

Reinforcement Learning with a Supervisor for a Mobile Robot in a Real-world Environment*

Karla Conn and Richard Alan Peters
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37235
Email: karla.conn@vanderbilt.edu

Abstract – This paper describes two experiments with supervised reinforcement learning (RL) on a real, mobile robot. Two types of experiments were performed. One tests the robot’s reliability in implementing a navigation task it has been taught by a supervisor. The other, in which new obstacles are placed along the previously learned path to the goal, measures the robot’s robustness to changes in environment. Supervision consisted of human-guided, remote-controlled runs through a navigation task during the initial stages of reinforcement learning. The RL algorithms deployed enabled the robot to learn a path to a goal yet retain the ability to explore different solutions when confronted with a new obstacle. Experimental analysis was based on measurements of average time to reach the goal, the number of failed states encountered during an episode, and how closely the RL learner matched the supervisor’s actions.

Index Terms – Reinforcement learning, mobile robots, Q-learning.

I. INTRODUCTION

Our research applies reinforcement learning techniques to real-world robots. Reinforcement learning has been tested in many simulated environments [1-6] but on a limited basis in real-world scenarios [7-9]. A real-world environment poses more challenges than a simulated environment, such as enlarged state spaces [2], increased computational complexity [6], significant safety issues (a real robot can cause real damage), and longer turnaround times for results. This research measures how well reinforcement-learning techniques perform when applied to a real-world task, managed as a discrete-event dynamic system (DEDS). Modeling the robot’s physical interactions as a DEDS addresses the inherent complexity of the problem by reducing the state space, thus improving the likelihood of a successful experiment.

This work tests reward anticipation models and reinforcement methods on a four-wheeled mobile robot to determine if it can learn through reinforcement to complete a specific navigation task. Reward signals were used as reinforcement during the trials. To associate reward as a consequence of behavior (i.e. to learn the association) the robot must be able to explore its environment and exploit its existing knowledge. Exploration and exploitation are key

elements in reinforcement learning [10]. The robot’s learning architecture incorporated temporal difference learning. The robot used this architecture to shape its decisions while applying the RL algorithms to facilitate its determination of the optimal path from an initial position to the goal position.

There are several works that apply RL techniques to robot navigation problems and support its usefulness. Our work is complementary to this body of research. Although the research in [7-9] admits that real-world environments are more complex than simulated ones, they support the view that RL is useful for robotics. To limit the complexity of their trials[†], the researchers simplified the tasks involved. The initial states in [7-8] placed the robot in the direction of the goal. Therefore, actions needed to reach the goal were in one direction. Success in the task we tested required more behaviors than previously studied, including more than one 90° turn. Moreover, in our work, the robot encountered more obstacles than in the environments used in [7-9]. Both position and orientation define the goal location in this work, whereas [7-8] use position only. The location of the goal is in view at all times in [9], not so in our work. Since we intended for our robot to learn the correct actions in all states, we had to limit the scope of the state space. We did this by teleoperating the robot so it could learn a few sequences of behaviors that would lead it to the goal. Through this direction, it developed state-action pairs that closely matched what it was taught during the supervision phase. We also used a dense reward function to give the robot a more complete assessment of its condition during each step of the task. Although Smart and Kaelbling acknowledged that a dense reward function is more informative than a sparse one, they employed a sparse reward function in their work because it is "much simpler to design" [8] and made their experiments more tractable.

In summary, this paper presents original experiments using RL algorithms on a mobile robot in the real world. An overview of the reinforcement-learning algorithm, its connection to biological observations, and an evaluation of RL techniques used to speed learning are presented in Section II. The methods used in these experiments are outlined in Section III, which includes a description of the

* This research has been supported through DARPA/NASA-JSC Grant #NAG9-1446 and Grant #NNJO4HI19G.

[†] Since the computational complexity of the learning task depends on the size of the state space, it is necessary to simplify either the environment, the task, or both.

task and experimental procedure. The learning algorithm is explained in Section IV. Section V presents results of the experiments as well as comparisons and contrasts between different experiment types. Discussion follows in Section VI with conclusions that can be drawn from the results and suggestions on future applications where this work can continue.

II. BACKGROUND AND MOTIVATION

Robotic researchers often design systems that mimic biological systems because of their adaptableness to changing conditions and their ability to solve a variety of problems robustly. A system within the mammalian brain that has been studied with mathematical descriptions and computer simulations is the dopamine mediated reward-dependent decision processes. The response of dopamine neurons has been found to signal a prediction of future reward [11]. The mathematical models generally incorporate the concept of Temporal Difference (TD) error. TD error can be used as a part of an RL algorithm. TD error is used in the value function that describes the “goodness” of being in a state given a goal. These models can be used to describe the outcomes of a variety of experiments involving states, actions, a policy, a reward function, a value function, and a goal. Our work applied some of these biologically inspired control algorithms to investigate the usefulness of RL in the real world on a mobile robot.

A. Q-learning

To deal with a potentially large search space in a real-world environment and to manage the reward signal over time, we used the RL algorithm Q-learning [12]. Other work [8] applies this algorithm to optimization tasks on mobile robots, because it combines the states and actions into a pair. Q-learning incorporates TD error to calculate the difference between a current Q-value and a prediction of future rewards. Since TD error evaluates the Q-value on a step-by-step basis, it enables us to manage reward at each step along a task. This model-free optimization technique includes TD error, δ_t , as follows. Let r_t , s_t , and a_t be the reward, the state of the system, and the action taken, at time t . Let A represent the set of all possible actions. Then

$$\delta_t = \{r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \mid a' \in A\} \quad (1)$$

and is represented in the one-step equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t, \text{ or equivally,}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \delta_t [r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)], \quad (2)$$

where taking action a_t in state s_t at time t transitions the agent into state s_{t+1} and results in reward r_{t+1} . The updated value of the state-action pair, on the left hand side of Eqn. 2, represents the expected value $Q(s_t, a_t)$ of taking action a_t in state s_t to move into state s_{t+1} and then acting optimally thereafter $\max_{a'} Q(s_{t+1}, a')$, because $\max_{a'}$ represents the action for that state with the maximum Q-value. The learning rate α controls convergence of the data. A higher value of α

will make for quicker learning than lower values [10]. The discount factor γ determines if earlier rewards are more important than later rewards. Setting $\gamma=0$ is not advantageous for the type of RL problem in this work, because the goal for the task is several time steps into the future. Therefore, $\gamma=0.9$ is used to discount future rewards slightly, but there is still a strong emphasis on maximizing the cumulative sum of all future rewards.

The Q-function is usually saved as a table with entries for pairings of states to each action. The Q-values are initially set arbitrarily. As interaction with the environment progresses, the values are updated according to Eqn. 2. This process continues for each step (transition from state s to s') of an episode[‡] [10]. The terminal state is any state that ends interaction with the environment, such as a goal state, a fail state, or when a maximum number of steps has been reached in an episode, whichever comes first. Once a terminal state ends interaction with the environment, another episode can begin. A group of episodes, with Q-values evolving for each episode, is called a trial. Several trials make up an experiment. More information on the Q-learning algorithm can be found in [10][12].

III. PROBLEM OVERVIEW

The scenario for these RL experiments was to have the robot learn how to travel from an initial position in a corridor of an office building to a goal position on the same floor in an adjacent corridor. RL was used to determine an optimal path. Tests were also run to challenge the architecture’s ability to re-plan when the environment changed. A robot using RL can learn the optimal path to a goal while retaining the ability to explore different solutions. In RL experiments, a *policy* is defined as a mapping from *states* to *actions* that lead to subsequent states [10]. In this experiment, states were defined at positions along the path so as to partition the journey into segments connected by simple actions.

A. Robot and Sensors

These experiments were carried out on an *ActivMedia* Pioneer 2-AT robot (Fig. 1). As depicted, the robot was equipped with a SICK laser range finder that reported distance readings to obstacles across a 180-degree span. Two laptops were needed to relay commands and data during the experiments. An off-board controller laptop sent commands to the other laptop that rode on the robot during experiments. Both systems ran a Linux operating system.

B. Software

Player software was used to connect to the robot and its sensors. Player, a robot server, contains configuration files for many commonly used robots and sensors. It acts as a link to the hardware (e.g. SICK laser range finder and robot) for sending commands to the motors and returning data from the range finder. Additionally, NDDS (Network Data

[‡] An episode is one run from the start state to a terminal state.



Fig. 1. Pioneer robot with equipment.

Distribution Service) controlled communications between the robot and its computers [14]. The GTK libraries were used to create a GUI for these experiments. See Fig. 2 for one tab of the GUI used in these experiments. The upper right corner of the tab displays a window with an outline of the robot and range finder. Also displayed are plots of distance data from the range finder that was sent using NDDS from the robot’s laptop to the controller laptop.

C. Environment

Two environments were used in this work. The first tested the robot’s reliability in implementing a task it had been taught. The second used the same location but included obstacles that obstruct the path to the goal. The obstacle course tested the robot’s robustness to completing the task given changes in the environment. Fig. 3 shows a 2D layout of the environment with dimensions, indicators for the start and goal, and outlines of the obstacles used in the robustness experiments. Compared to the space available in the hallways, the obstacles may seem insignificant. From the robot’s point of view however, these obstacles altered the state space in an appreciable way. The periphery measurements in Fig. 3 do not designate the limit of the available hallway area but are indicative of the absolute limits within which the robot traveled during the experiments. Travel outside these bounds was never observed during any of the 450 episodes conducted.

The robot is depicted in the start position on the left of Fig. 3 and in the correct orientation for the goal position on the right. The middle of the robot’s base was centered on the start position at the beginning of every episode. The RL algorithm enabled the robot to learn how to travel from a start position near the table and chair area to the goal position near the robot lab. The robot learned to move forward in the starting hallway until it could turn right into the adjacent perpendicular hallway, travel down it for some distance, and turn left into the goal position. The learning algorithm considered the goal to be found when three conditions were satisfied: the robot was in close proximity to the goal location, it was oriented in the direction facing the

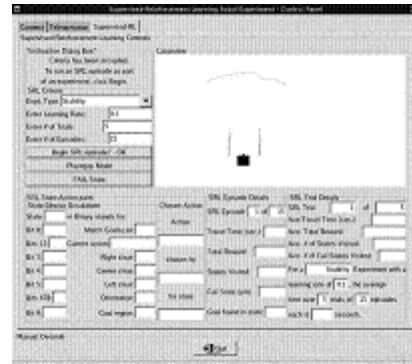


Fig. 2 GTK GUI tab with data feedback window

wall opposite the robot lab, and the laser scan matched a stored scan previously taken in the goal position.

The robot traveled in the environment shown in Fig. 3 for all experiments. This environment incorporates constraints not studied in previous work such as corridors of limited space [9] and corners [7-9]. Provost et al. [6] used a simulated environment in which a mobile robot had to maneuver around a corner. However, the goal in [6] was generally defined as any space 0.5m away from the end of a corridor after making a turn. If robots are to be trained to navigate in highly developed, real-world environments, their goals must be more precise.

During the reliability experiments the two obstacles shown in Fig. 3 were not present. For the robustness experiments, common items from the area were chosen as obstacles and placed to interfere with the path and laser scans. One obstacle was a trashcan with dimensions 0.3m wide by 0.5m long, placed along the wall that the robot faced in the goal position. The other was a chair with dimensions 0.5m square, placed in alignment with the goal position but along the opposite wall.

D. State-Action Pairs

Defining states and actions is one area in RL problems that can be more art than science [10]. The requirement for finding the right balance in state-changing actions may not be difficult for simulated environments, but the real world can be more difficult to divide into distinct states. In real-world environments, actions that will consistently change the state of the environment are not always possible. Small movements may result in a state that looks the same as the

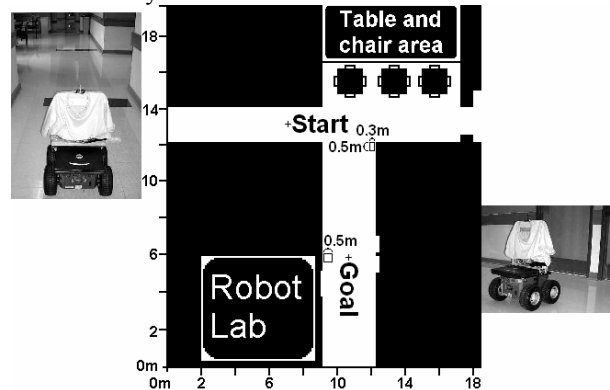


Fig. 3. 2D display of environment with images of the robot in the start position (left) and goal position (right).

previous one ($s = s'$), but larger actions may skip over key transition areas. For experiments in this work, actions were defined that transitioned the robot from its current state, s , into a distinct new state, s' , ($s \neq s'$) with each complete movement. Table I lists the actions, the forward and turn speed values for the actions, and the resulting movement.

Ten Boolean features, corresponding to bits of information, comprised the state representation. The most significant bit indicated whether or not the current laser readings matched the stored readings from the goal position. The next two bits indicated the action taken to transition the robot from the previous state into its current state. Data from the laser scan was used in the next three bits. Each indicated a section (right, front, left) of the scan as clear (1) or blocked (0). These three bits and the thresholds that defined them uniquely described different areas of the environment as distinguishable states and provided a ribbon of safety between the robot and its surroundings. The thresholds for detecting a section as clear or blocked were designed based on the distance needed to complete an action in that direction and added safety. The next three bits (next to the least significant bit) described the direction that the front of the robot faced. Cardinal and ordinal directions were used to establish eight uniform areas within a 360° circle. The least significant bit represented whether the robot was in the goal region (1) or not (0), based on odometry. Thresholds of 3m from the goal position in the x and y direction set the boundaries of the goal region. For example, the initial state could have a binary state representation, 0000100100, which is equivalent to 36_{10} . This state indicates: the current laser scan did not match the recorded goal scan, the current action was stop, the front section alone was clear, the robot faced an eastward direction, and the robot was not in the goal region.

Ten binary bits mean there were $2^{10} = 1024$ states, and four actions were available in each state. Therefore, $Q(s,a)$, which held values for all state-action pairs, contained 4096 values. Reducing the search space if possible quickens learning [15]. In this work, reducing the state space was also beneficial for safety reasons. For example, if the robot's laser sensor detected an obstacle in the front section and the policy chose the forward action, then the robot would travel forward and most likely collide with the obstacle. Therefore, to prevent such collisions, any state-action pair with a blocked region that chose an action in the direction of the blocked region was rejected. Out of the 4096 state-action pairs, 1536 involved states that reported an obstacle in the direction of the action. Therefore, restricting these state-action pairs resulted in a 37.5% reduction of the state-action space.

TABLE I
AVAILABLE ACTIONS.

Action	Forward Speed	Turn Speed	Movement Outcome
Stop	0 mm/sec	0 °/sec	2 sec pause
Forward	300 mm/sec	0 °/sec	500mm forward
Right	0 mm/sec	-10 °/sec	45° right turn
Left	0 mm/sec	10 °/sec	45° left turn

IV. LEARNING ALGORITHM

The learning algorithm used a reward function and a policy. Reinforcement learning algorithms always incorporate some type of reward function, sparse or dense. A sparse reward function returns reward with a value of 0 for most states and perhaps a positive reward of 1 when the goal is found. Dense reward functions use mostly non-zero values. We used a dense reward function to make the robot more knowledgeable about the optimization task. The policy chose actions to maximize the reward function. A supervisor was included in the policy along with random choices and choices based on the best evaluation in the Q-table.

A. Reward Function

The reward function returned a highly positive reward for a goal state, a moderately negative reward for a fail state, and a slightly negative reward for all other states. The reward changed the value of the state-action pair that transitioned the robot into the next state and was based on the category of this state. Ultimately, it was desirable for an action to transition the robot from one state to a goal state. Therefore, the reward was sent to commend the state-action pair that moved the robot into a goal state. This relationship is represented in Eqn. 2. (See Table II for the categories of states and their associated reward function.) The large positive reward for a goal state counteracted any negative reward accumulated during an episode of the task. A fail state was one in which the robot could not move to the right, left, or forward based on the safety constraints. A significant negative reward was returned to condemn any state-action pair that moved the robot into a fail state. The design of the reward was based on the specific task but remained flexible if the task should change. The n_{max} term was the maximum number of steps allowed before an episode was terminated. For this navigation task, 256 steps were sufficient to find a terminal state. However, if the task should change to be longer or shorter, the n_{max} term could have easily been changed to reflect the different task length. The reward for all other states was slightly negative based on a set uniform time constant T_{step} and the current number of steps n . In our experiments at least 1.75 seconds passed for each step of an episode.

B. Policy

Finding the goal in the shortest amount of time maximized the reward signal. Thus, the policy had to develop optimal state-action pairs to transition the robot from the start position to the goal position in the least possible time. In this work, suggested actions from a Supervisor Table and random actions were used to explore the environment, and the Q-table exploited what the robot had learned so far in the episode.

TABLE II
REWARD FUNCTIONS FOR STATES.

State	Reward Function
$r(goal) =$	2000
$r(fail) =$	$-(n_{max} * T_{step}) = -(256 * 1.75) = -448$
$r(other) =$	$-(n * T_{step}) = \{-1.75, -3.5, -5.25...\}$

The Supervisor Table was a list with 1024 entries, each corresponding to a state of the environment used in this work. The values stored in the table represent the action the Supervisor suggested as the best action to take in that state. The Supervisor Table was initialized with zeros. Then the table was populated systematically using a hierarchy of actions: primarily forward, then right turn, and lastly left turn. However, a preference was established that the robot should turn left in the goal region to face the correct east direction. Another systematic coupling of states to actions was made for states that indicated the robot was in the goal region. Lastly, a remote-controlled run through the task shaped the Supervisor Table. A human, specifically the first author, guided the robot through the task, during which the states and actions were recorded in the Supervisor Table. Then, the Supervisor Table was considered an optimal policy for this navigation task. Using this table alone for this task, an agent would find the goal with probability one. Other research has shown advanced controllers, such as the Supervisor Table in these experiments, can improve and quicken learning [2][5].

In the policy, suggested actions from the Supervisor Table were selected with a probability of 0.5. The other half of the time, actions were divided up between random selection and actions with the highest Q-value according to an ϵ -greedy method. The random selection was divided uniformly between the four available actions. Given a state, the greedy portion of the policy compared the Q-values of all the state-action pairs and chose the action associated with the highest value. Variable, ϵ , that governed the balance between choosing random actions and actions from the Q-table, decreased linearly as the number of episodes in a trial increased. Random actions were given preference at the beginning of a trial, but as episodes increased reliance shifted to evaluations in the Q-table.

V. RESULTS

Six experimental conditions were tested for comparison: two environments and three values (0.1, 0.5, 0.9) of the learning rate, α . Since there is no definite answer as to what value of α will yield the best solution [16], three values in the widely-accepted (0-1] range [10] were used for each of the two environments. Each experiment consisted of 5 trials of a set of fifteen episodes. Analysis was broken up by environment, the value of α , and the parameter being measured. Comparisons were made between experiments for time taken to find the goal, percent of episodes that ended in a goal state, and a final Q-table that most resembled the Supervisor Table.

Traversing from the start position to the goal position along the most optimal path resulted in the fastest possible time to complete the task. The goal is attainable from routes other than the optimal one, but completion times will be longer. Therefore, finding the goal quickly is a hallmark of an optimal learner. Moreover, since interaction with the environment can end in a non-goal state, simply finding the goal is an essential characteristic of optimal performance. Thus, the goal of learning is to find an effective policy of

state-action pairs that lead the robot to the goal state. The Supervisor Table is assumed to have an optimal policy for completing this task. Hence, a policy similar to the Supervisor Table is advantageous.

A. Time to Find Goal

To compare the time results against a baseline, one autonomous episode was run with the policy choosing actions from the Supervisor Table only. Comparisons were made between this baseline and the single fastest episode that ended in a goal state for all experimental conditions. The time measurements taken to find a goal state were then averaged across all trials.

The highest value of α produced results with the fastest time to find the goal for a single episode for each environment type (Table III). Table IV shows the average and standard deviation of the time taken to find the goal based on n number of episodes that ended in a goal state instead of a fail state. The results indicate that $\alpha=0.9$ continued to produce good results, and that the experiments in the robustness environment fared better with respect to time than the reliability environment.

The obstacles placed in the robustness environment challenged the robot's ability to find the goal. The obstacles were not designed to block the robot from reaching the goal but simply to complicate the task. Our hypothesis was that different environment settings would produce results of lesser quality than the reliability experiments. However, observations during testing showed that the obstacles played to both sides of the success rate. On some occasions the obstacles presented a fail state that would not exist in the reliability experiments. In other cases the obstacles created states that encouraged actions toward the goal. The results reflect the fact that the obstacles in some cases actually narrowed the environment toward the optimal path to the goal.

TABLE III

SINGLE EPISODE RESULTS FOR FASTEST TIME TO FIND GOAL

Experiment Condition	Fastest Time(s) to Goal
Supervisor baseline	58
Reliability, $\alpha = 0.1$	83
Reliability, $\alpha = 0.5$	65
Reliability, $\alpha = 0.9$	61
Robustness, $\alpha = 0.1$	67
Robustness, $\alpha = 0.5$	63
Robustness, $\alpha = 0.9$	59

TABLE IV

AVERAGE TIME RESULTS ACROSS TRIALS TO FIND GOAL

Experiment Condition	ave.(s)	s.d.(s)	n
Supervisor baseline	58	--	--
Reliability, $\alpha = 0.1$	161	114	12
Reliability, $\alpha = 0.5$	114	30	18
Reliability, $\alpha = 0.9$	115	50	25
Robustness, $\alpha = 0.1$	125	51	16
Robustness, $\alpha = 0.5$	108	40	16
Robustness, $\alpha = 0.9$	110	34	17

B. Percentage of Goal State Found

Reaching the goal position was the objective of the task. Therefore, the percent of episodes that ended in a goal state under a particular experimental condition is noteworthy. The overall average of finding a goal state is graphed in Fig. 4 for comparison between experimental conditions. The robustness results proved better at finding the goal in the case when $\alpha=0.1$ than reliability experiments. For the cases when $\alpha=0.5$ and $\alpha=0.9$, the percent for finding the goal is comparable to results from similar reliability experiments even though there are additional obstacles in the robustness environment.

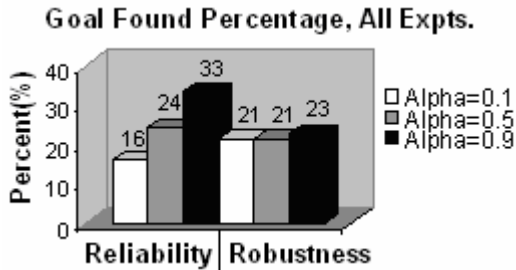


Fig 4. Percent of episodes that found goal

C. Actions Match to Supervisor Table

The Q-table was evaluated on a cumulative basis until the end of a trial. Therefore, the Q-table at the end of the 15th episode would reflect the most-informed conclusions about which actions should be taken in given states. A set of 22, often-visited states were chosen to compare the actions the Supervisor Table would take in those states against the Q-table at the end of a trial. The percent of state-action pairs that matched was averaged across the 5 trials, and the results were graphed to compare which experimental condition lead to a final Q-table that matched the Supervisor Table most closely. Fig. 5 shows $\alpha=0.9$, again, performed the best.

VI. CONCLUSIONS AND FUTURE WORK

In this work, we programmed a robot with the ability to connect several actions together to reach a goal with specific position and orientation. We hypothesized that along the way the robot would develop state-action pairs that would resemble a high-level supervisor table. Q-learning and a dense reward function were implemented. We tested three values of the learning rate, α , in two environments to determine which value of α lead to the most optimal performance across environments.

We found that a high value of α led to the best results in the experiments. Dense reward functions were also found to be helpful to the learning system. If the reward function can represent more to the robot about the given task, it would make sense that the robot could perform better at the task than with a sparse reward function. That hypothesis was supported by the results of the experiments.

The experiments presented here are an initial step towards future work in the area of improving the learning capabilities of real robots using reinforcement-learning techniques. Future experiments could be designed that do not include fail states yet do ensure the robot maintains a safe

Match Supervisor Table, All Expts.

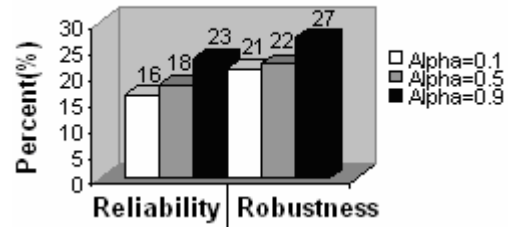


Figure 5. Percent of actions that match Supervisor Table distance from obstacles. Then evaluation could be done based on the robot's ability to find the goal alone, not including episodes that end in a fail state. Also, an extended study using a higher-valued α , such as 0.99, would be a logical next step. This work does not currently involve prior knowledge about the distance from the goal at any step along the episode. However, experiments could be done to include *a priori* knowledge about the environment in order to implement a search algorithm such as A* search with the RL algorithm to test if performance is improved.

REFERENCES

- [1] G. Tesauro, "Temporal-difference learning and TD-Gammon," *Communications of the ACM*, 38(3), 1995.
- [2] J. Rando, A. Barto, and M. Rosenstein, "Combining reinforcement learning with a local control algorithm," in *Proc. 17th Int. Conf. on Machine Learning*, 2000.
- [3] R. Coulom, "Feedforward neural networks in reinforcement learning applied to high-dimensional motor control," in *Proc. 13th Int. Conf. on Algorithmic Learning Theory*, 2002.
- [4] T. S. Dahl, M. J. Mataric, and G. S. Sukhatme, "Adaptive spatio-temporal organization in groups of robots," in *Proc. IEEE/RSS Int. Conf. on Intelligent Robots and Systems*, 2002.
- [5] I. Ghory, "Reinforcement learning in board games," *Tech. Report CSTR-04-004, CS Dept., Univ. of Bristol*, May 2004.
- [6] J. Provost, B. J. Kuipers, and R. Miiikulainen, "Self-Organizing perceptual and temporal abstraction for robot reinforcement learning," in *AAAI-04 Workshop on Learning and Planning in Markov Processes*, pp. 79-84, 2004.
- [7] H. Kimura and S. Kobayashi, "Reinforcement learning for locomotion of a two-linked robot arm," in *Proc. 6th European Workshop on Learning Robots (EWLR-6 1997)*, 1997.
- [8] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 4, pp. 3404-3410, 2002.
- [9] T. Martinez-Marín and T. Duckett, "Fast reinforcement learning for vision-guided mobile robots," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 1425-1430, 2005.
- [10] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- [11] W. Schultz, P. Dayan, and R. Montague, "A neural substrate of prediction and reward," *Science*, vol. 275, pp.1593-1599, 1997.
- [12] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp.279-292, 1992.
- [13] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage project: tools for multi-robot and distributed sensor systems," in *Proc. Int. Conf. on Advanced Robotics*, pp. 317-323, 2003.
- [14] "Can Ethernet be real-time?," Network Data Delivery Service (NDDS), <http://www.rti.com/products/ndds/literature.html>.
- [15] M. Huber and R. A. Grupen, "A hybrid architecture for learning robot control tasks," *Robotics Today*, vol. 13, RI/SME, 2000.
- [16] P. Cook, and G. Hayes, "Could active perception aid navigation of partially observable grid worlds?," *Lecture Notes in Computer Science*, vol. 2837, pp.72-83, March 2003.