Robot Behavioral Selection Using Q-learning *

Eric Martinson Alexander Stoytchev Ronald Arkin

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280 U.S.A.

Abstract

Q-learning has often been used in robotics to learn primitive behaviors. However, the complexity of the algorithm increases exponentially with the number of states the robot can be in and the number of actions that it can take. Therefore, it is natural to try to reduce the number of states and actions in order to improve the efficiency of the algorithm. Robot behaviors and behavioral assemblages provide a good level of abstraction which can be used to speed up robot learning. Instead of coordinating a set of primitive actions, we use Q-learning to coordinate a set of well tested behavioral assemblages to accomplish a robotic target intercept mission.

1 Introduction

One driving force behind the use of reinforcement learning in mobile robot tasks is simplicity in task specification. It is often easier for the human designer to specify a reinforcement function and let the robot learn the desired task, than to specify the entire task by hand. This form of task specification is especially attractive because even inexperienced robot users can provide the reinforcement scheme even if they cannot program the robot directly. The main limitation of this approach, however, is that the complexity of reinforcement learning algorithms increases exponentially with the number of possible robot actions. Therefore, it is obviously desirable to reduce the number of actions, if possible.

Robot behaviors and behavioral assemblages [2] provide a good abstraction above the low-level details of sensorimotor interactions. Similarly, perceptual triggers [2] provide a form of digested perceptual information that helps the robot decide when to switch from one behavior to another. This form of encapsulation reduces the action and perceptual state spaces and is thus suitable for learning high-level robot tasks, even entire missions.

Dietterich's research on Hierarchical Reinforcement Learning with MAXQ [5] illustrates a framework in which such coordination could succeed. His work uses a Q-learner for each subtask within a larger hierarchical structure. Each Q-learner however, is an independent coordination mechanism, which, once finished training, could be encoded as a distinct robot behavior. In this paper, we retain a hierarchical framework for reinforcement learning, but replace subtask Q-learner's with completed robot behavioral assemblages. This approach differs from existing approaches in behavior-based robotics (see below) in that we use Q-learning to learn complete robot missions instead of individual behaviors.

This research is part of an ongoing DARPA program entitled Mobile Autonomous Robot Software (MARS). Our overall project focuses on multi-level learning in hybrid deliberative/reactive architectures. Other related papers from our laboratory relevant to this effort include [1, 9, 10].

2 Related Work

Reinforcement learning [8, 17], as used today in behavior-based robotics, can be classified into the following categories depending on the task: *learning primitive behaviors, optimizing primitive behaviors*, and *learning composite tasks.*

When learning a *primitive behavior* the task of the robot is to master one specific behavior based on a reinforcement function. A primitive behavior is a behavior that cannot be broken further into subbehaviors. If a more complex task is desired, several primitive behaviors can be learned individually and then combined together by a sequencing mechanism (see Compositional Learning below).

Mahadevan and Connell [13] used Q-learning to teach a behavior-based robot how to push boxes around a room without getting stuck. The task

*This research is supprted by DARPA/U.S. Army SMDC Contract #DASG60-99-C-0081. Approved for Public Release, distribution unlimited.

was broken manually into three subtasks (behaviors): finding a box, pushing a box, and recovering from stalled situations. The task decomposition method was compared to a monolithic approach where the task is learned as a single behavior. The results show that the task decomposition method learns the task about twice as fast. Asada et al [4], in the context of robot soccer, trained a robot to shoot a ball into a goal using visual feedback. Furthermore, Asada et al [3] evaluated three different methods for learning new behaviors by coordinating existing behaviors learned separately by reinforcement learning.

Behavior optimization is useful when a robot almost knows how to achieve a task but the execution of the task needs to be fine-tuned. The knowledge of the robot may come from prior experience or from a rough solution suggested by a human. Franklin [7] used reinforcement learning to refine robot motor control for nonlinear tasks. Smart and Kaelbling [16] use human-generated control strategies to bootstrap a robot controller that uses Q-learning to refine these policies from experience.

In *Compositional Learning* the assumption is that the robot already has a repertoire of primitive behaviors and its goal is to learn how to organize them to produce a more complex behavior which is contructed from these primitive behaviors. Maes and Brooks [11] used this approach to train a 6-legged robot to walk. An interesting variation of *Compositional Learning* is *Robot Shaping* [6, 15]. In this case a human trainer, or a training program, trains a robot to perform sequential tasks such that each new subtask in the sequence is either a known task or a slight modification of a known task.

3 Overview of Q-Learning

Probably the most widely used reinforcement learning method for robotic systems is Q-Learning [18]. 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 [8], 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*).

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:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} Q^*(s',a') \tag{1}$$

The optimal policy in this case is:

$$\pi^* = arg\max_a Q^*(s,a) \tag{2}$$

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:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3}$$

where α is the learning rate, and gamma is the discount factor($0 \leq \gamma < 1$). It can be proven [18] 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 refer to [18, 8].

4 Task Description

The Q-learning method described above was used to build a behavioral coordination mechanism for an intelligent anti-tank mine robot. The anti-tank mine is designed to intercept enemy tanks as they move down a nearby road and destroy them. The only sensor the mine has available to it determines the location of enemy tanks within a certain radius. From this data the velocity and direction of travel of the tank can be computed.

The sensor information is used in two perceptual triggers: CAN_INTERCEPT and NEAR. The first trigger, CAN_INTERCEPT, is true if the tank is interceptible given its current velocity and the maximum velocity of the mine. The NEAR trigger is true if the mine is within detonation range of the enemy tank. Combined, these two triggers imply 4 distinct perceptual states. Figure 4 shows the possible transitions between these states. The circles represent the perceptual states in binary format: 00 means both triggers are false, 01 means only CAN_INTERCEPT

is true, 10 only NEAR is true, and 11 means both triggers are true. The arrows demonstrate the possible transitions between these states in the intercept task. For example, if the CAN_INTERCEPT trigger is valid (i.e., perceptual state 01), but the anti-tank mine fails to intercept the tank, the state transitions to 00. However, if the mine successfully catches the tank, then the tank stops and the mine remains in state 10 or 11 until the tank is destroyed.

After making an observation, the mine can choose three different actions to perform: WAIT, INTERCEPT, or TERMINATE. The WAIT action causes the mine to stop moving. The INTERCEPT action makes the mine move towards the nearest distance intercept point with the tank. Specifics on how the intercept point is dynamically calculated as the mine moves can be found in [14]. The TERMINATE action blows up the anti-tank mine and ends the learning scenario.

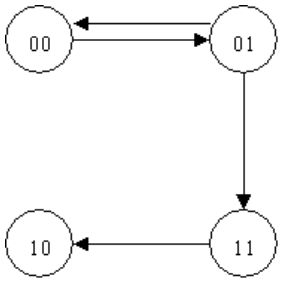

Figure 1: *The perceptual state space of the robot in the intercept task.*

The robot is rewarded only when it successfully intercepts and destroys the tank. This corresponds to executing the TERMINATE action when the NEAR trigger is true. The value of the reward is +10. In Figure 4 we see the values of the Qtable rise above this maximum reward of 10. This occurs because the tank does not have a recognizable terminate state to transition into after the TERMINATE action occurs. Instead, when the tank executes TERMINATE, it remains in the current state. This keeps the expected reward, Q(s',a'), nonzero and potentially large in the update equation (Equation 3). The implementation forcefully ends the scenario after rewarding the Qlearner.

5 Simulation Results

To test and train the Q-learner, the learning process is divided into 200 learning scenarios. A learning scenario consists of 300 time-steps in which the mine is attempting to intercept the tank. If the TERMINATE action is chosen before 300 time-steps, rein-

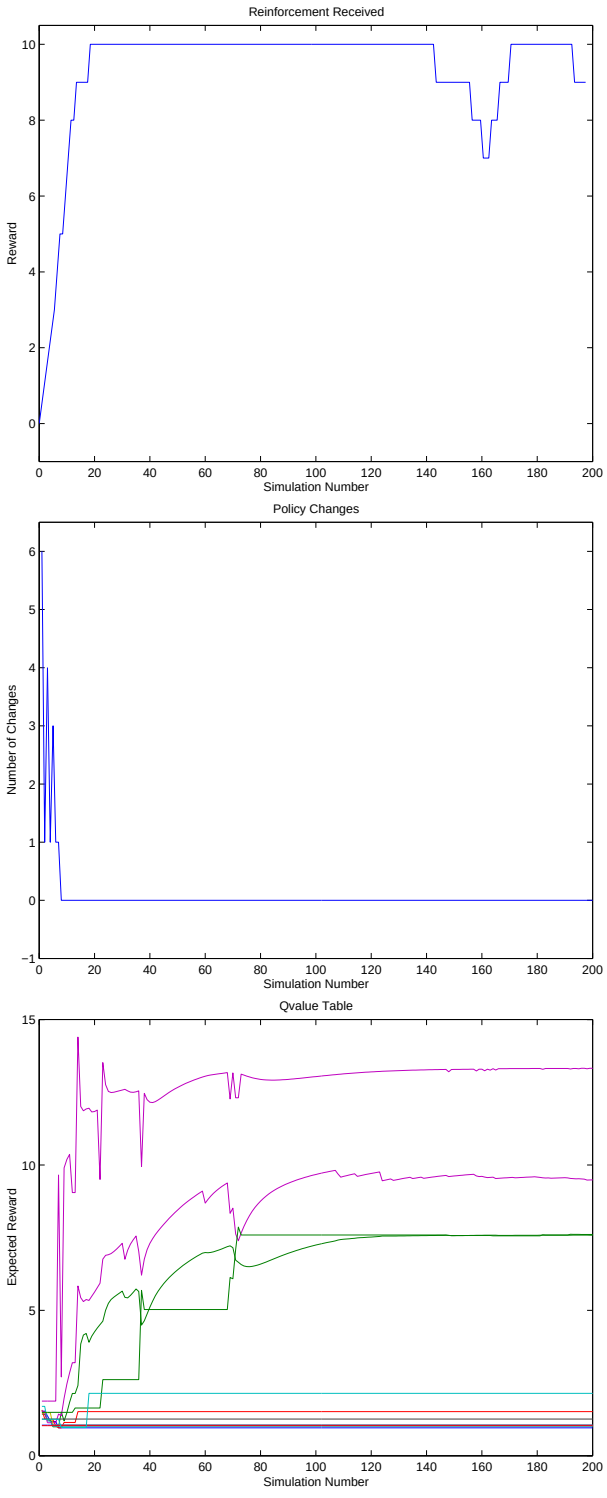

Figure 2: *Convergence properties for a training session with no obstacles. Top: Reinforcement received. Middle: Changes in Policy. Bottom: Qvalue Table. The spike in the reinforcement received (Top) around 160 runs occurred because the Q-learner randomly chose faulty actions in two nearby simulations. The plateau occurs because the robot received the maximum reward for 120 simulation runs.*

forcement immediately ceases. At the end of each scenario the two agents (anti-tank mine and enemy tank) are restarted from their original starting positions and a new scenario begins.

The Qvalue table is initialized in the beginning of every experiment with a set of random values between 0 and 1. Every time the robot's perceptual state changes or a reward is received, the table is queried to determine the action with the highest Q-value. This is different from Kaelbling's approach which queries the table at every time-step (Equation 3).

Whenever the Qvalue table is queried, Watkin's update rule is also applied. The update rule uses a discount factor of $\gamma = 0.9$, and a decaying α value. The α value is initialized to 1.0 and is reduced each time the table is updated using the update formula $\alpha = 0.99\alpha$. An exploration method using a decaying exploration factor was used. The initial exploration rate, e, was set to 0.5 and the following decay formula was used: $e = 0.99e$.

5.1 Convergence Metrics

The success of the Q-learner was judged by the convergence properties of the Q-value table. Several convergence criteria were used:

- reinforcement received vs. trial number
- number of policy changes
- convergence of the Qvalue table

The first metric plots the amount of reinforcement received in every learning scenario. In general, the robot should receive reinforcement more often as the training progresses if the behavioral mapping is converging to a successful solution. It is evident from both top graphs in Figures 4 and 5.1, that the robot is indeed receiving reinforcement more often as its training proceeds. Both graphs were smoothed with a running window filter to show the general learning trend. As seen in both figures, it is impossible to determine where or if convergence occurred, or to identify the exact difference in learning rates between training sessions. The graph oscillates significantly due to variations in sensor readings, obstacles in the environment, or continuing exploration. Another metric is needed to determine stabilization.

The second metric monitors the number of policy changes over time. This metric is derived by looking at the Q-table at each time-step and counting how many times the policy defined by the Q-table changes. The middle graphs of Figures 4 and 5.1 show the number of policy changes that occur during each learning session. The purpose behind this

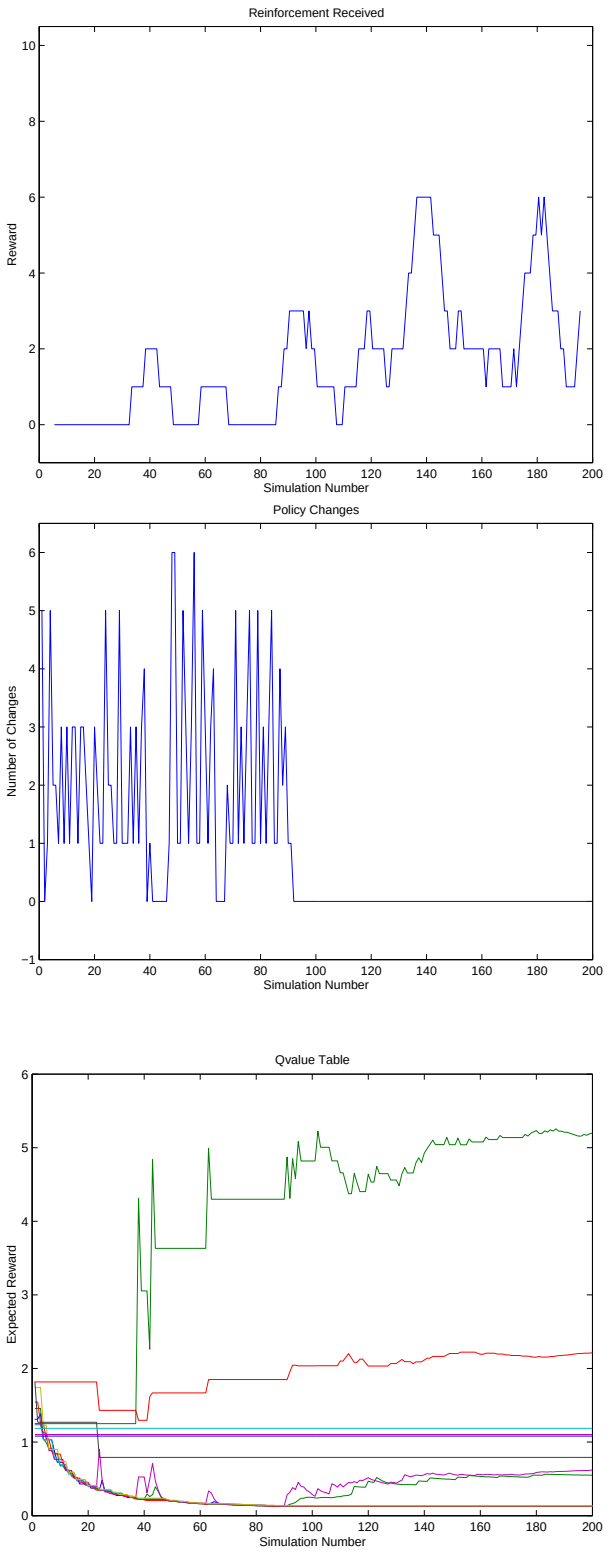

Figure 3: *Convergence properties for a training session with 15% Obstacles. Top: Reinforcement received. Middle: Changes in Policy. Bottom: Qvalue Table.*

metric is to show the volatility of the Q-learner. If it is spiking with a large number of changes it is less likely that the Q-learner is nearing convergence.

The third metric monitors the change in the Qvalues as learning progresses. The lower graphs in Figures 4 and 5.1 demonstrate the performance of this metric. At the point of convergence, several values in the table show a marked increase in value, separating them from the other values in the graph.

5.2 Policy Extraction

Once the Qvalue table has stabilized, a policy for the robot can be extracted by choosing for each internal state the action with the highest associated Q-value. An example is seen in 1.

Perceptual State		Action	Qvalue
Near	CanIntercept		
False	False	WAIT	0.51537
		INTERCEPT *	4.69802
		TERMINATE	0.53961
False	True	WAIT	0.56231
		INTERCEPT *	6.01844
		TERMINATE	0.54739
True	False	WAIT	1.34008
		INTERCEPT	1.20292
		TERMINATE *	6.58522
True	True	WAIT	1.4863
		INTERCEPT	1.10925
		TERMINATE *	11.4678

Table 1: *The values of the Q-table in the final session using 5% Obstacles. A (*) indicates the best action to choose for each internal state. Note that some of the values in the Q-table are greater than the maximum reward.*

5.3 Successful Policies

In this task, two possible policies exist in which the anti-tank mine receives the reward. The first solution is to perform the WAIT action until the CAN_INTERCEPT trigger becomes true. At that time, the mine performs an INTERCEPT action until the NEAR trigger becomes true and then executes the TERMINATE action.

The second possibly correct policy never performs the WAIT action. It immediately chooses the INTERCEPT action even though the nearest distance intercept point is unknown. When the INTERCEPT action is performed without a point of intercept the mine moves in a straight line along whatever direction it is currently facing. This is analogous to patrolling versus lying in wait.

5.4 Convergence in the presence of Obstacles

To further test the convergence properties of the algorithm the task was repeated in the presence of environmental obstacles. Using the *MissionLab* Simulator [12], artificial worlds with 0% through 30% obstacle coverage were generated. Figure 5.4 shows a world with 10% obstacle coverage within the area of concern. The box drawn in the figure demonstrates the maximum dimensions of the obstacle field through which the anti-tank mine has to pass. It is designed to make certain that the mine cannot pass around the edges of the field to catch the tank.

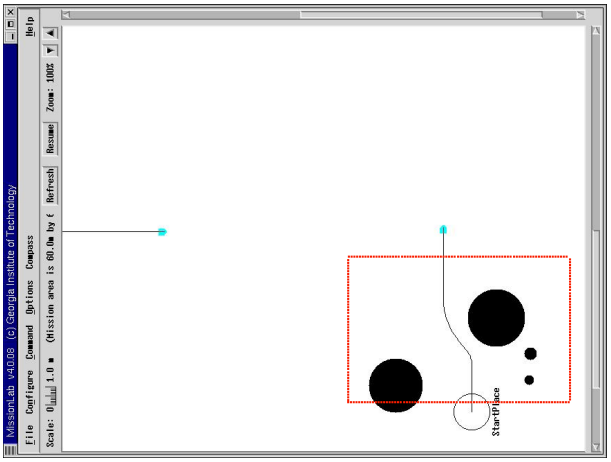

Figure 4: *A sample mission with 10% obstacle coverage. The dotted line represents the bounding area of the obstacle field*

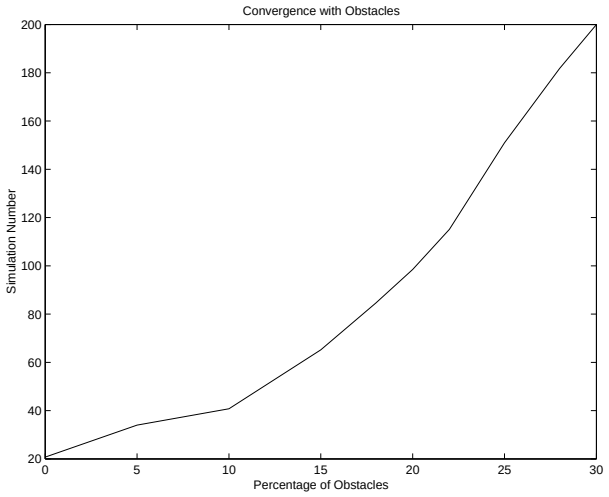

Figure 5: *Evaluation of the convergence of the algorithm with increasing obstacle densities. The plot shows an average over 8 trials for each obstacle density.*

The presence of obstacles slows down the movement of the anti-tank mine robot and in general it takes more time steps for the robot to reach the intercept point. The maximum number of time steps (300) allowed may not be enough in this case since the task

has a time component and if the robot mine does not reach its goal in a certain time window it cannot receive any reinforcement (i.e., the tank will pass by and there won't be a second chance to intercept it in this learning scenario). Therefore, the convergence rates are expectedly slower than in the case with no obstacles. Figure 5.1 shows the results for 15% obstacle coverage. Figure 5.4 shows the degradation in performance for the number of policy changes metric associated with the increase of obstacle coverage. After 8 trials for each of the obstacle densities, the average number of simulations required for convergence rises substantially with the percentage of obstacles in the environment. Obstacle percentages greater than 30% are not shown in this graph, because in all of the testing with greater percentages, the landmine never once received a reward for a successful intercept. Additionally, with 30% obstacle testing, the robot was rewarded only once throughout all of the testing (Fig. 5.4).

As mentioned earlier, the competing solutions problem becomes critical to the stabilization of the Q-learner when obstacles are involved. The greater the number of obstacles, the more often the chosen policy fails to receive a reward. In other words, the two competing solutions will continue to oscillate until the alpha value decays completely. This phenomenon is seen in Figure 5.1, and was typical to the results of training sessions with obstacles.

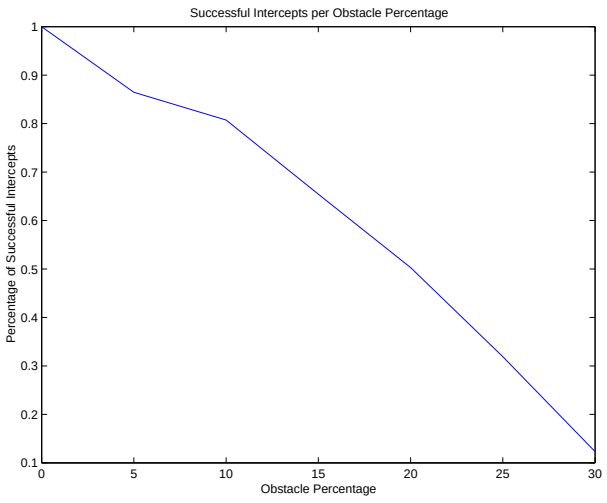

Figure 6: *Percentage of successful intercepts versus obstacle density.*

6 Robot Results

The real robot experimentation used a Nomad 150 robot with a top mounted SICK laser scanner acting as a surrogate intelligent anti-tank mine. A rolling cylinder that could be manually moved in a straight line with a rope and pulley system took the place of the tank (Fig. 7). The robot experiments had two objectives: 1) to verify that the results of the learning performed in simulation can be easily ported to a real robot; and 2) to test if the method is feasible for on-line learning on the robot.

The first objective was achieved in a relatively straightforward manner, except for some minor details that had to be added to the implementation of the physical robot's behaviors and perceptual triggers. The modifications consisted primarily of adding filters to the perceptual triggers to reduce the effect of false readings which did not occur in simulation. At each time step, the readings from the SICK laser scanner were analyzed to detect circular objects possessing a 1-ft radius. This determined the current position of the tank relative to the current location of the mine. The change of tank position over time was used to calculate the point of interception (see [14] for details on this procedure). In most experiments the mine-robot had to move approximately 8 feet to reach the nearest point of interception.

After the differences between simulation and hardware were sorted out, a successful solution derived in simulation was tested on the robot. The surrogate tank was manually moved across the room in a straight line, along a path the mine-robot could intercept. When the mine-robot determined that the intercept was possible, it moved in a straight line towards the intercept point. It then successfully terminated the enemy robot when it came in range. Both of the correct solutions appearing in simulation were tested in this manner.

The second set of testing with the Q-learner was designed to show that learning on the real robot was equivalent to the learning in simulation. Using the improved version of the Q-learner, the intercept scenario was run 20 times on the robot on three separate occasions. This time the Q-learner was started from scratch, with no prior simulation experience. The mine-robot converged to a successful solution by the 15th run in all three tests. Figure 6 shows the properties of the Q-value table for one of these runs. Notice that the actions which result in a successful intercept have diverged significantly from the initial incorrect actions.

7 Summary

This work shows that Q-learning is a valid approach for robot behavioral assemblage selection. In both simulation and on the real robot, we have demonstrated learning successful policies given a limited set of behaviors and perceptual triggers. Addition-

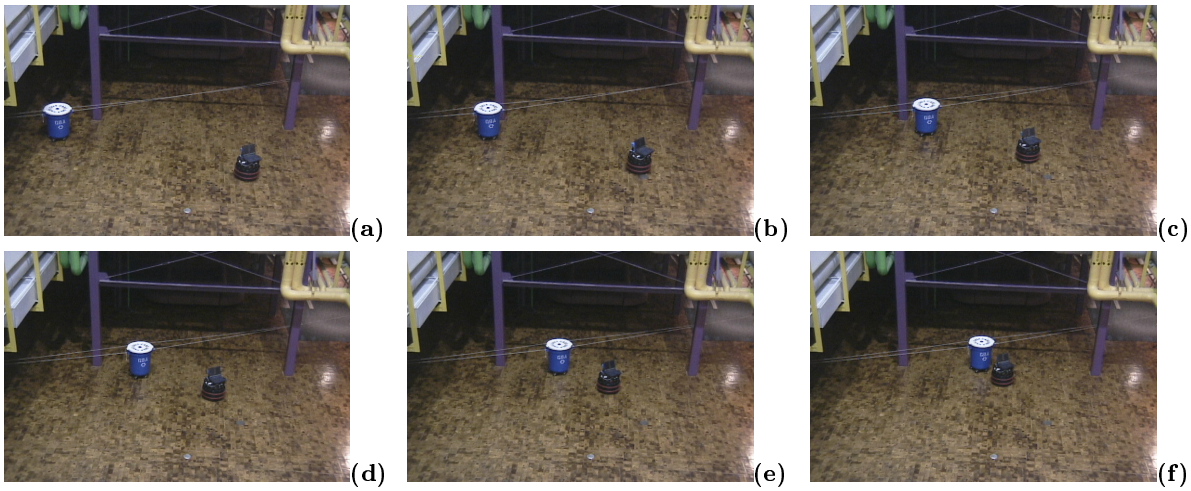

Figure 7: *A successful interception: (a) Tank moves in from the left (b)-(e) Mine moves in straight line toward nearest distance intercept (f) Interception.*

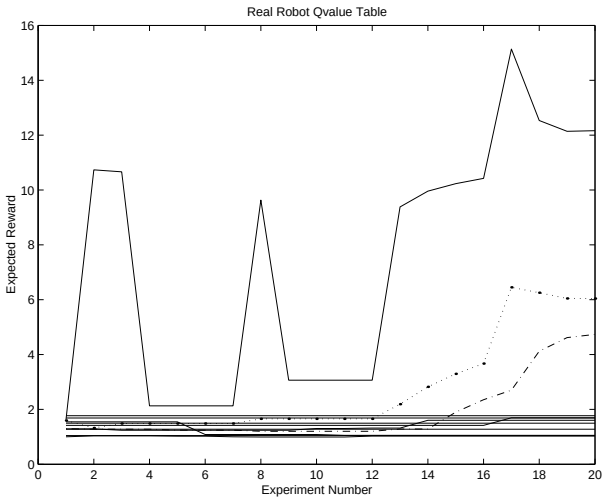

Figure 8: *Q-values during a real robot learning experiment. Notice how three of the values separate from the rest around the 12th run. This is where the learning stabilizes. The remaining values remain almost linear throughout the testing.*

ally, the policies learned by the simulation could be directly ported to a robot platform with the equivalent behaviors and perceptual triggers. In other words, if all of the behaviors and triggers used by the Q-learner work in both environments, then the results of simulation testing, perhaps involving hundreds or thousands of trials, could be used for more complex tasks that would be impossible to learn on the robot itself.

In addition, we demonstrated the effects of environmental noise, i.e., obstacles, on the convergence rate of the Q-learner. We found that the number of simulations required for convergence increases exponentially with the percentage of noise in the environment. The number of possible successful scenarios, however, was found to be inversely related to the percentage of noise in the environment.

There are several issues that remain to be addressed. The first question is the nagging problem of complexity. As the number of behavioral assemblages and perceptual triggers used by the robot increases, the complexity of the algorithm increases exponentially. Using high-level robot behaviors to avoid the curse of dimensionality helps, but this is accomplished at the expense of generality. One solution we are currently exploring is to combine the Q-learning coordination mechanism with other machine learning algorithms which can adjust behavioral parameters in response to environmental pressures.

A second particularly important question to this research is the application of reward. When should continuous reward be applied and when a delayed reward? When should negative reinforcement be applied? Preliminary work on this problem, espe-

cially with regards to negative reinforcement, shows that noise can have an adverse effect on performance with these alternative reward scenarios. Applying a penalty when a trigger has misfired, or obstacles in the environment are preventing success, can prevent the Q-learner from choosing that action ever again.

Acknowledgments

This research has been supported by DARPA under the Mobile Autonomous Robot Software (MARS) Program. The authors would like to thank Brian Ellenberger for his help with the implementation of an early version of the system.

References

[1] Amin, A. and Koenig, S., "Probabilistic Planning for Behavior-based Robots", (2001). *Proc. FLAIRS-01*, Key West, FL., pp.531-535.

[2] Arkin, R.C., (1998). *Behavior-based Robotics*, MIT Press.

[3] Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., and Hosoda, K. (1994)., "Coordination Of Multiple Behaviors Acquired By Vision-Based Reinforcement Learning", *Proc. IEEE International Conference on Intelligent Robots and Systems*, pp.917-924.

[4] 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.

[5] Dietterich, Thomas G. 2000. "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", *Journal of Artificial Intelligence Research, 13*, pp.227-303

[6] Dorigo, M. and Colombetti, M. (1998). *Robot Shaping: An Experiment in Behavior Engineering*, MIT Press/Bradford Books.

[7] Franklin, J.A. (1988). "Refinement of Robot Motor Skills Through Reinforcement Learning", *Proc. 27th IEEE Conference on Decision and Control*, Austin, TX, pp. 1096-1101.

[8] Kaelbling, L.P., Littman, M.L., and Moore, A. W. (1996). "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research*, Volume 4.

[9] Lee, J.B. and Arkin, R.C., (2001). "Learning Momentum: Integration and Experimentation", *IEEE International Conference on Robotics and Automation*, Seoul, Korea.

[10] Likhachev, M. and Arkin, R.C., (2001). "Spatio-Temporal Case-based Reasoning for Behavioral Selection", *IEEE International Conference on Robotics and Automation*, Seoul, Korea.

[11] Maes, P. and Brooks, R. (1990). "Learning to coordinate behaviors". *Proc. Eighth National Conf. on AI*, pp. 896-802, San Mateo, CA.

[12] 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.

[13] Mahadevan, S. and Connell, J., (1991). "Automatic Programming of Behavior-Based Robots Using Reinforcement Learning", *Proc. AAAI-91*, pp. 768-73.

[14] Martinson, E., Stoychev, A. and Arkin, R. (2001). "Robot Behavioral Selection Using Q-Learning", Technical Report GIT-CC-01-19, College of Computing, Georgia Institute of Technology, Atlanta, GA.

[15] Saksida, L.M., Raymond, S.M., and Touretzky, D.S. (1998). "Shaping robot behavior using principles from instrumental conditioning", *Robotics and Autonomous Systems*, 22(3/4):231-249.

[16] Smart, W.D. and Kaelbling, L.P. (2000). "Practical Reinforcement Learning in Continuous Spaces", *Proc. Sixteenth International Conference on Machine Learning.*

[17] Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, Mass.

[18] Watkins, C. (1989). "Learning from Delayed Rewards", Ph.D. Thesis, King's College, Cambridge, UK.