully move the learned policy from the simulation to the robots.

4. CBR MISSION PLANNING WIZARD

Suppose there is a solider standing in front of an unknown building. He has multiple survey robots ready to be deployed. His assignment is to inspect the building for existence of biological weapons. However, there might be an enemy sentry patrolling inside the building. Because of the possible contamination, the soldier himself should not enter the building. Because of the complexity of the task, it may be infeasible to teleoperate all of the robots by himself to carry out the mission. Quick specification of an autonomous multirobot mission is therefore desired. The case-based reasoning (CBR) mission planning "wizard", a high-level mission specification feature recently integrated into the *MissionLab* system, was designed especially to help in this type of situation.

The CBR mission planning wizard enables novice users to create complex multirobot mission plans by adding tasks and constraints to a map-based interface. It then composes a suitable mission plan by retrieving relevant mission plan fragments and adapting them to fit the new situation.

The diagram in Figure 5 illustrates the case-based reasoning cycle for problem-solving situations [3]. Out of the six tasks in the cycle, our current implementation of the CBR mission planning wizard supports four of them:

- *Retrieve*: Retrieval of partially matching cases from memory.
- *Propose Ballpark Solution*: Extraction of a solution from the retrieved cases above.
- *Adapt*: Revision of the proposed solution above to fit the new situation.

- *Criticize*: Further adaptation of the solution above.

Implementation of the *Evaluate* (evaluation based on an external feedback) and *Store* (storage of the new solution) tasks is one of our current research agendas. All the cases (mission plans) are currently stored by experts manually. Even with the current implementation, however, our formal usability study (explained below) has showed the significant advantage of the CBR mission planning wizard on creating complex multirobot missions.

In order for the CBR mission planning wizard to propose a mission plan, the user first specifies requirements and preferences of a desired mission using the map-based mission specifier or *mlab* (Fig. 6). Here, the requirements and preferences refer to global mission preferences (e.g., *Number of Robots* and *Maximum Velocity*), local tasks (e.g., biohazard assessment task, explosive ordinance disposal task, and waypoints task), and task-specific constraints (e.g., *Environment*, *Localization Method*, *Enemy Consideration*, and *Aggressiveness*).

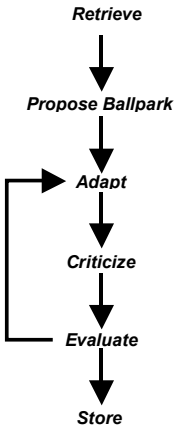


Figure 5: The CBR cycle [3]

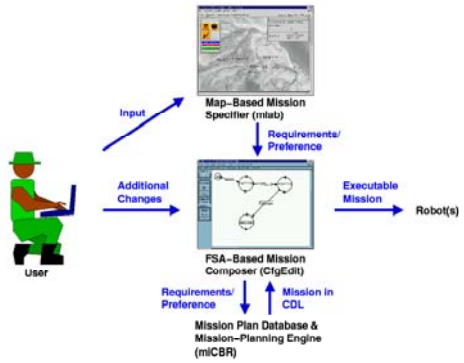


Figure 6: Mission planning using the *MissionLab* system with the CBR mission planning wizard feature.

After the mission requirements and preferences are specified, the information is passed to the mission plan database and mission planning engine (or *miCBR*), where previous mission plans (written in Configuration Description Language [5]) are stored as "cases" along with their indexing features. In *miCBR*, the cases whose indexing features match the user-specified requirements and preferences are extracted and refined to propose a new mission plan that fits with the current situation. The proposed FSA-based mission plan is then sent back to the mission composer (*CfgEdit*), where the user verifies it with the graphical user interface. Once the user is satisfied with the plan, it can be compiled and executed on robots.

The current state of the CBR mission planning wizard was examined with a formal usability study [2]. In terms of multirobot mission

specification, we found that increasing the number of robots involved in a mission does not particularly degrade the quality of the user-created mission plan if the CBR mission planning wizard is used. However, when the mission planning wizard is disabled, the quality of the multirobot mission appears to degrade. Here, the "quality" refers to how many assigned tasks (e.g., "did the robot use waypoints?", "did the robot alert after finding biohazard?" etc.) the user-created plan was able to accomplish.

All of the participants in this study had never previously used *MissionLab*, and they were divided into two control groups: one group (Group A) who only used the basic *MissionLab* features, and the other group (Group B) who used the mission planning wizard. A total of 29 people (14 in Group A, and 15 in Group B) participated. Two tests were conducted on each participant. In Test 1, the participant was asked to create a single robot mission in which the robot follows a series of waypoints to approach an unknown building that may contain biological weapons. Inside the building, the robot had to enter all the opening rooms, check for the existence of a biohazard, and alert if it found any.

Similarly, the overall mission objective in Test 2 is to approach an unknown building by following waypoints, and assess the building for existence of biohazard. However, in Test 2, two "friendly" robots and one enemy sentry are involved in the mission; as the sentry would be patrolling inside the building, each participant had to create a mission plan that would coordinate two robots to look after each other while surveying the rooms. Before conducting the tests, all the participants went through 70 minutes of basic training to learn the functionalities of *MissionLab*, and another 20 minutes training was given to Group B to learn about the CBR mission planning wizard feature.

While the success rate of Group A (without the CBR mission planning wizard) in Test 2 (multirobot mission) was relatively worse than in Test 1 (single robot mission), the success rate of Test 2 by Group B (with the CBR mission planning wizard) seems better than their Test 1 score. The sizeable improvement of Group B's Test 2 may be due to the fact that the participants may have started becoming comfortable using the feature after the first test (Test 1 was always conducted before Test 2). Regardless, we conclude that increasing the number of involved robots in the mission would not degrade the quality of the user-created mission plans if the CBR mission planning wizard is used.

A comparison of average test duration was also determined. Although it is apparent that using the CBR mission planning wizard reduces the time to create mission plans (compared to the system without it), increasing the number of involved robots in the mission does not seem to pose any significant impact on the test duration. In other words, the test duration did

not particularly increase or decrease throughout the tests for either with and without the CBR mission planning wizard.

5. SUMMARY

Incorporation of a range of disparate learning algorithms is both feasible and desirable within a hybrid deliberative/reactive architecture. In particular, we have presented three different methods suitable for multirobot missions: learning momentum, a parametric adjustment technique; Q-learning of roles when represented as behavioral assemblages in the context of team performance; and a case-based wizard to enhance the user's ability to specify complex multirobot missions.

Future work involves expanding other learning algorithms already in use for single robot missions including as well as investigating the interactions between these methods when they are allowed to be active concurrently.

ACKNOWLEDGEMENTS

This research is supported by DARPA/U.S. Army SMDC contract #DASG60-99-C-0081. Approved for Public Release; distribution unlimited.

REFERENCES

1. Balch, T., Arkin, R. C., "Communication in Reactive Multiagent Robotic Systems," *Autonomous Robots*, Vol. 1, 1994, pp. 1-25.
2. Endo, Y., MacKenzie, D.C., and Arkin, R.C. *Usability Evaluation of High-Level User Assistance for Robot Mission Specification*, Georgia Tech Technical Report GIT-GOGSCI-2002/06, College of Computing, Georgia Institute of Technology, 2002.
3. Kolodner, J. and Leake, D. "A Tutorial Introduction to Case-Based Reasoning." *Case-Based Reasoning: Experiences, Lessons and Future Directions*, MIT Pr., 1996, pp 31-65.
4. Lee, J. B., Arkin, R. C., "Learning Momentum: Integration and Experimentation," *Proc 2001 IEEE International Conf. on Robotics and Automation*, May 2001, pp.1975- 1980.
5. MacKenzie, D.C., Arkin, R.C. and Cameron, J.M. "Multiagent Mission Specification and Execution." *Autonomous Robots*, Vol. 4(1), 1997, pp. 29-52.
6. Martinson, E., Stoytchev, A., and Arkin, R., "Robot Behavioral Selection using Q-learning", *Proc. IROS 2002*, Sept. 2002, Lausanne, CH, pp. 970-977.
7. Mataric, M. "Reward Functions for Accelerated Learning" in *Machine Learning: Proc. Eleventh International Conference,* San Francisco, CA, 1994, 181-189.
8. Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass, 1998.
9. Watkins, C., "Learning from Delayed Rewards", Ph.D. Thesis, King's College, Cambridge, UK, 1989.