SUPERVISED-REINFORCEMENT LEARNING FOR A MOBILE ROBOT IN A REAL-WORLD ENVIRONMENT

By

Karla G. Conn

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August, 2005

Nashville, Tennessee

Approved:

Professor Richard Alan Peters, II

Professor Doug Fisher

ACKNOWLEDGMENTS

This work would not have been possible without the financial support from DARPA/NASA-JSC Grant #NAG9-1446 and Grant #NNJO4HI19G. I would also like to thank my advisor, Dr. Richard Alan Peters, for his support during the research process. The students in the Center for Intelligent Systems (CIS) lab were especially helpful, and I offer my sincerest gratitude to them: Stephen Gordon, Juan Rojas, Josh Phillips, Will Dodd, Paul Fleming, Katherine Achim, Albert Spratley, and Hande Kaymaz-Keskinpala. Flo Fottrell deserves special thanks for all her support. Outside the CIS lab, I would like to thank Megan Rothney for being critical without being coarse and for being a genuine friend. Outside of Vanderbilt University, I owe recognition to Nate Koenig for help with the Player software and John Sweeney for help with the SICK laser.

Finally, let it be known that I thank God for all areas of my life, which I received without cost and will give without cost. My parents have always been supportive of my decisions in life, and I cannot thank them enough! I also thank my siblings and extended family for their continued support as well. Lastly, to Aaron Welch, who has been my comfort on a daily basis throughout this scholastic effort, and I thank him for his unwavering encouragement.

TABLE OF CONTENTS

Page

Appendix

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Can machines be taught? If so, what are some methods for teaching machines a task and how will the machine learn? Machine learning is a field with its focus on systems that can learn through their own experiences and evaluation, resulting in increased competence. Programmers could directly define all behaviors for completing a successful task, but this process becomes quickly limited to well-defined condensed problems. For example, consider the game of backgammon. At first it may seem simple to teach a machine the rules and design a means to improve its play. However, the number of board layouts for a game of backgammon becomes too large to teach the machine directly from a programmer (Tesauro, 1995). The system needs to be able to learn from its mistakes and successes independently; thus creating an autonomous learner who can find the optimal solution through trial-and-error. As Oliver G. Selfridge (2000) so eloquently summarized, “Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever.”

Questioning how machines can learn and be taught often incites questions about how humans learn. Studies have been conducted that show learning is driven by changes in the expectations about future events such as rewards and punishments (Schultz et al., 1997). Moreover, the signals from dopamine neurons have been pinpointed in the brains of primates, which use temporal-difference error to calculate reward (Schultz et al., 1993). Replicating such biological processes is popular in engineering, because it helps explain and further understand how human brains work and allows for the development of more human-like machines.

Reinforcement learning (RL) is a type of machine learning technique that is very attractive because of its connection to actual cognitive processes in primate and human brains (Schultz et al., 1997). RL is based on observations from behavioral experiments on how animals learn predictions and is designed to allow agents to evaluate their actions through reinforcement (Sutton and Barto, 1998). Using RL, agents learn through interaction with the environment and feedback, in the form of rewards, on their performance. This interaction is used to optimize a solution towards a goal. RL problems and solutions often incorporate other learning techniques to improve on the original dynamic-programming algorithm. Such techniques include Q-learning (Watkins, 1989), which defines a value function, and Temporal-Difference (TD) learning, which defines an error as the difference between two states. Both Q-learning and TD learning will be explained in further detail later in this thesis.

Pure reinforcement learning algorithms are an example of unsupervised learning, insomuch as they are designed so that the agent has little to no knowledge of the environment or specific goal at the beginning of a learning trial. Unsupervised learning methods allow an agent to learn through trial-and-error. For example, a person learning how to juggle three balls on their own is unsupervised learning. The person will initially drop all the balls but with practice will learn how to keep them in the air. Therefore, RL on its own is a particular type of unsupervised learning that includes a reward function as feedback from the environment. However, this feedback is used only to evaluate the states, not as any form of instruction for the agent. The agent is simply allowed to try actions and the RL framework evaluates the state the agent is in based on reward and the calculated value of the state. RL methods are set up so that the agent knows very little about the environment or procedure to solve the task but learns by trying several actions. The learning process is slow and computationally intensive. Therefore, RL researchers have tried to find ways that an agent could be told more about the environment, goal, or specific actions that would help achieve the goal whilst maintaining the optimization advantage and learning-on-its-own concept of RL (Dahl et al., 2002; Provost et al., 2004). One resulting solution uses supervised learning techniques and reinforcement learning together (Rosenstein and Barto, 2004). Supervised learning involves some type of external supervision, like being taught to ride a bike. A person learns from their own attempts at riding the bike and from specific instruction on how to balance and pedal from the supervisor, like a teacher. Supervisors can be used in conjunction with RL to speed learning, especially in tasks that face inevitable failure without some prior knowledge. Using RL on real robots presents a greater need for prior knowledge and a descriptive reward function in order to achieve some level of success (Smart and Kaelbling, 2002). The following work contains both a supervisor in the learning process as well as a dense reward function to increase the likelihood of success.

This research focuses on machine learning, specifically testing reinforcement learning techniques on real-world robots. Reinforcement learning is a promising algorithm that has been tested in many simulated environments (Tesauro, G. 1995; Randløv et al., 2000; Coulom, 2002; Dahl et al., 2002; Ghory, 2004; Provost et al., 2004) but on a limited basis in real-world scenarios (Kimura and Kobayashi, 1997; Smart and

Kaelbling, 2002; Martínez-Marín and Duckett, 2005). Far fewer reinforcement learning experiments have been conducted in real-world environments (Tangamchit et al., 2002). Such an environment poses more problematic hurdles than a simulated environment, such as enlarged state spaces, increased computational complexity, significant safety issues when using machines, and longer turnaround of results. This research measures how well reinforcement-learning techniques perform when applied to real-world tasks, managed as a discrete-event dynamic system (DEDS). Approaching a real-world experiment as a DEDS addresses the hurdles of real-world environments, such as reducing the state space and simplifying the complexity of the environment; thus setting the stage for a successful experiment. This work tests reward anticipation models and reinforcement methods on a robot to determine if it can learn through reinforcement to complete a novel task. The focus is on a specific navigation task using an ActivMedia Pioneer 2-AT robot (ActivMedia, 2001). Reward signals will be used as reinforcement during the trials. In order to create a robot that can learn to associate reward as a consequence of its behavior the robot must be able to explore its environment and exploit its existing knowledge. These two objectives of exploration and exploitation are key factors in reinforcement learning (Sutton and Barto, 1998). Reinforcement learning methods including temporal-difference learning will be used in the construction of the robot's learning architecture (Rosenstein and Barto, 2004). The Pioneer 2-AT uses this architecture to shape its decisions while exploring the environment through a SICK laser range finder. The laser presents the world in an egocentric manner. The robot's calculations about the environment are all robot-centric, or with reference to itself. The Pioneer 2-AT uses its robot-centric view of the world while applying the RL algorithms to facilitate its determination of the optimal path from an initial position to the goal position.

In summary, the following work is limited only by the influence of the references listed below. That is to say, there are a plethora of articles and books on the subject of reinforcement learning, spanning the languages of the world. In addition, each presents variations on the original algorithm for approaching the creation of successful applications. This basis sets the scope of this thesis and clarifies the influence on the work. Otherwise, a reader may be misled that this work is born out of the influence of all material ever written on the subject, which is beyond the scope of this thesis. With this scope in mind, the thesis presents original experiments using RL algorithms on a mobile robot in the real world. Chapter II of this thesis presents an overview of the reinforcement learning algorithm, its connection to biological observations, and an evaluation of RL techniques used to speed learning. Chapter III outlines the methods used in these experiments; describing the task and experimental procedure used to conduct the experiments. In chapter IV, the results of the experiments are presented, comparing and contrasting between different experiment types. Chapter V discusses conclusions that can be drawn from the results and offers suggestions on future applications where this work can continue.

CHAPTER II

BACKGROUND

Biological observations can serve as inspiration for robotics researchers. Through these inspirations, the fields of engineering and neuroscience meet. Biological systems show researchers what is possible and allow researchers to study the method behind the process that governs a biological behavior (Murphy, 2000). David Marr (1982), a neurophysiologist, outlined these levels of study into *computational theory*, which has aided the robotics community in defining control architectures for machines. This thesis work gathers motivation from behavioral (Mackintosh, 1983) and physiological (Schultz and Romo, 1988; Schultz, 1992; Schultz et al., 1993) experiments that study the prediction of reward events. The specific role of dopamine has been linked to a neuronal response to the anticipation of reward (Schultz and Romo, 1988). Noticed in animals, this predictive response supported Sutton's development of a model of prediction of reward to be used in engineering systems (Sutton, 1988). Sutton and Barto (1998) represent this anticipation in math equations, and these equations can be used to study learning patterns.

Biological interest

Robotic researchers strive to mimic biological systems, especially the human brain, because of its adaptability to changing conditions and capacity to solve a variety of problems with a robust success rate. One such phenomenon of the brain that has been studied and linked to mathematical descriptions and simulated replications is the role the basal ganglia play in reward-dependent decision processes, studied in animals such as rats (Beninger, 1983; Beninger, 1989; Robbins and Everitt, 1992) and monkeys (Schultz et al., 1993; Houk et al., 1995). The basal ganglia are key sub-cortical brain nuclei that include the substantia nigra in the midbrain, the subthalamic nucleus, the putamen, and the caudate nucleus (Vu, 1998). Dopamine neurons in the midbrain send signals to the striatum, a part of the brain linked to motor control, reward, and motivation (Hikosaka and Wurtz, 1983a, b; Chevalier et al., 1985; Deniau and Chevalier, 1985). To better understand the location of these areas of the brain, look at the left side of Figure 1. The striatum is identified, and the proximity of it to the frontal cortex is shown. The right side of Figure 1 is a different view of the brain with areas of the basal ganglia labeled. Dopamine neurons release dopamine, a neurotransmitter, in the substantia nigra that serves as a main input to the striatum, and noticeably contribute in the brain’s motor control decisions, reward anticipation, and motivation. The dopamine pathway is shown in Figure 2. This image indicates how the substantia nigra in the midbrain releases dopamine to the putamen part of the striatum. All of these regions are involved in the dopamine pathway which effects reward anticipation in the brain.

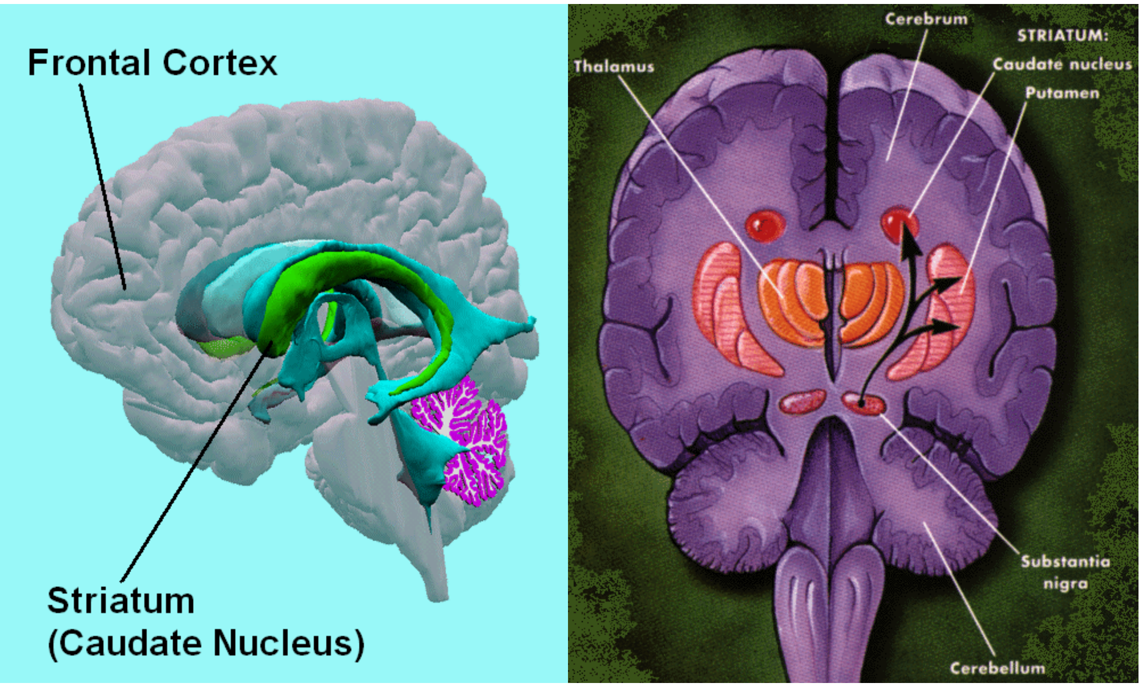


Figure 1. Diagrams of the brain: pointing out the frontal cortex and the striatum (left) (Brown et al., 2002) and indicating the cerebrum, cerebellum, thalamus, striatum, substantia nigra, and their position with respect to each other (right) (Dreyer, 2004)

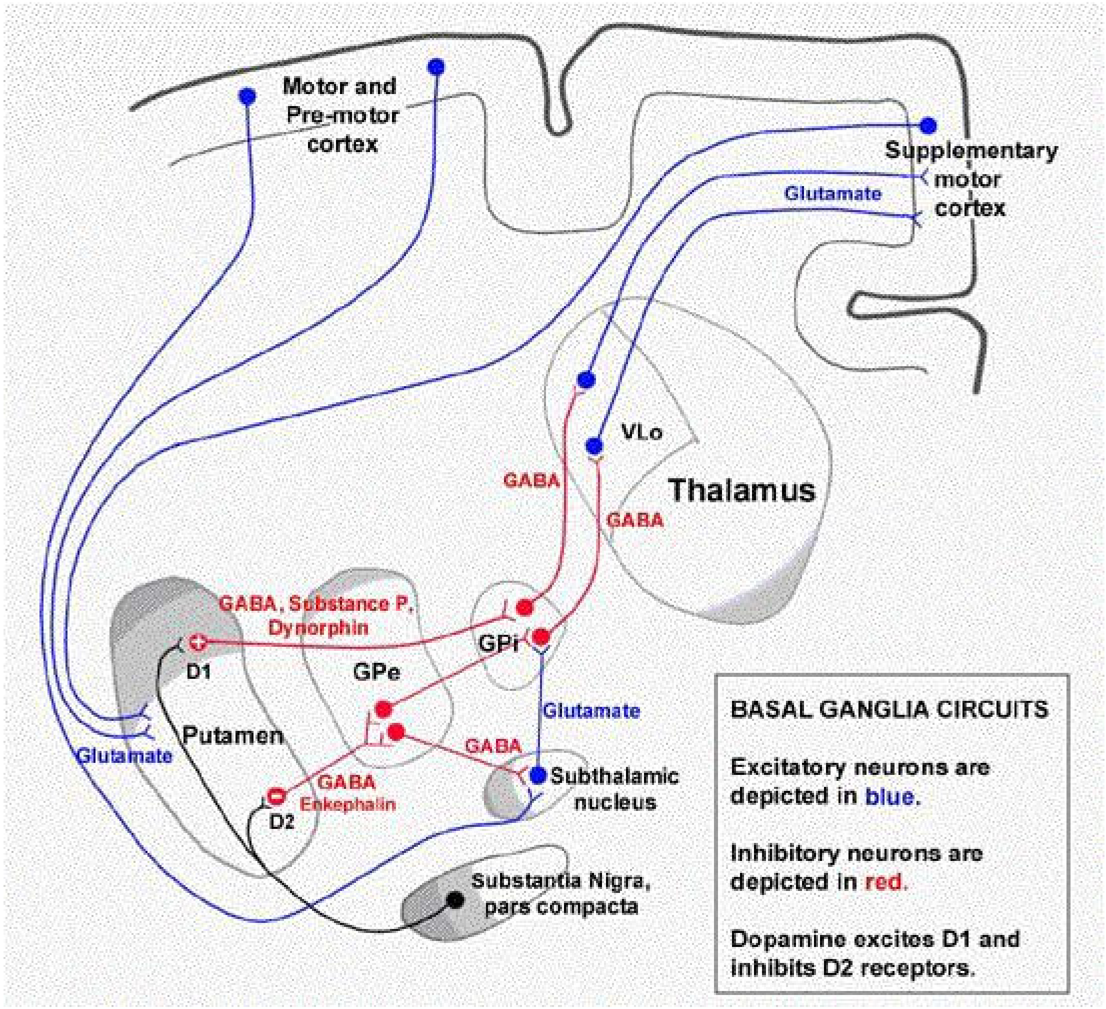


Figure 2. Basal ganglia circuit, including the dopamine pathway (Dreyer, 2004)

The response of dopamine neurons has been studied in animals and found to signal a prediction of reward (Schultz et al., 1997). Electrodes are placed in the basal ganglia region of a monkey's midbrain, where dopamine neurons abound. Then classical conditioning experiments are conducted while the firing rate of the neurons is recorded. Figure 3 shows the results of this study. The top section of the figure shows the results from experiments in which a reward, in this case fruit juice, is given to the monkey. The reward (marked as 'R' in the figure) occurs with no preceding stimulus, and the positive reaction reflects an unexpected reward. The monkey thinks the reward was better than life before the reward, which is indicated in the positive change in dopamine levels. One might conclude that dopamine neurons simply signal a reaction to reward. However, further experiments show the monkey is actually predicting the onset of reward. Next, a conditioned stimulus (marked as 'CS' in the figure) is introduced in the form of a small light before juice is administered. At first, the dopamine response continues to increases only after the juice; however, after several days the monkey learns that the light indicates juice will be given soon. Firing rates for the dopamine neurons shift to increase immediately following the appearance of the small light, and neuronal activity after the reward is stable, as can be seen in the bottom left section of Figure 3. One might now conclude that the dopamine neurons simply signal a reaction to the conditioned stimulus and the reward, such as firing every time something good happens. However, in the bottom left section of the figure, the dopamine neurons do not increase their firing rate when the juice is delivered, because the monkey is not expecting any future reward when the juice is given. Further experiments show that dopamine neurons actually predict when future reward is expected. The last section of Figure 3, in the bottom right, shows that the dopamine neurons can also diminish their response when a conditioned stimulus appears but no reward is received. The dopamine neurons increase their firing rate when the light appears (indicated as 'CS' on the figure), but when the time comes for the reward, no reward is given, and the neurons actually decrease their firing rate. There is negative response indicating the expected future reward (Schultz et al., 1997). So dopamine neurons are signaling more than a reaction to positive rewards that occur, and they signal more than a positive response to conditioned stimulus. Dopamine neurons signal a *prediction in expected future reward,* and this prediction can change from positive to negative.

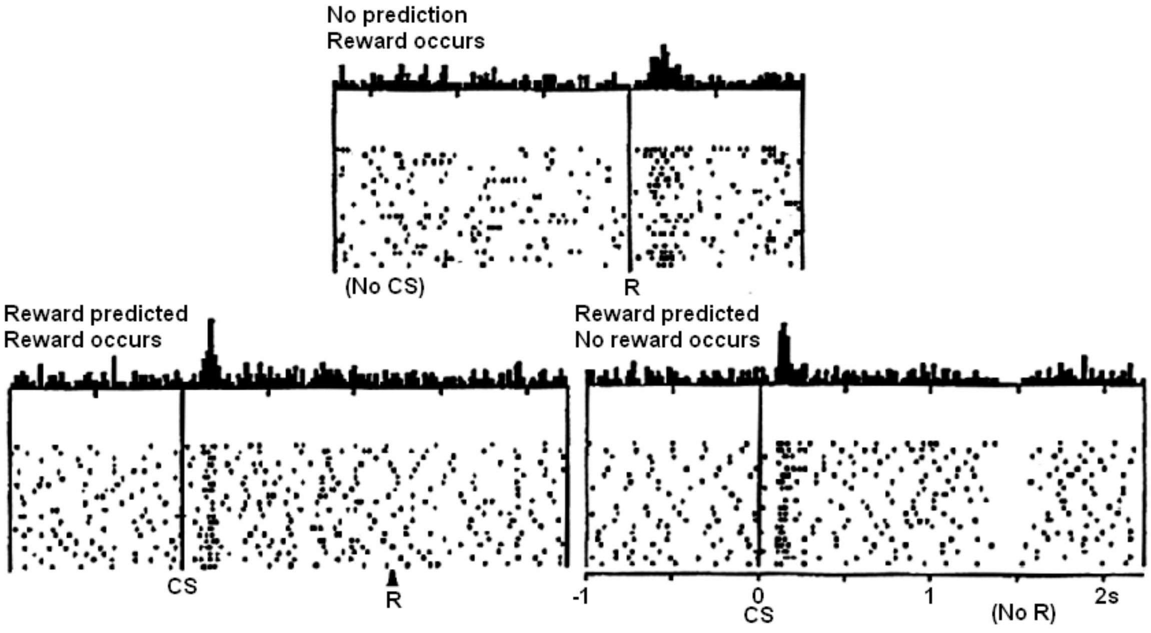


Figure 3. Output of midbrain dopamine neurons during classical conditioning experiments (adapted from (Schultz et al., 1997))

These physiological studies are especially appealing because the prediction of expected future reward is a major element in some noteworthy learning algorithms studied by computer scientists. This connection joins the fields of engineering and neuroscience as mentioned earlier. However, instead of inspirations crossing the boundaries of the two disciplines, now there is some real evidence on how the areas can meet. From behavioral studies, mathematical models have been developed that represent this behavior conditioning during reward-dependent trials in terms of dopamine levels in the basal ganglia. These studies have lead to mathematical representations of the predictions of reward that dopamine neurons in the monkey brain produce and the animal uses as it learns. The equations represent the concept of temporal-difference (TD) error (Sutton, 1988). TD error, like a dopamine neuron, signals a change in expected future reward. This error is the main variable in temporal-difference learning, an often-used addition in reinforcement learning algorithms. Reinforcement learning methods can be applied to a variety of experiments involving states, actions, a policy, a reward function, and an optimization task.

This work applies these biologically inspired control algorithms to experiments using a mobile robot in a real-world environment. These experiments attempt to make strides in investigating the usefulness of RL in the real world. RL is a promising and therefore worthy algorithm to explore; furthermore, RL in the real world is an area worth expanding. This work seeks to further expand mobile robot experiments in the direction of adaptive control using RL in real-world environments.

Reinforcement Learning

In order to attempt such strides, one must understand the reinforcement learning algorithm and useful ways of amending it for improved performance. Reinforcement learning methods solve problems through reinforcement signals and trial-and-error. Reinforcement learning can be thought of as a set of problems, with the objective of finding an optimal solution. With RL an agent in an environment chooses actions to transition the agent into the next state of the environment. This process is illustrated in Figure 4. An agent is placed in an environment, and at each time step, t, an agent chooses an action, a, for state, s, and receives reward, r (Sutton and Barto, 1998).

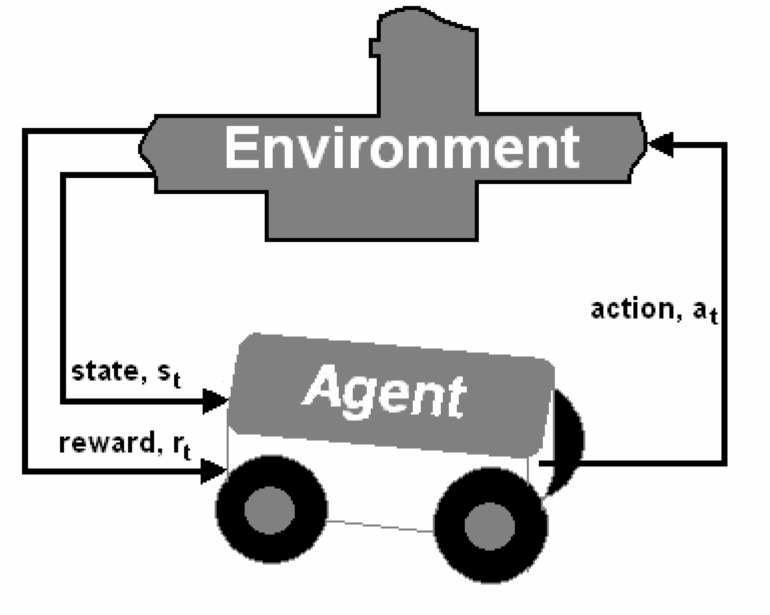


Figure 4. Agent and environment interaction in a reinforcement learning problem

Given a state space *S* and a list of actions *A*, the objective in a RL problem is to optimize the interaction with the environment. When approaching optimization, it can be helpful to think of the environment as a Markov Decision Process (MDP). Furthermore, RL problems are often defined as MDPs, because they satisfy the Markov property. Satisfying the Markov property requires that the state representation include all relevant information needed to make a decision, which can be more than the immediate sensor data. Hidden state information in the environment does not affect the Markov property, as long as the state representation includes relevant information. Essentially, the agent is not held responsible for information unavailable to it, but the state representation is held accountable for presenting enough information. Ultimately, most RL problems can be solved by assuming they are MDPs (Kaelbling et al., 1996).

An MDP characterizes a task in a probabilistic environment with transition probabilities between the states of the environment. In an MDP the probability of transitioning from one state to another with a chosen action is called a transition probability. If the current state is s and an action a is chosen, the probability that the agent transitions into state s' is $P_{ss'}^{a}$. Moreover, all transition probabilities that define the changes between states in an MDP are represented in the following expression:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s^{t} \mid s_{t} = s, a_{t} = a\} \tag{1}$$

If the state space and action space are finite, than the optimization task is defined as a finite MDP. A large majority of RL problems, including the work presented in this thesis are finite MDPs (Sutton and Barto, 1998). For tasks that terminate, as with this work, the terminating states can be described as absorbing. All states that do not end the interaction with the environment are non-absorbing states. Also, if the task ends in a finite number of time steps with probability one, then the MDP is called absorbing (Dayan and Watkins, 2001).

RL allows the agent to optimize its ability to select the best actions that lead it to the goal of the task. As the agent selects actions, the probability of choosing an action that transitions the agent to the next best state increases. In terms of an MDP, if the transition probability for one action is greater than another (i.e. $P_{ss'}^{a_1} > P_{ss'}^{a_2}$) and s' is the next best state, than a_1 is the more-optimal action to choose over a_2 (Kaelbling et al., 1996). A mapping from each state, $s \in S$, and action, $a \in A$, to the probability of choosing that action in the state is called a policy, π. The objective in RL problems is to optimize a policy for the task. For state, s the policy, $\pi(s)$ maps the probability of selecting actions for the set of all actions available in that state, $A(s)$. With the policy,

π, a value function, V^{π}, can be made. In an MDP, the value function for state s is defined as

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\}, \tag{2}$$

where E_{π} is the expected value of the reward using policy π. R_t is the cumulative sum of all discounted rewards:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{3}$$

where the final time step, $T = \infty$ (Sutton and Barto, 1998). Therefore, the value function can be written as

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s\right\}, \end{aligned} \tag{4}$$

where γ is the discount factor, $0 \le \gamma \ge 1$, for k time steps in the future. The discount factor is used to regulate future rewards. If $\gamma = 0$, the agent is concerned with immediate rewards only, disregarding future rewards. Therefore, future rewards become equally as important as current rewards (Kaelbling et al., 1996). Setting $\gamma = 0$ is not advantageous for the type of RL problem set forth is this thesis, because the goal for the task is several time steps into the future. The agent must associate a higher worth for future rewards. Increasing γ's value toward one makes the agent understand the big picture of the optimization task. Future rewards are weighted more heavily than if $\gamma = 0$, and the agent tries to optimize the sum of all future rewards instead of focusing narrowly on immediate rewards (Sutton and Barto, 1998). For this work, $\gamma = 0.9$. Future rewards are discounted slightly, but there is still a strong emphasis on maximizing the cumulative sum of all rewards to reach a goal several time steps into the future.

Solving an RL problem means an optimal policy must be found to maximize reward. Of all the policies for a problem, one will be better or equal to all other policies, in terms of its expected return, R_t. Even for problems that have more than one optimal policy, all are represented $\pi *$. For problems in a finite MDP, an optimal policy can be defined using the value function, and all optimal value functions are represented $V *$,

$$V * (s) = \max_{\pi} V^{\pi}(s)\,, \tag{5}$$

for all $s \in S$. The optimal policy $V * (s)$ is the function RL problems are trying to solve; however, in practice sufficient approximations are made with methods discussed in the next section to satisfy the definition of $V * (s)$ (Sutton and Barto, 1998).

Nevertheless, since $V *$ is an optimal policy for a function, it must satisfy the Bellman optimality equations:

$$\begin{aligned} V * (s) &= \max_{a} E_{\pi *}\{R_t \mid s_t = s, a_t = a\} \\ &= \max_{a} E_{\pi *}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\} \\ &= \max_{a} E_{\pi *}\left\{r_{t+1} + \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s, a_t = a\right\} \\ &= \max_{a} E_{\pi}\{r_{t+1} + \gamma V * (s_{t+1}) \mid s_t = s, a_t = a\} \end{aligned} \tag{6}$$

for all $s \in S$, $a \in A(s)$, and $s' \in S^{+}$, where S^{+} is the state space S including a terminal state. The Bellman optimality equations represent the relationship between the value of a state and the value of its successor states. The value of a state ($E\{\}$) under an optimal policy ($\pi *$) must equal the expected return (R_t) when the best action ($\max_a$) is chosen from that state ($s_t = s$). Solving the Bellman optimality equations is one way to solve the reinforcement learning problem, but this route is rarely used directly. Solving these equations for every state in the environment necessitates that the Markov property holds true, the agent has complete knowledge of the environment, and there is enough computational power to solve the equations. For most tasks, one or more of the requirements cannot be satisfied. Even in tasks where the first two requirements are fulfilled, the last one may not be plausible. Therefore, researchers often settle for approximations when solving RL problems (Sutton and Barto, 1998).

Dynamic Programming

One approach to solving the value function is to apply dynamic programming (DP) algorithms. These algorithms suffer from some of the same requirements of directly solving the Bellman optimality equations, but their presentation is theoretically useful. These methods form a foundation for later techniques (Sutton and Barto, 1998).

One DP approach is called policy iteration. This approach starts with a definition for policy evaluation. Ultimately, policy evaluation and policy improvement steps alternate to solve the optimal policy (Dayan and Watkins, 2001). First, policy evaluation assigns arbitrary values to the initial value function, V_0, for all states. These values are used to look one step ahead in (4). Even though they are set arbitrarily, the values are considered known for the next state and used to calculate the values for the current state. Each successive approximation of the value functions $V_1, V_{2,} V_3 ...$ is found by solving the Bellman equation for V^π, resulting in an iterative procedure:

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) \,|s_t = s, a_t = a\}, \tag{7}$$

where k is an iteration step. As $k \to \infty$, V_k converges to V^π. Iterative policy evaluation is the term for this algorithm. In order to improve on this policy, the action that maximizes the right hand side of (7) must be found. Therefore, a new greedy policy, π', is considered:

$$\pi'(s) = \arg\max_a E\{r_{t+1} + \gamma V^\pi(s_{t+1}) \,|\, s_t = s, a_t = a\}, \tag{8}$$

where a is the action that maximizes the equation. This equation is called policy improvement, and it follows that

$$V^{\pi'}(s) = \max_a E\{r_{t+1} + \gamma V^\pi(s_{t+1}) \,|\, s_t = s, a_t = a\}, \tag{9}$$

for all states. Now that there is a policy, π, that can be improved to make a better policy, π', this process can continue until an optimal policy is found. This sequence converges to V^* and follows the pattern:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V^{\pi_2} \xrightarrow{I} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^*, \tag{10}$$

where $\xrightarrow{E}$ is the policy evaluation and $\xrightarrow{I}$ is the policy improvement. Furthermore, policy iteration is the name of this algorithm (Sutton and Barto, 1998).

Another dynamic programming method is called value iteration, because it directly searches the optimal value function. This method is similar to policy iteration but requires only one sweep of the state space (Kaelbling et al., 1996). Also, policy evaluation and policy improvement are combined:

$$V_{k+1}(s) = \max_a E\{r_{t+1} + \gamma V_k(s_{t+1}) \,|s_t = s, a_t = a\}, \tag{11}$$

for all states (Sutton and Barto, 1998).

These two methods converge to an optimal policy for discounted finite MDPs, but there are drawbacks to them. Dynamic programming algorithms require a complete model of the environment, which may not be possible in some optimization tasks. Also, DP is not ideal for problems with a large number of states, because their solution requires sweeps over the entire state space (Sutton and Barto, 1998). These sweeps can take far too long to process, especially if the task is designed for the agent to react quickly.

Monte Carlo Methods

Another method for solving the optimal policy is called Monte Carlo, to signify they are based on averaging of a complete return. The return is the reward function that RL must maximize, represented in (3). Monte Carlo (MC) algorithms do not require a complete model of the environment as DP methods do. Instead MC methods develop an optimal policy by using sample interactions with the environment and taking the average of known reinforcements. Therefore, MC methods evaluate on an episode-by-episode basis, not on a step-by-step basis. Interaction with the environment must terminate in order to calculate the return for all states. MC methods use several episodes to calculate the average of the known reinforcements and form the value function, represented in (2) (Sutton and Barto, 1998).

However, dynamic programming and Monte Carlo methods are not practical, especially for real-world tasks with mobile robots. DP algorithms must have complete knowledge of the environment, which is often not possible. Even in situations where complete knowledge is possible, the algorithms suffer from the curse of dimensionality. As states increase to describe the environment, the computational resources needed to manage the information drastically increase to the point that processing is not possible. MC methods are also at a disadvantage because of this curse. Additionally, MC methods need sample episodes to calculate the value function and may not be privy to information from states not visited during the samples. Therefore, researchers turned to methods that can solve the value function on a step-by-step basis without a complete model of the environment.

Temporal-Difference Learning

Sutton and Barto (1998) tout, “If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning.” TD learning combines aspects of both DP algorithms and MC methods. Like DP algorithms, it performs value iteration in one-step and does not have to wait for the completion of an episode. Like MC methods it can learn without a complete model of the environment. Unlike these methods, TD learning can calculate the value function on a step-by-step basis. The value of the current state can be calculated using the next immediate reward and value of the next state. The values are updated according to the following equation:

$$V_{k+1}(s_t) = V_k(s_t) + \alpha(r_{t+1} + \gamma V_k(s_{t+1}) - V_k(s_t)), \tag{12}$$

where alpha(α) is a constant step-size learning rate parameter, $0 < \alpha \leq 1$. This update equation can be rewritten using the general form:

$$NewEstimate \leftarrow OldEstimate + StepSize[Target - OldEstimate], \tag{13}$$

where [*Target – OldEstimate*] measures an error. In the constant-α Monte Carlo method, the target is R_t:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)], \tag{14}$$

but the end of an episode must be reached to calculate $V(s_t)$. In TD learning the target is $r_{t+1} + \gamma V(s_{t+1})$:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{15}$$

where $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ represents the TD error, δ_t (Sutton and Barto, 1998).

The update equation is used to evaluate how well the algorithm is estimating $V(s_t)$ and how good of a choice the action is for the current state. If the error is positive, the old estimate of $V(s_t)$ is lower than its actual value. Therefore, the algorithm needs to update the value of being in that state to a higher value. Transitions between states are based on the actions chosen. Therefore, if the error is positive, the action chosen is good and should be chosen more often in that state. In MDP terms, the transition probability for that action in the current state should increase. In TD learning terms, the value of that state for the action is increased. Similarly, if the error is negative, the old estimate of $V(s_t)$ is higher than its actual value. The action should not be favored for the state, and the value of that state for the action is decreased. These estimations have advantages over previously discussed algorithms, because they use previous knowledge and evaluate the value function on a step-by-step basis. On the right hand side of (15), $V(s_t)$ is the one from the previous interaction with the environment. Therefore, TD learning makes use of acquired information that methods like MC do not (Sutton and Barto, 1998).

Q-learning

A popular algorithm that uses TD learning is called Q-learning (Watkins, 1989). This method does not need a complete model of the environment and simplifies the formation of the policy (Sutton and Barto, 1998). When evaluating the value function, two mappings are necessary. One mapping connects the states to actions, $S \rightarrow A$. Another mapping connects the states to their real-numbered values, $S \rightarrow \Re$. Q-learning combines this process to one mapping, $S \times A \rightarrow \Re$ (Kaelbling et al., 1996). Reducing the mappings makes Q-learning more attractive to optimization tasks with large state spaces. Combining the mappings creates some slightly different notation. The Q-function, analogous to the value function, denotes the value of the state when the action is taken, $Q(s,a)$. Q-learning uses state-action pairs to represent the value, instead of states alone. This model-free optimization technique incorporates TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{16}$$

and is represented in the one-step equation:

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_t$, or equally,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)], \tag{17}$$

where taking action a_t in state s_t at time t results in reward r_{t+1} being given and transitions the agent into state s_{t+1}. The updated value of the state-action pair, on the left hand side of 17, 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 $\max_{a'}$ indicates the action with the maximum estimate for state s_{t+1} up to time, t. Therefore, the algorithm adjusts its estimate for a state-action pair with each action the agent takes (Dayan and Watkins, 2001).

The function is usually saved in tabular form with a list of the states and each action. The Q-values are initially set arbitrarily. As interaction with the environment continues, the values are updated according to (17). This process continues for each step (transition from state s to s') of an episode(Sutton and Barto, 1998). In RL problems, an episode is one run from the start state to a terminal state. The terminal state could be any state that ends interaction with the environment, which includes the goal state, a fail state, or when a maximum number of steps has been reached in an episode. Once a terminal state ends interaction with the environment, another episode can begin. At the start of each episode, the start state is initialized, but the Q-values update from the previous episode, following initialization from the first episode. A group of episodes, with Q-values evolving for each episode, is called a trial. Several trails make up an experiment. The sequence of the Q-learning algorithm is laid out in Figure 5.

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode);
        Initialize $s$
        Repeat (for each step of episode):
                Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
                Take action $a$, observe $r$, $s'$
                $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$;
        until $s$ is terminal
```

Figure 5. Q-learning algorithm with TD control (Sutton and Barto, 1998)

The ε-greedy policy mentioned in Figure 5 uses the parameter epsilon (ε) to vary when a random action is chosen and when the best action in terms of the Q-values is chosen. Greedy policies derive their name because of their design to make selections based solely on the maximum value of the available choices. An ε-greedy policy is not purely greedy, because it does not select the action with the maximum Q-value all the time. Selection of random actions versus an action with the maximum estimated Q-value is based on the value of ε. Policies that use ε-greedy methods have been found to improve Q-learning algorithms. Therefore, they are often used in RL solutions that apply Q-learning (Sutton and Barto, 1998).

CHAPTER III

METHODS

This research seeks to use RL as a learning architecture that develops an optimal path between an initial state and goal state as well as an architecture that can adapt to changes in the environment, requiring new paths between states and actions to be formed. The system was tested using a Segway Robotic Mobility Platform (RMP) (Segway LLC, n.d.) and a Pioneer 2-AT (ActivMedia, 2001) robot in an indoor navigation task. In each experiment, the agent (in this case a mobile robot) had access to the same table of supervisor actions. The supervisor consists of human-guided remote-controlled runs through the navigation task and acts as a teacher for the initial stages of reinforcement learning. Six experimental conditions were tested for comparison. The first three experiments tested the robot's *stability* in implementing a task it has been taught. The second set of experiments included obstacles that obstruct the path to the goal and tested the robot's *flexibility* when completing the task with a change in the environment.

Among the stability and flexibility experiments, three values of alpha (α), the learning rate in an RL problem, were examined. Conclusions are mixed as to what value of α, which ranges from zero to one, is optimal for a given optimization task employing RL techniques. Sufficient information was not found in the literature (consisting of all works cited in the Bibliography section of this thesis) to answer what value of alpha would be most beneficial for the navigation task studied in these experiments. Experts say a high value of α (around 0.9) will make for quicker learning than lower values

(Sutton and Barto, 1998). Some experiments use lower values (around 0.2) and show that this value is necessary for slower learning (Smart and Kaelbling, 2002). Slower learning should aid in a more flexible evaluation of the environment, because the Q-values are not changing as dramatically as with a higher value of α. However, in a paper comparing RL techniques, Crook and Hayes (2003) conclude, "The success of an agent with 1-step backup learning algorithms [can be] heavily dependent on the value of alpha (α), [even resulting in] no satisfying policies being learnt." Since there is no definite answer as to what value of α will yield the most optimal solution, investigations into three values of alpha (specifically 0.1, 0.5, and 0.9) for each of the two types of experiments were performed. These values define the six experiments conducted for this thesis.

Evaluation of the results is based on measurements of time taken to reach the goal, averaged across the trials in an experiment. Also, the number of goal states found during an experiment is compared across trials. Assessments will also be made for how closely the Q-learning table of optimal actions matches the supervisor's table of actions for a specific group of significant states. Analysis of the results seeks to find the conditions which yield the highest success rate for finding the goal.

Problem Overview

The premise of the optimization task for these RL experiments is to have the robot learn how to travel from an initial position on the third floor of Featheringill Hall on the campus of Vanderbilt University to a goal position on the same floor using reinforcement learning to determine the optimal path. Tests were also run to challenge the architecture's ability to re-plan when the environment changes. RL methods take advantage of exploring the environment while choosing the actions predicted as most rewarding. Therefore, a robot using RL can learn the optimal path to a goal but also retains the means to explore different solutions. In reinforcement learning experiments, the *states* are mapped to *actions* that connect between the next states that define a *policy* for the robot to use for a given task (Sutton and Barto, 1998). In this experiment, states are defined at positions along the task that break up the journey into segments connected by simple actions. The possible variables of raw data that can be used to define the states include: distance and orientation readings from the robot's wheel odometers, distance readings from a SICK laser range finder, and the angle of a particular distance reading from the laser scans. Preliminary data files were recorded using the real Segway robot in a real-world environment and used as the basis for defining the states of the environment for the reinforcement learning episodes using other real robots. Gazebo and Player/Stage software were also used as simulation tools to test experiments before running trials with the robot in the real world. Stage has been used as an environment for other RL experiments on simulated mobile robots (Dahl et al., 2002; Provost et al., 2004), which encouraged the desire to go a step further in this work and use RL in the real world to test how well the theory holds in a real environment compared to a simulated one.

Environment

The environment for the RL agent in this work consisted of an area on the third floor of Featheringill Hall (Figure 6) on the campus of Vanderbilt University. This floor is home to the Center for Intelligent Systems (CIS) robotics research lab, which houses many mobile robots; including Segway and a Pioneer 2-AT robot named Skeeter, and a humanoid robot, named ISAC (Intelligent Soft-Arm Control). See Figure 7 for pictures of the mentioned robots and Figure 8 for a layout of the third floor with rooms marked where the robots are kept, for easy reference to these locations and robots later.

Figure 6. Photo of outside of Feathergill Hall at Vanderbilt University (taken by author)

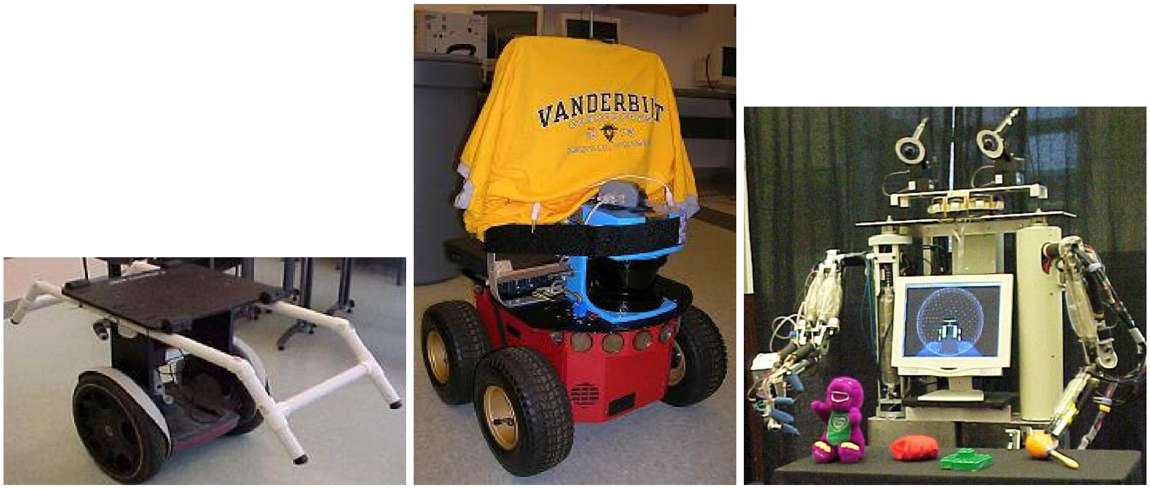


Figure 7. Photos of Segway (left), Pioneer 2-AT named Skeeter (middle) (taken by author), and ISAC (right) (taken from (CIS, 2005))

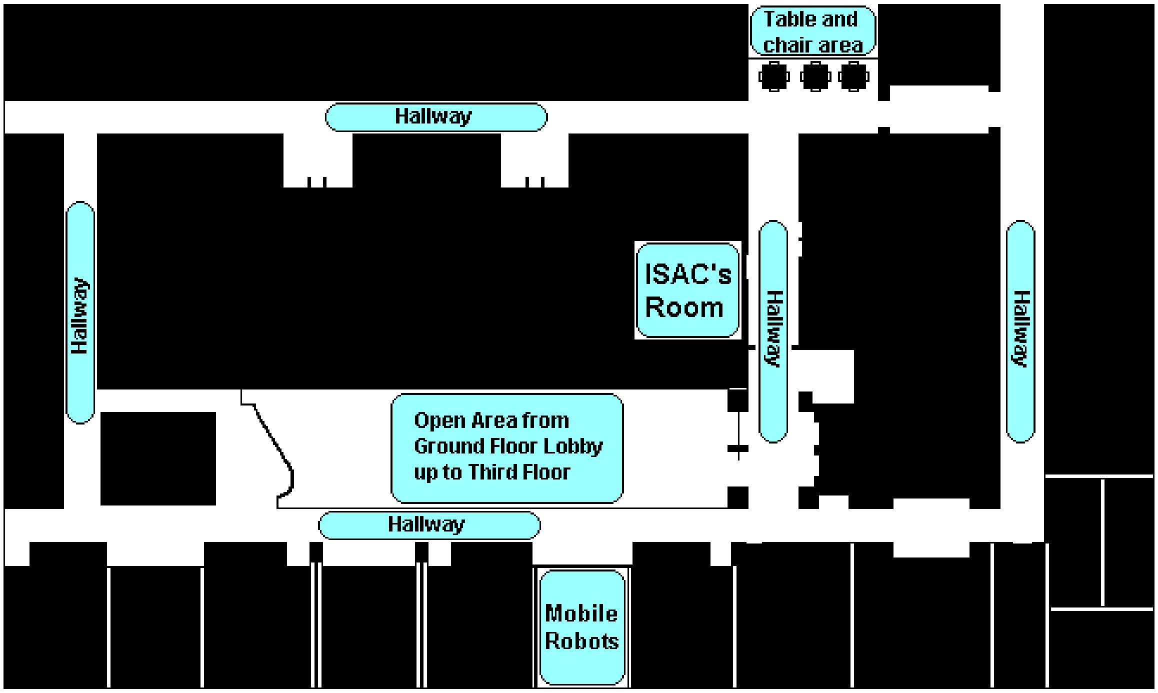


Figure 8. Two-dimensional diagram of the third floor with general areas and specific rooms that house the robots marked

This work studied how a mobile robot with a RL algorithm could learn to travel from a start position near the table and chair area (see photos in Figure 9-10) to the goal position (see photos in Figure 11-13) near ISAC's room. The robot must learn to move forward in the starting hallway until it can turn right into a hallway perpendicular to the hallway where the robot started, then travel down that hallway and turn left into the goal position. The learning algorithm for these experiments consider the goal found when three elements are satisfied: the robot is in close proximity to the goal location, the robot is oriented in the direction facing the wall opposite ISAC's room, and the laser scan matches the stored laser scan taken in the goal position with some degree of certainty. More specific definitions of these elements will be discussed later in this chapter.

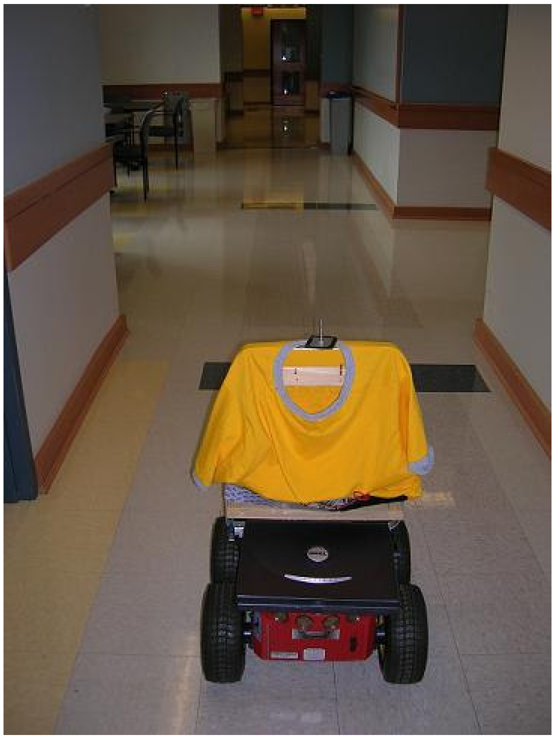


Figure 9. Photo displaying the start position from the hallway facing the direction the robot faces at the start of an episode with the robot in the start position for reference (taken by author)

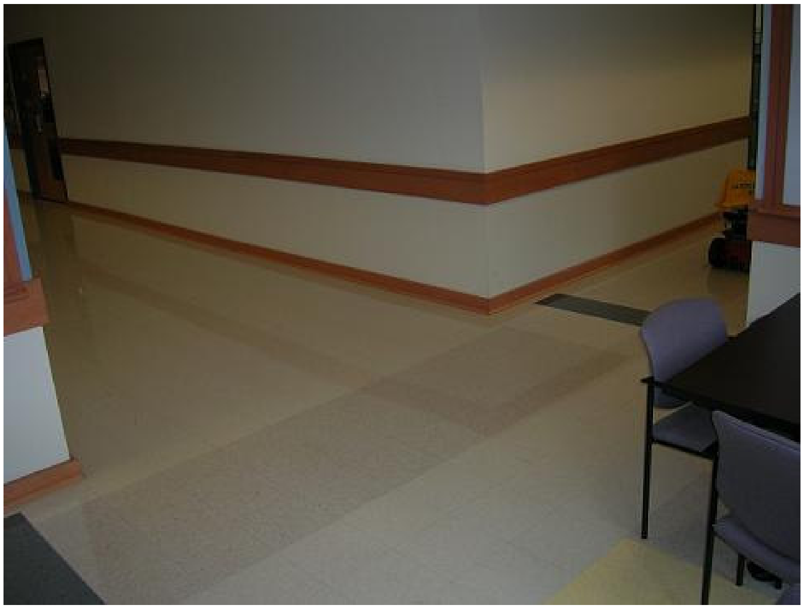


Figure 10. Photo displaying the start position from the table and chair area with the robot in the start position for reference (taken by author)

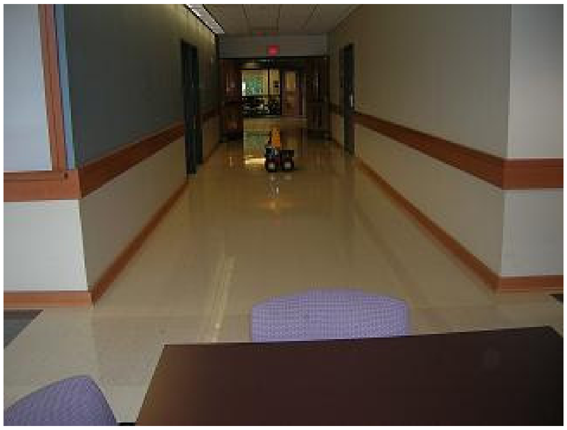


Figure 11. Photo displaying the goal position from the table and chair area with the robot in the goal position for reference (taken by author)

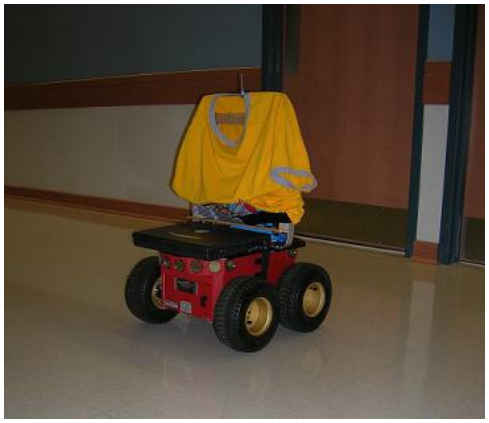


Figure 12. Photo displaying the goal position from inside ISAC's room with the robot in the goal position for reference (taken by author)

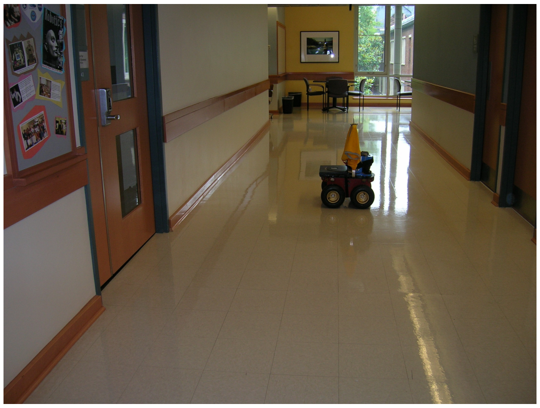


Figure 13. Photo displaying the goal position from the hallway facing the table and chair area with the robot in the goal position for reference (taken by author)

See Figure 14 for a two-dimensional illustration of the environment marked with the start position, goal position, and dimensions. The middle of the robot's base is centered on the start position at the beginning of every episode.

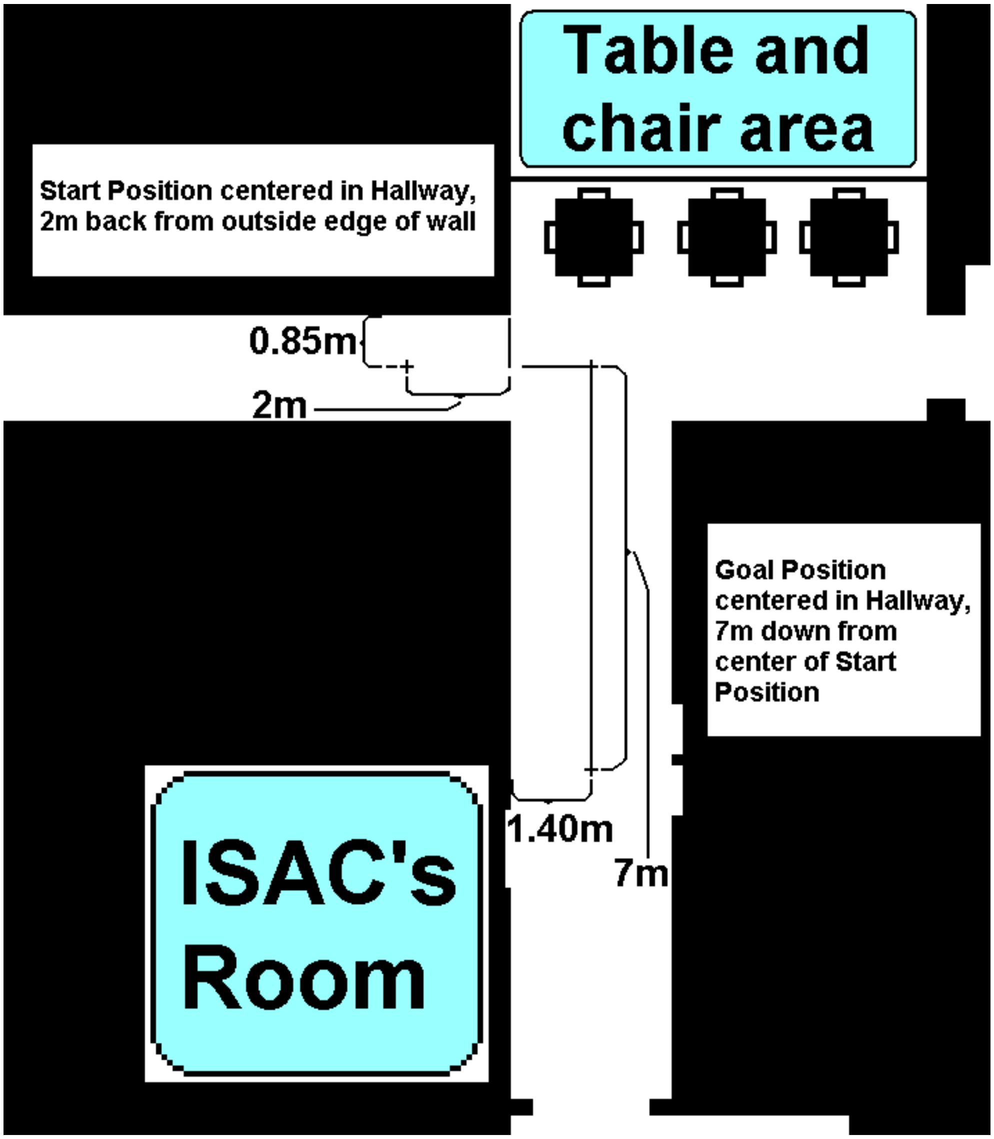


Table and chair area
Start Position centered in Hallway, 2m back from outside edge of wall
0.85m
2m
Goal Position centered in Hallway, 7m down from center of Start Position
1.40m
7m
ISAC's Room

Figure 14. Two-dimensional diagram of the environment on the third floor with measurements indicating the start position and goal position

The environment for the RL agent consists of the area marked with length and width increments in Figure 15. These measurements do not designate the limit of the available hallway area but are indicative of the absolute limits to which the robot traveled as observed during experiments. Travel outside these bounds was never observed during any of the 450 episodes conducted.

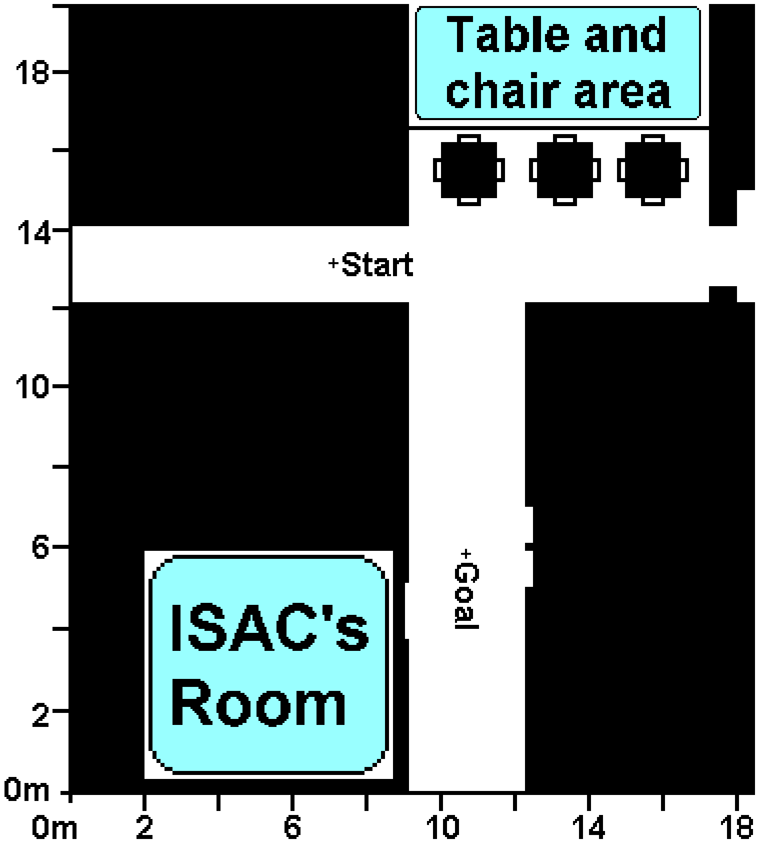


Figure 15. Two-dimensional diagram of the environment with measurements indicating the observed boundaries

Movement beyond the marked area was discouraged based on safety constraints and attraction to the goal area. Either the robot would travel into a fail state, a state in which no action but stop was possible based on safety constraints; or, the robot would find the goal position, which terminates the episode. The robot is constantly updating the Q-values to optimize the task of navigating from the start position to the goal position. Therefore, rationally, the robot would not travel far beyond the area bounded in Figure 15, because these places do not lead it to the goal it is designed to find.

This area shown in Figure 15 is the environment the robot travels in during all experiments. However, during flexibility experiments there are two obstacles placed in the environment. The obstacles were chosen as common items from the area that would block parts of the path from the start to goal position. The obstacles were placed to interfere with the path and the laser scans, which are discussed in greater detail later, during episodes of the flexibility experiments. One obstacle was a trashcan, placed along the wall the robot faces in the goal position. The other obstacle was a chair, placed in alignment with the goal position but along the wall opposite of the wall the robot faces in the goal position. See Figure 16 for a picture of the environment with obstacles for comparison to Figure 11 without obstacles. Figure 17 is a diagram of the environment used in flexibility experiments with dimensions of the obstacles.

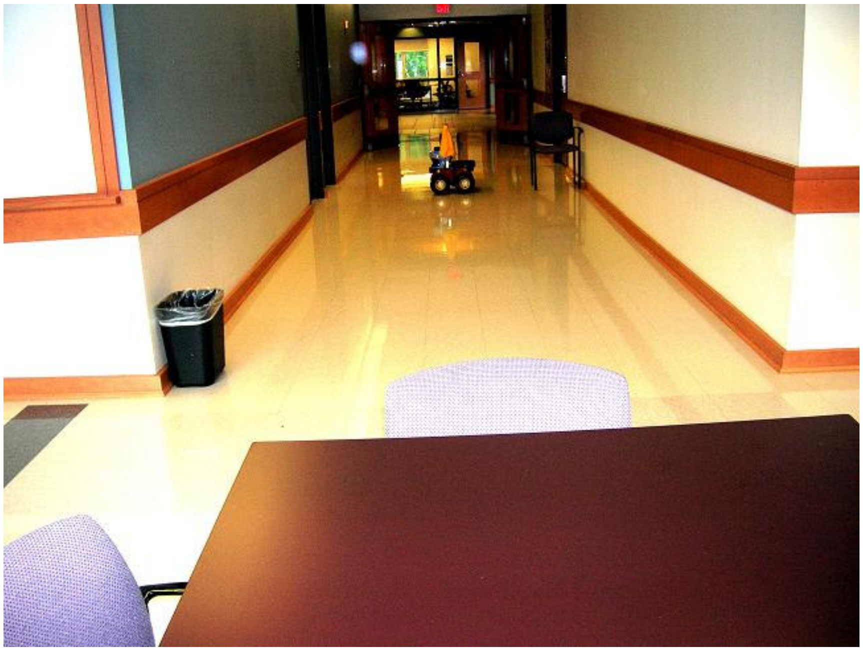


Figure 16. Photo displaying common obstacles in the environment used during flexibility experiments (taken by author)

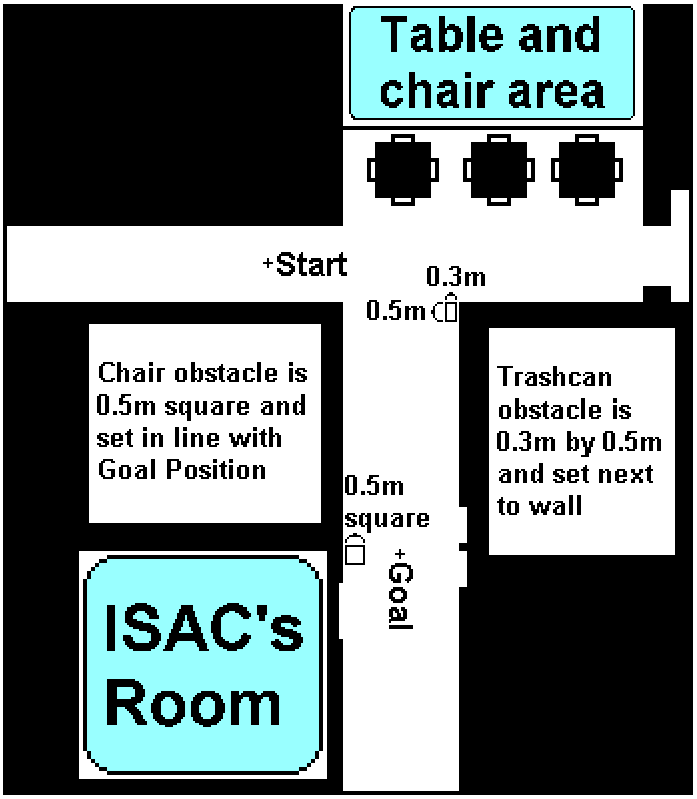


Table and chair area
+Start
0.3m
0.5m
Chair obstacle is 0.5m square and set in line with Goal Position
Trashcan obstacle is 0.3m by 0.5m and set next to wall
0.5m square
+Goal
ISAC's Room

Figure 17. Two-dimensional diagram of the environment used for flexibility experiments with obstacles and measurements marked

State-Action Pairs

Typical of most artificial intelligence tasks, once an environment is set, states and actions need to be defined. The states represent the environment to the agent (a mobile robot for these experiments) such that it can choose actions to transition between states to find the goal (Russell and Norvig, 1995). For these experiments the actions need to transition the robot from one state, s to the next, s' such that $s \neq s'$. The actions need to be big enough in order for the environment to look different to the robot at the end of an action from the start of an action. However, the actions need to be small enough so that the robot does not move beyond a key change in the environment. Actions should not change the state drastically so that the robot can evaluate the change. Defining states and actions is one area of RL problems that can be more art than science (Sutton and Barto, 1998).

The requirement for finding the right balance in state-changing actions may not seem difficult for simulated environments, but the real world is more complex. In simulated environments, such as computer chess-playing games, the state space describes the game board, which completely defines the environment. Any valid chess move is an action that will change the environment and the state description such that the state before the move, s is different from the state after the move, s' ($s \neq s'$). In real-world environments, picking actions that will automatically change the state of the environment is not guaranteed. Small movements may result in a state that looks the same as the previous one ($s = s'$), but larger actions may skip over key transition areas. For experiments in this work, actions are defined that transition the robot into a new state with each movement of the wheels. See Table 1 for a list of the actions, the forward and turn speed values for the actions, and the resulting movement.

Table 1. 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
Turn Right	0 mm/sec	-10 °/sec	45° right turn
Turn Left	0 mm/sec	10 °/sec	45° left turn

The selection of these actions strikes a balance between actions that are too small and ones that are too large. Given the dimensions of the environment (roughly 18 meters by 20 meters) and the characteristics of navigation task (forward and turning movements), the actions were chosen to allow the robot to move at each step of the task into an area that looked different than the previous one based on the state definition. The explanation of the state definition also contributes to a better understanding as to how these actions regularly transition the robot from state s to state s' during an episode.

In simulated environments such as the above-mentioned chess game, state representation could describe the board with position information about each piece and perhaps an account of the last move. This state representation completely describes all relevant information about the environment; thus, satisfying the Markov property, which is desirable in RL problems. In a real-world environment, packaging all details about the environment for the agent into a state representation may not even be possible because of the complexity of the environment. Any element not described in the state representation is considered hidden state information, because it is hidden from the agent. RL problems settle for assuming the Markov property is satisfied if the states represent enough information for the agent to make informed decisions. This assumption is not exclusive to RL techniques but can also be found in many methods to artificial intelligence (Sutton and Barto, 1998; Russell and Norvig, 1995).

Therefore, the state definition used in these experiments must include enough information about the environment for the robot to make informed decisions when choosing actions. The states need to describe the environment with enough detail such that the agent (robot) can make an informed decision about the action to take in the state. This definition strives to satisfy the Markov property while balancing between this property and the curse of dimensionality. If too many elements about the environment are used to describe a state, than the state space can become too large to effectively search it during RL episodes.

In these experiments, several elements are used to define the states. The previous action taken is included. The robot's internal sensors report odometry and orientation information, which are used to define the state of the environment. An external sensor, a laser range finder is used to determine areas that are clear to move into and ones that are blocked, which are used to define the states. Also, the data from the laser is used to compare the current laser readings to the scan of readings taken in the goal position. Lastly, the environment is divided into regions that form part of the state definition. Ultimately, ten Boolean features make up the state representation. Furthermore, these ten features correspond to bits which represent information about the environment. See Table 2 for a condensed description of the bits and what they represent for a quick reference guide.

Table 2. Delineation of State Representation into Binary Bits and their Meaning

Bit	Indicates	Boolean Values
Most Significant Bit	Comparison of current laser scan to stored scan at goal position (1 bit)	0 – Does not Match 1 – Matches
. .	Current action (2 bits)	00 – Stop 01 – Forward 10 – Turn Right 11 – Turn Left
.	Right side Clear (1 bit)	0 – Obstacle, Blocked 1 – No obstacle, Clear
.	Front section Clear (1 bit)	0 – Obstacle, Blocked 1 – No obstacle, Clear
.	Left side Clear (1 bit)	0 – Obstacle, Blocked 1 – No obstacle, Clear
. . .	Orientation of the robot in a Cardinal or Ordinal Direction (3 bits)	000 – North 001 – South 010 – East 011 – West 100 – Northeast 101 – Southeast 110 – Northwest 111 – Southwest
Least Significant Bit	Region (1 bit)	0 – Not Goal Region 1 – Goal Region

The most significant bit indicates whether or not the current laser readings match the stored readings from the goal position within a degree of certainty. The robot is equipped with a SICK Laser Measurement System (LMS) – model LMS 200 (SICK, 2005). The SICK LMS 200, discussed in the Hardware section of this chapter, takes a scan of the environment and returns 180 measurements. These measurements correspond to 180 angles, uniformly distributed on the half-circle scan. Each reading returns the distance in millimeters to the nearest obstacle from that angle. The stored measurements from the goal position are compared to the current measurements from the laser during an episode of the experiments. For all 180 readings, if the current reading is within 300mm of the stored reading, than that measurement counts towards the total percentage of the scan that matches the stored scan. The most significant bit in the state representation becomes true if more than 10% of the current scan matches within 300mm of the scan stored from the goal position. This method of matching the scans has an advantage of processing the information quickly. Observations found the method to be reliable in matching the scan saved in the goal position to similar areas in the environment. Matching the laser scan perfectly to the goal scan was not necessary for finding the goal, because finding the goal relies on satisfying three factors, only one of which deals with matching the laser scan. The 10-bit binary representation of the state with a matching laser scan and all other bits false would be written as 1000000000, which converts to the decimal number 512. Likewise, a state with all bits false, including the most significant bit signifying if the laser scan matches the goal scan, would be written 0000000000, which is 0 in decimal form. More examples of the state representation will be given as the individual bits are explained.

The next two bits indicate the action taken to transition the robot from the previous state into its current state. See Table 3 for a list of the actions and how they are represented in binary and decimal form.

Table 3. Actions in Binary and Decimal Form

Action	Binary Representation	Decimal Conversion
Stop	00	0
Forward	01	1
Turn Right	10	2
Turn Left	11	3

In the initial state for all episodes, the action is stop. If the agent chooses to move forward from the initial state, s the action bits would change from 00 to 01 in the next state, s'. For example, if the initial state, s had all other bits true, the action bits would signify the stop action, and the state representation in binary would be 1001111111, which is 639 in decimal form. If the robot chooses to move forward and all other bits remain true, the new state s' would be 101111111, or 767.

Data from the laser scan is used in the definition for the next three bits, each indicating a section of the scan as clear (1) or blocked (0). The laser range finder returns straight-line distances from each angle and the x and y components of each distance. The first of these bits reports on obstacles to the right side of the laser. The laser is attached and centered on the front of the robot, so this bit equally describes obstacles to the right of the robot. The right side is defined by straight-line readings between angles 18 and 112. If at least one laser scan reading between 18º and 112º is less than 1000mm, then the bit is false. The last of these bits reports on obstacles on the left side of the robot. If at least one straight-line laser scan reading between 68º and 162º is less than 1000mm, then the bit is false. The middle of these three bits represents obstacles in front of the laser. This bit is determined using the x components of the laser scan data. If at least one laser scan reading between angles 68º and 112º is less than 1450mm, then the bit is false. For example, a state with an obstacle within 1000mm blocking the right and left side and all other bits false would be written 0000100000, which is 32 in decimal form. In state 32 only the front section is clear. Furthermore, a state with all sides clear and all other bits false would be 0001110000, or 112.

The definitions for the right, front, and left side of the laser were designed based on the actions, the robot's dimensions, and the position of the laser range finder. The left and right motions are 45º turns, so the right and left sections had to at least be 45º spans from the leftmost and rightmost edges of the robot. For a right turn, the robot's front section (112º to 68º) and rightmost side (68º) had to move freely through at least 45º to its right (23º); moreover, 5º of leeway were added on to make the final span 112º to 18º. Likewise, a left turn required the front (68º to 112º) through leftmost side (112º) to move through at least 45º of clear room to the left (157º), and adding 5º of leeway made the final front and left span 68º to 162º. For the forward action the front span of the robot had to be clear so that it could move forward without hitting an obstacle. This section is defined as the span of laser readings from 68º to 112º, and the x-components of the laser readings at these angles were used to determine if obstacles were in front of the robot. The three sections with the laser sensor and robot outline are illustrated in Figure 18. The front side of the Segway measures 0.6m; similarly, the distance across Skeeter's front including the wheels that extend out from the main body measure 0.56m. The laser sat roughly 0.1m forward from the center of the base on both robots. Therefore, initial tests with the Segway could be used to design the safety thresholds used in the experiments on Skeeter.

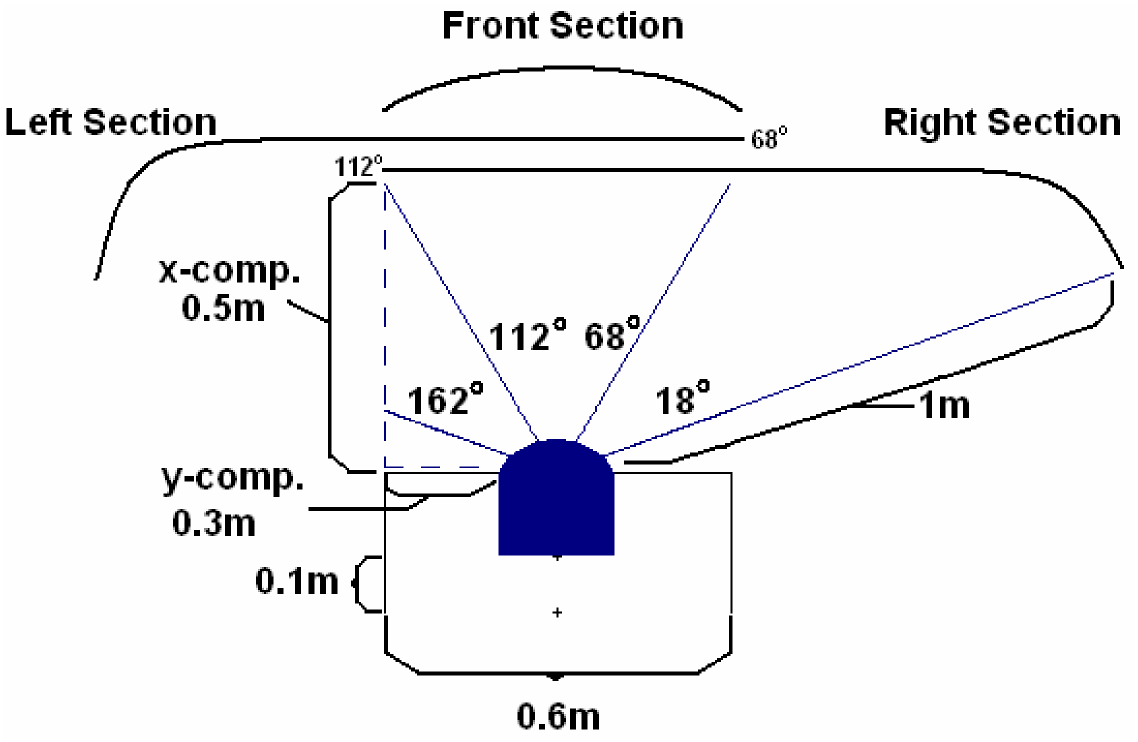


Figure 18. Diagram of the right, left, and front sections of the laser scan used to define three bits in the state representation

These three bits and the thresholds that define them also serve to uniquely describe different areas of the environment as distinguishable states and provide a ribbon of safety between the robot and its surroundings. The thresholds for detecting an obstacle were designed based on the distance needed to complete an action in that direction and added safety. Initially, the thresholds were based on Segway's safety pipes (Figure 7) plus any distance traveled in the action itself (e.g. forward distance or turning radius). These thresholds were tested on Skeeter and found to be efficient on both conditions of distinctively describing areas of the environment and preventing collisions with obstacles. Ultimately, the successful forward threshold was composed of:

$$900\text{mm} + 500\text{mm} + 50\text{mm} = 1450\text{mm}, \qquad (18)$$

where the first measurement is for safety, the second for the forward movement, and the third for the possible overshoot in the movement. The Segway robot with the necessary safety pipes creates a larger turning radius than Skeeter; however, the left and right thresholds were applicable for breaking up the environment into unique states. The learning algorithm maps one optimal action to each state. Therefore, the environment needs to be described in states so that different actions can be taken in different areas. The 1000mm threshold on the bits that describe an obstacle-free area to the right or left serves to describe different areas of the environment as different states. For example, the robot starts each episode in a hallway. This area of the environment needs to be described by a different state than a more open area so that different actions can be mapped to these states. The initial state might be represented by the following state in binary, 0000100100 and decimal, 36 form. Only the front section is clear; therefore, the robot should choose to move forward safely. These thresholds do not want to restrict the robot so much that it cannot explore most areas of the environment, but they are designed to give the states some uniqueness and some account as to the possible safe actions available. The thresholds create a balance between allowing the robot to explore the environment without colliding into an obstacle while uniquely describing different areas of the environment, which may restrict the available actions that can be executed safely.

The three bits next to the least significant bit describe the direction that the front of the robot is facing. Cardinal and ordinal directions were used to establish eight areas, which help define unique states in the environment. The Player software, discussed later in this chapter, was used to gather data from the robot. The robot's internal sensors report its pose, position and orientation, in the form of an x-position, y-position, and theta. The angle reported in theta was used to determine the direction the robot was facing. See Figure 19 for a diagram of the orientation angles and odometry directions with respect to the front of the robot.

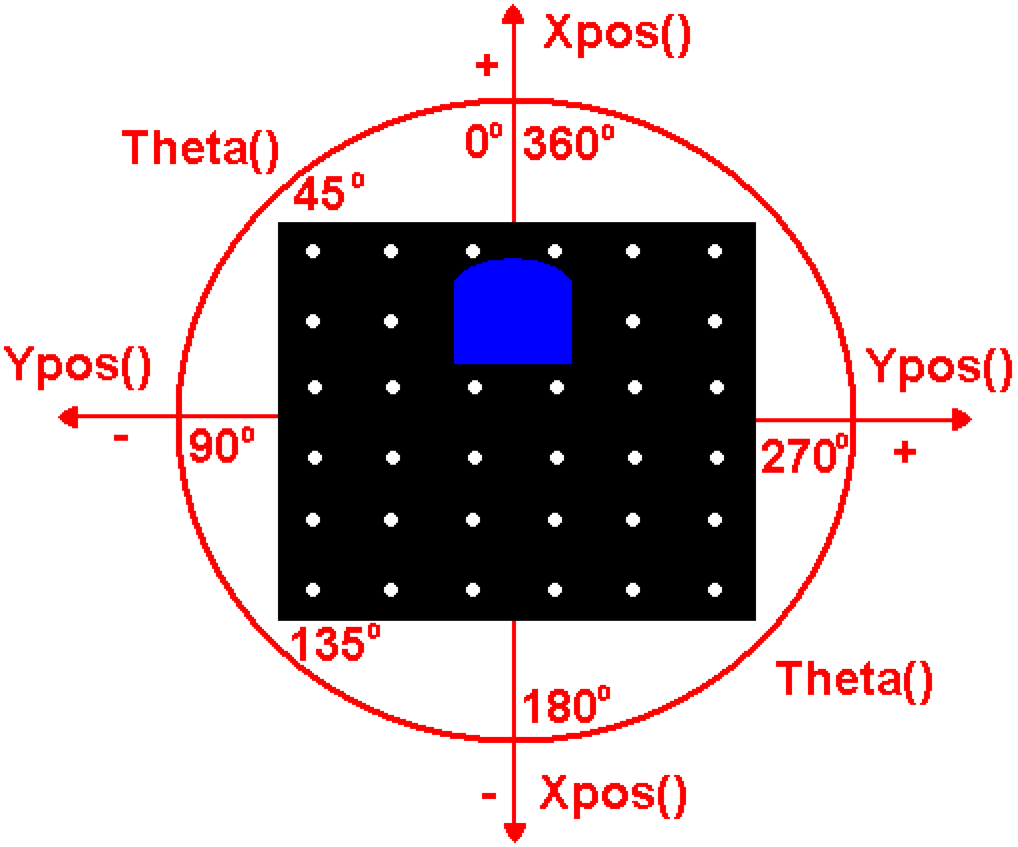


Figure 19. Diagram of a robot with a blue SICK laser outline indicating the front end with an overlay of how odometry and orientation readings are arranged

Segway and Skeeter use the same scheme for reporting pose information. In the start position, the robot reports theta equal to 0º. Based on a map of the campus the relative position of the robot in the start position was east, so a heading of 0º was set to east. If the robot chose to turn right from the start position, a turn angle would be calculated as the robot moved until it had turned 45º. Theta at the end of a turn right action from the start position would be around 315º, which corresponds to a southeast direction. If the robot was facing southeast and all other bits were false, the state would be 0000001010 in binary and 10 in decimal form. See Table 2 for a list of all cardinal and ordinal directions and their representation in binary bits.

Accommodations were made to handle the switch between 0º and 360º on the orientation circle. Ultimately, the eight orientation sections meant east corresponded to any theta between 22.5º, exclusive, and 0º, inclusive, as well as any theta between 337.5º, inclusive, and 360º, inclusive. Following clockwise southeast consisted of theta values between 337.5º, exclusive, and 292.5º, inclusive. South fell between 292.5º, exclusive, and 247.5º, inclusive. Southwest was accessed as being between 247.5º, exclusive, and 202.5º, inclusive. Likewise, west was defined between 202.5º, exclusive, and 157.5º, inclusive. Northwest was comprised of theta values between 157.5º, exclusive, and 112.5º, inclusive. North rested between 112.5º, exclusive, and 67.5º, inclusive. The eighth section defined northeast between 67.5º, exclusive, and 22.5º, inclusive.

The least significant bit represents whether or not the robot is in close proximity of the goal position, based on odometry. The environment is broken up into two regions, the goal region and any location not in the goal region. Odometry readings in the x and y direction (see Figure 19) are used to compare whether or not the robot is within the goal region. Thresholds of 3m in either direction set the boundaries of the region. See Figure 20 for a diagram of the environment with the goal region marked.

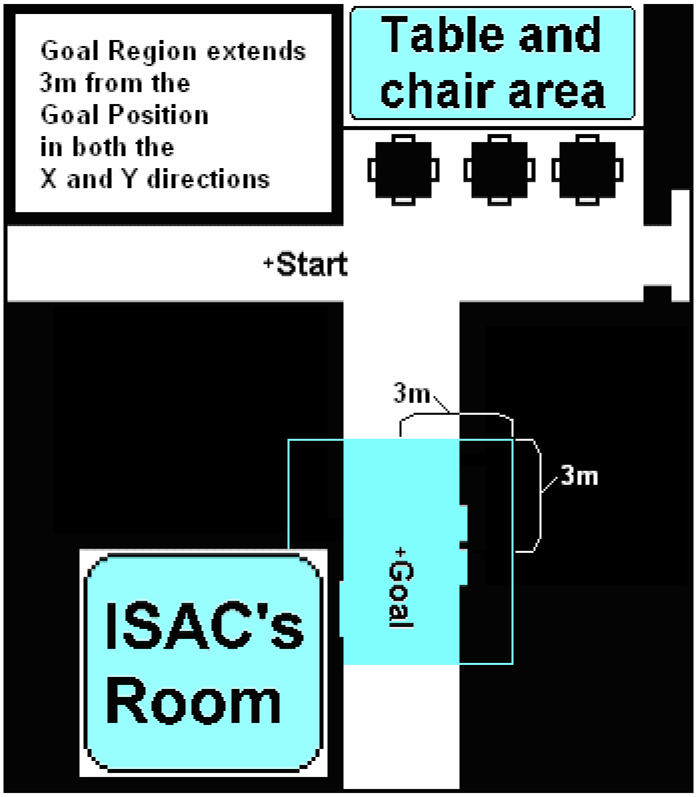


Figure 20. Diagram of the environment with the goal region outlined

If this bit is true the robot is within the goal region, but if this bit is false the robot is somewhere other than this region. In these experiments, the robot begins in the start position with readings (0mm, 0mm, 0º), which signify (odometry reading in the x direction, odometry reading in the y direction, orientation) or to coincide with Figure 19 (Xpos(), Ypos(), Theta()). Calculations are made during episodes of the experiments to determine how far the robot has traveled in both the x and y direction. These distances are compared with stored information about the goal position to determine if the robot is within the bounds of the goal region around the goal position. For example, if the robot is in the goal region but all other bits are false, that state would be written 0000000001 or 1 in decimal form.

The ten binary bits mean there are $2^{10} = 1024$ states that make up the state space, S, for the environment. Four actions make up the set of all actions, $A(s)$, for the state space. Therefore, $Q(s,a)$, which holds values for all state-action pairs, contains 4096 values. The 4096 state-action pairs represent the full set of all combinations that need to be tested to exhaustively evaluate the RL problem. However, full exhaustive searches, evaluating every action for every state, are not often used in RL, nor are they required in order to evaluate the state-action space for finding an optimal path to the goal. For example, some actions selected in the real world would result in the robot colliding with a wall. Restricting such actions does not diminish evaluation of the RL problem, instead it aides in quicker learning by reducing the state-action space that needs to be evaluated.

Researchers try to reduce the search space if possible in order to quicken learning (Huber and Grupen, 1999). Reducing the state space in this work to avoid collisions is beneficial from a search aspect as well as for safety reasons. If the robot's laser sensor detects an obstacle in the front section and the policy chooses the forward action, then the robot would travel forward and most likely collide with the obstacle. Consider state 36, which in binary equals 0000100100. In this state, the left and right sides are blocked. More specifically, at least one laser reading between 68° and 162° measures less than 1000mm, and at least one laser reading between 18° and 112° measures less than 100mm. The front section is clear, which means all laser readings between 68° and 112° measure 1450mm or more. The possible choices of action include 0 (stop), 1 (move forward), 2 (turn right), and 3 (turn left) (See Table 3); which correspond to evaluating Q(36,0), Q(36,1), Q(36,2), or Q(36,3). Choosing action 2 (right turn) or action 2 (left turn) may result in a collision with an obstacle in the environment, such as a wall. Therefore, in order to prevent such collisions, any state-action pair with a blocked region that chooses an action in the direction of a blocked region is not allowed. An added bonus is this safety precaution reduces the state-action space. Out of the 4096 state-action pairs, 1536 involve states that report an obstacle in the direction of the action. Therefore, restricting these state-action pairs results in a 37.5% reduction of the state-action space.

Learning Algorithm

The learning algorithm used involved a reward function and policy. Reinforcement learning algorithms always incorporate some type of reward function, which can be either sparse or dense. When using a sparse reward function, the environment returns reward with a value of 0 for most states and perhaps a positive reward of 1 in the goal states. Dense reward functions use mostly non-zero values. In this thesis, a dense reward function is used in order to make the robot more knowledgeable about the optimization task. The policy chooses actions in order to maximize the reward function. In this thesis a supervisor is included into the policy along with random choices and choices based on the best evaluation in the Q-table.

The reward function returns 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 changes the value of the state-action pair that transitions the robot into the next state and is based on the category of this state. Ultimately, it is desirable for an action to transition the robot from one state to a goal state. Therefore, the reward is sent to commend the state-action pair that moved the robot into a goal state. This relationship is represented in equation 14. See Figure 21 for the categories of states and their associated reward function.

Reward Function
r(goal) = 2000
r(fail) = -(nmax*Tstep) = -(256*1.75) = -448
r(other) = -(n*Tstep) = {-1.75, -3.5, -5.25...}

Terms
n – step number
n_{max} – maximum steps before terminal state
T_{step} – represents uniform time between steps

Figure 21. Reward function equations and terms

Whether or not the goal is found is not directly represented in the state definition for a connection to the reward. An action transitions the robot into a new state, which is evaluated for a reward. Only then does the algorithm check if the goal has been found, based on three criterions: current laser scan must match stored laser scan from the goal position, robot must be in the goal region, and robot must be in the East direction. Therefore, the goal could be found in and represented by more than one state definition, with bit changes such as different actions transitioning the robot into a goal state. If all criteria for a goal state are satisfied, then the reward for the goal is given for the state-action pair.

The reward for a goal state is set at a highly positive value to counter any accumulated negative reward during an episode of the task. A fail state is one in which the robot cannot move to the right, left, or forward based on the safety constraints. If the robot transitions into a fail state, interaction with the environment ends the episode.

Therefore, this state returns a significant negative reward to condemn the state-action pair that moved the robot into a fail state. The design of the reward for a fail state is based on the specific task but remains flexible if the task should change. The n_{max} term is the maximum number of steps allowed before an episode is terminated. Basically, if 256 steps are taken in this task without a goal or fail state found, the episode has gone on long enough, and the robot should begin a new episode. For this navigation task, 256 steps are well enough steps to find a terminal state. However, if the task should change to be longer or shorter, the n_{max} term could easily be changed to reflect the different task length while maintaining the current representation in the fail state reward. The reward for all other states is slightly negative based on a set term T_{step}. This term represents time in the reward function without explicitly doing so. No extra calculations are taken for each state visited along an episode to determine the time from the start of an episode to that state. Instead, T_{step} represents a uniform time constant. For the actions available in this work, at least 1.75seconds passes for each step of an episode. Therefore, the reward function for states other than a fail or goal indicates the robot is taking too much time to find the goal. Ultimately, the robot is trying to find the optimal path from the start position to the goal position based on time.

In order to find the goal in the shortest amount of time; thus, maximizing the reward signal, the policy must develop optimal state-action pairs to transition the robot from the start position to the goal position. In reinforcement learning, the policy trades off between exploration and exploitation. The robot exploits the information it knows about the visited states while exploring new states to reevaluate what it knows and eventually learn the most optimal path through the states to the goal. In this work, suggested actions for a Supervisor Table and random actions are used to explore the environment and the Q-table is relied upon to exploit what the robot has learned thus far in the episode.

The Supervisor Table is a list with 1024 entries, each corresponding to the states of the environment used in this work. The values listed represent the action the Supervisor suggests as the best action to take in that state. See Table 3 for the correspondence between actions and their representative numbers. The Supervisor Table is initialized with zeros. Then a systematic population of the table takes place based on a hierarchy of actions and specifics about the task. The hierarchy makes forward actions most desirable, right turn actions the next desirable movement, and left turn actions least desirable. For states with bits that indicate a forward action is possible (i.e. no obstacles in the front section of the robot), the action is set to forward. For states with bits that indicate there are obstacles to the left and front of the robot, the Supervisor Table lists a value for the turn right action. This systematic population continues to follow the hierarchy for all combinations of obstacle bits. Another systematic assignment of states to actions takes place for states that indicate the robot is in the goal region. A preference is established that the robot should turn left in the goal region to face the correct East direction for the goal state. Lastly, a remote-controlled run through the task from the start position to the goal position influences the Supervisor Table. A human, specifically, the author, guides the robot through the task, during which, the states and actions are recorded in the Supervisor Table. The Supervisor Table is considered an optimal policy for this navigation task. Using this table alone in this specific task, an agent would find the goal with probability one. Other research has shown advanced controllers, such as the Supervisor Table in these experiments, to improve and quicken learning (Smart and Kaelbling, 2002; Martínez-Marín and Duckett, 2005).

In the policy, suggested actions from the Supervisor Table were selected with a probability of 50%. The other half of the time, actions were divided up between random selection and actions with the highest Q-value in the Q-table for the particular state according to an ε–greedy method. At the beginning of an episode, the robot knows nothing about the task, so the policy should choose more random actions. As the number of episodes increases over a trial, the robot has a better evaluation of the values in the Q-table. Therefore, the robot can rely more on the Q-table to choose actions near the end of a trial of several episodes. The variable governing the balance between choosing random actions and actions from the Q-table is called epsilon, ε. This variable decreases based on the total number of episodes in a trial and the current episode according to the following equation:

$$\varepsilon = \varepsilon_{start} * (1.0 - (episode_{current} \div episode_{total})) \, , \tag{19}$$

where $\varepsilon_{start} = 0.9$ and $episode_{total} = 15$ for these experiments. See Table 4 for the linear progression of ε over the 15 episodes used in all experiments in this work. The random selection between the four available actions is divided uniformly and based on a random number generated using the computer’s clock. Given a state, the greedy portion of the policy compares the Q-values of all the state-action pairs in that state and returns the action associated with the highest value. This method is termed greedy because it selects based on the highest value only – no other measure.

Table 4. Progression of epsilon, ε, over all episodes

$episode_{current}$	ε
0	.90
1	.84
2	.78
3	.72
4	.66
5	.60
6	.54
7	.48
8	.42
9	.36
10	.30
11	.24
12	.18
13	.12
14	.06

No matter how the action is chosen the Q-values are updated to evaluate the state-action pair; thus, learning progresses. Therefore, the Q-values are evaluated over each episode until the end of the trial, each episode allowing the agent to evaluate and improve the policy closer and closer to the optimal policy for finding the goal. The analysis of how well this learning algorithm evaluated the task is presented and discussed in Chapter IV.

System Overview

Hardware

These experiments were carried out on a Pioneer 2-AT (all terrain) robot named Skeeter, pictured from the front in Figure 22 (left). As depicted, the robot is equipped with a SICK laser range finder that reports distance readings from the laser to obstacles across a 180 degree span.

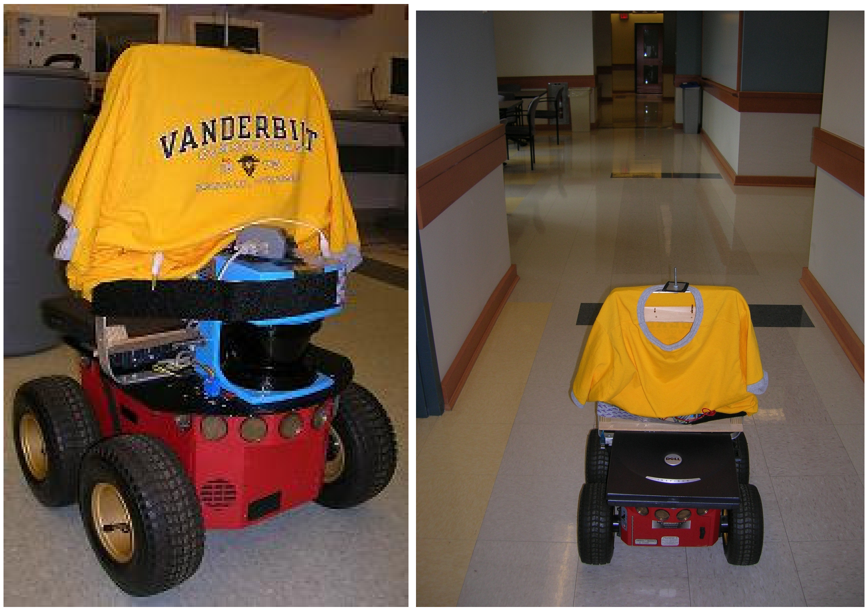


Figure 22. Skeeter from front (left) and back (right)

The laser was powered using two rechargeable sealed lead batteries. The laser range finder sensor connects to a laptop computer through an RS422 serial interface. The RS422 connection is a high-speed connection that requires a special serial-to-USB converter. The EasySync RS422 converter is used in the hardware configuration for these experiments. Two Dell laptops were needed to relay commands and data during the experiments. One, a Dell Inspiron 1150 was the control (not pictured), sent commands to the laptop that road along Skeeter during experiments, pictured from the back in Figure 22 (right). Both systems run the Gentoo Linux operating system, which is needed for the specific software.

Software

Player/Stage software, developed at the University of Southern California, runs only in Linux and provides us with a straightforward way to connect to the robot and its sensors (Gerkey et al., 2003). Stage is a two-dimensional simulation package that was not used in these real-world experiments. Player, a robot server, contains configuration files for many commonly used robots and sensors and acts as a link to the hardware (i.e. SICK laser range finder and robot) for sending commands to the motors and returning data from the range finder.

Additionally, NDDS (Network Data Distribution Service) was incorporated into the communication pathway in these experiments (RTI, 2004). See Figure 23 for a diagram of the pathway.

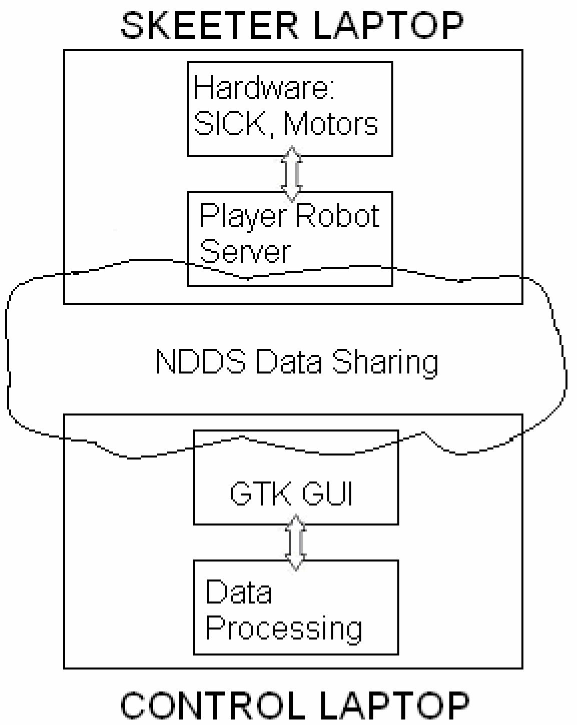
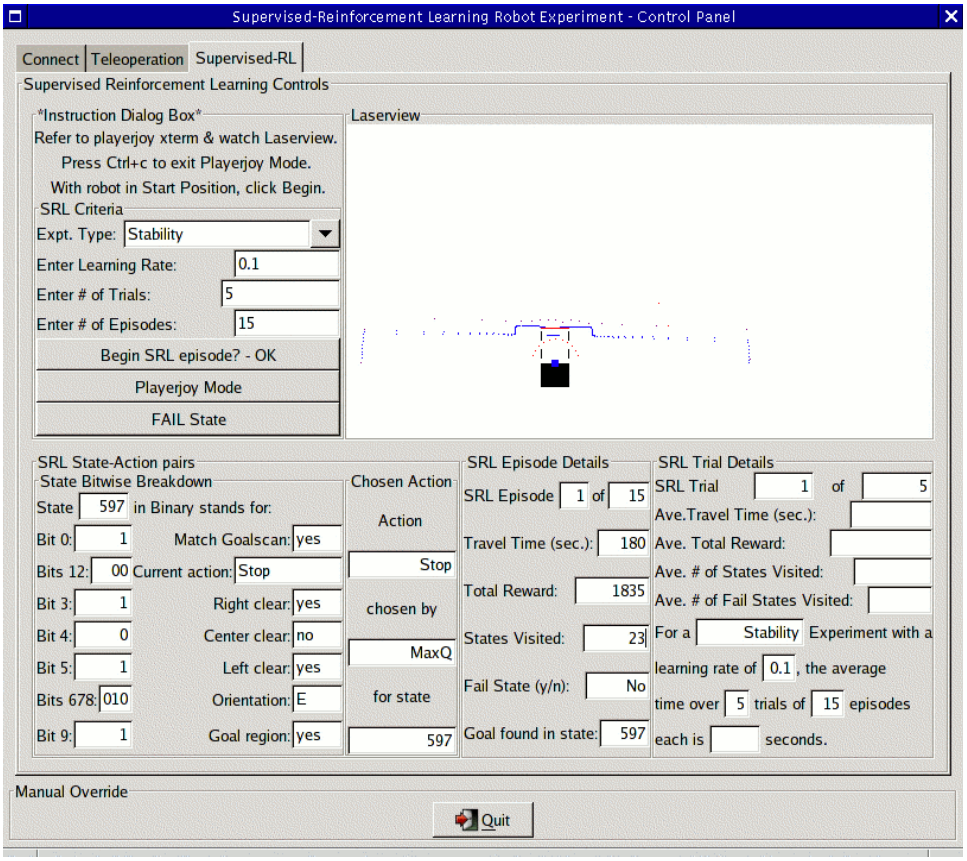


Figure 23. Diagram of the software pathway for communication between laptops

NDDS is a communication protocol developed by Real-Time Innovations for real-time data sharing. Any variables that need to be passed between the laptops are managed using publish and subscribe commands from NDDS. GTK was also used to create a user-friendly GUI for these experiments. With the GTK libraries, one can create interfaces with buttons, interactive text boxes, and even image windows for data feedback. See Figure 24 for one of tabs of the GUI used in these experiments.



Supervised-Reinforcement Learning Robot Experiment - Control Panel

Supervised Reinforcement Learning Controls

Instruction Dialog Box
Refer to playerjoy xterm & watch Laserview.
Press Ctrl+c to exit Playerjoy Mode.
With robot in Start Position, click Begin.

SRL Criteria
Expt. Type: Stability
Enter Learning Rate: 0.1
Enter # of Trials: 5
Enter # of Episodes: 15

SRL State-Action pairs
State Bitwise Breakdown
State 597 in Binary stands for:
Bit 0: 1 Match Goalscan: yes
Bits 12: 00 Current action: Stop
Bit 3: 1 Right clear: yes
Bit 4: 0 Center clear: no
Bit 5: 1 Left clear: yes
Bits 678: 010 Orientation: E
Bit 9: 1 Goal region: yes

Chosen Action
Action Stop chosen by MaxQ for state 597

SRL Episode Details
SRL Episode 1 of 15
Travel Time (sec.): 180
Total Reward: 1835
States Visited: 23
Fail State (y/n): No
Goal found in state: 597

SRL Trial Details
SRL Trial 1 of 5
Ave.Travel Time (sec.):
Ave. Total Reward:
Ave. # of States Visited:
Ave. # of Fail States Visited:
For a Stability Experiment with a learning rate of 0.1, the average time over 5 trials of 15 episodes each is seconds.

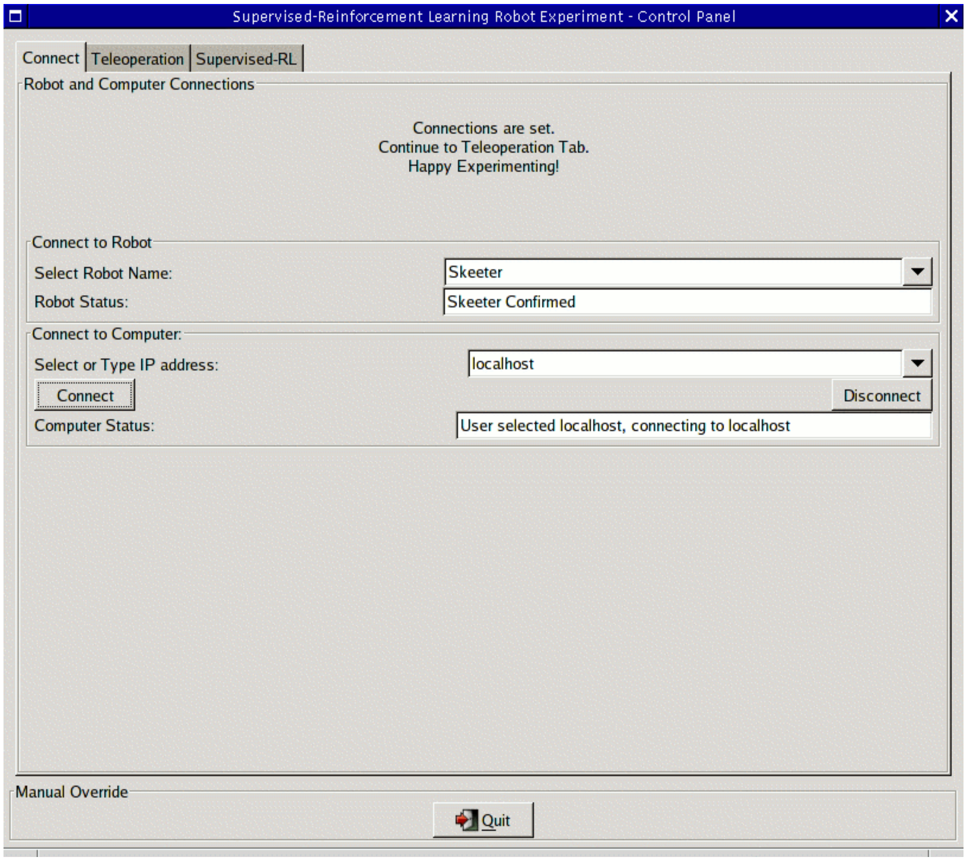
Manual Override

Figure 24. GTK based GUI with buttons, text boxes, and display window

The upper right corner of the tab displays a window with an outline of the robot and SICK laser range finder and plots of distance data from the laser range finder that is sent from the Skeeter laptop through the communication pathway. All these pieces of software are used to remain consistent with the trends in the CIS lab.

Experiment Procedure

Six experiments were conducted in this work, using two environments and testing three values of the learning rate in each environment. Each experiment consisted of 5 trials of a set of fifteen episodes. As previously mentioned, an episode consists of a run from the start position to a terminal state, either a goal, fail, or maximum number of steps, whichever comes first. The procedure involved a series of steps using the GUI designed for these experiments. The GUI consists of three tabs. The first prepared the communication link between the two laptops (see Figure 25).



Connections are set.
Continue to Teleoperation Tab.
Happy Experimenting!

Select Robot Name: Skeeter

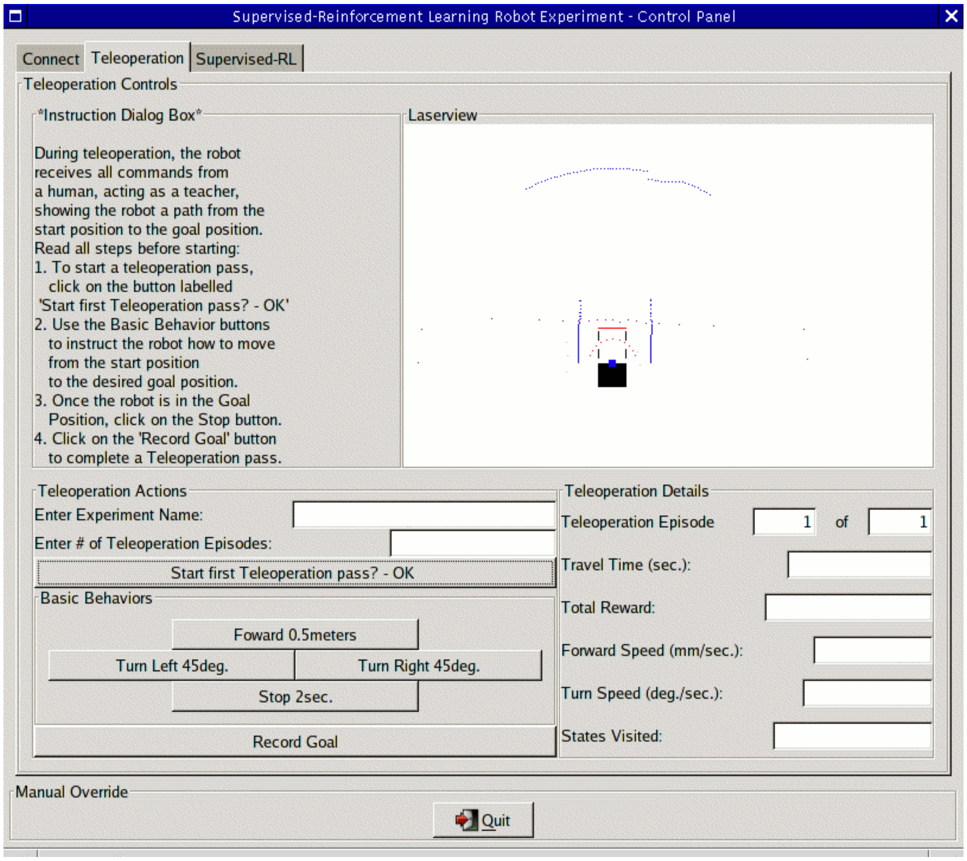
Robot Status: Skeeter Confirmed

Select or Type IP address: localhost

Computer Status: User selected localhost, connecting to localhost

Figure 25. Connect tab on the GTK GUI

The Skeeter robot was selected to connect to and the IP address of the laptop was entered. Once communication between the laptops was established, commands could be sent to the robot and data information received. The second tab allowed for the remote-controlled run through the task so that the actions taken during the run were saved in the Supervisor Table (see Figure 26).



During teleoperation, the robot
receives all commands from
a human, acting as a teacher,
showing the robot a path from the
start position to the goal position.
Read all steps before starting:

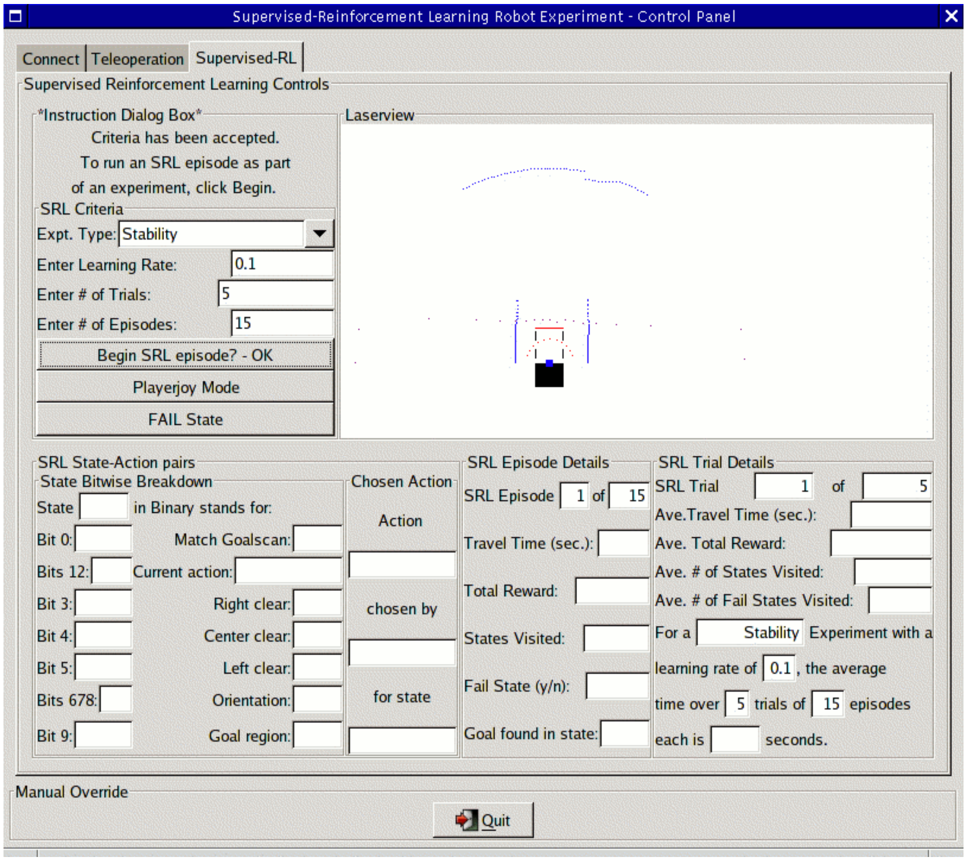
1. To start a teleoperation pass,
   click on the button labelled
   'Start first Teleoperation pass? - OK'
2. Use the Basic Behavior buttons
   to instruct the robot how to move
   from the start position
   to the desired goal position.
3. Once the robot is in the Goal
   Position, click on the Stop button.
4. Click on the 'Record Goal' button
   to complete a Teleoperation pass.

Teleoperation Episode 1 of 1

Figure 26. Teleoperation tab on the GTK GUI

This tab has labels that use the word "teleoperation." For clarity, the human-guided run through the task was remote-controlled and not strictly teleoperation, as defined by Murphy (2000). Teleoperation means the operator is only using an interface to guide the device. In these experiments, the operator had access to more information than data displayed on the GUI, so it was more like driving a remote-controlled car to select the actions to complete the task.

The third tab was used in the bulk of these experiments (see Figure 27.) In this tab interactive drop-down menus and text boxes are used to set the parameters of the experiment.



Supervised-Reinforcement Learning Robot Experiment - Control Panel

Connect | Teleoperation | Supervised-RL

Supervised Reinforcement Learning Controls

Instruction Dialog Box

Criteria has been accepted.
To run an SRL episode as part
of an experiment, click Begin.

SRL Criteria

Expt. Type: Stability

Enter Learning Rate: 0.1

Enter # of Trials: 5

Enter # of Episodes: 15

Begin SRL episode? - OK

Playerjoy Mode

FAIL State

Laserview

SRL State-Action pairs

State Bitwise Breakdown

State in Binary stands for:

Bit 0: Match Goalscan:

Bits 12: Current action:

Bit 3: Right clear:

Bit 4: Center clear:

Bit 5: Left clear:

Bits 678: Orientation:

Bit 9: Goal region:

Chosen Action

Action

chosen by

for state

SRL Episode Details

SRL Episode 1 of 15

Travel Time (sec.):

Total Reward:

States Visited:

Fail State (y/n):

Goal found in state:

SRL Trial Details

SRL Trial 1 of 5

Ave.Travel Time (sec.):

Ave. Total Reward:

Ave. # of States Visited:

Ave. # of Fail States Visited:

For a Stability Experiment with a learning rate of 0.1, the average time over 5 trials of 15 episodes each is seconds.

Manual Override

Quit

Figure 27. Supervised-RL tab on the GTK GUI

The type of environment was selected for labeling files. Then the number of episodes and trials were set along with the value of alpha, the learning rate. Once the parameters are set, the experiment can begin. During the experiment, distance readings from the laser range finder are posted on the display window as feedback to the operator.

CHAPTER IV

RESULTS

The results in these experiments are taken from 450 episodes of the robot traversing through the navigation task. Analysis is broken up by environment, the value of alpha, and the parameter being measured. Graphs of the time in seconds taken from the start to a terminal state for each episode are shown for the 5 trials performed under each experimental condition alongside graphs of the average time. The total accumulated reward is also graphed for each trial and the average of the trials. Comparisons are made between experiments with differing alpha values for the percent of episodes in which a goal state was found. Also, comparisons are made between the value of alpha that yielded a Q-table that most resembles the Supervisor Table. The Q-table is evaluated with each episode 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 36, 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 is averaged across the 5 trials, and the results are graphed to compare which experimental condition lead to a Q-table that matches the Supervisor Table closest.

Stability Experiment Results

The following are graphs of the time taken during a trial. The data is calculated during each episode of the experiment. A start time is recorded from the computer's clock at the beginning of an episode. Once an episode terminates, the end time is recorded. The difference, measured in whole seconds, is taken. Figure 28 shows the time measurements per episode for every trial performed in the stability environment with $\alpha = 0.1$, and Figure 29 shows the average of these measurements across trials.

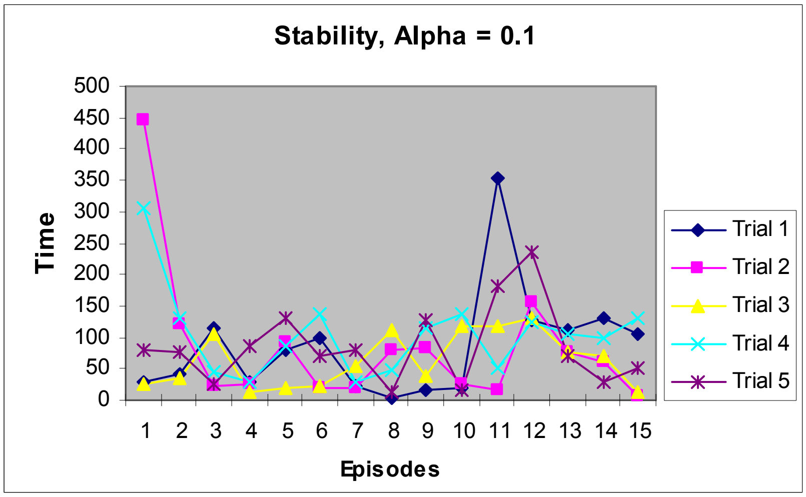


Figure 28. Time results for all trials from stability experiments, $\alpha = 0.1$

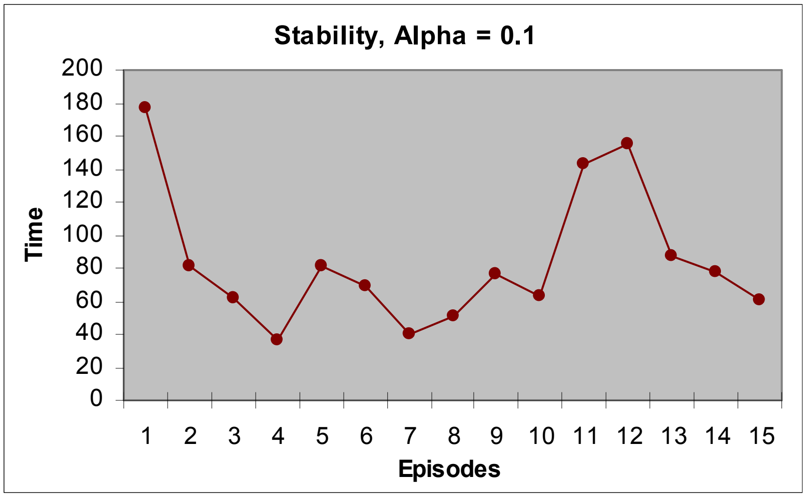


Figure 29. Time results averaged over trials from stability experiments, $\alpha = 0.1$

Figure 30 shows the time measurements per episode for every trial performed in the stability environment with $\alpha = 0.5$, and Figure 31 shows the average of these measurements across trials.

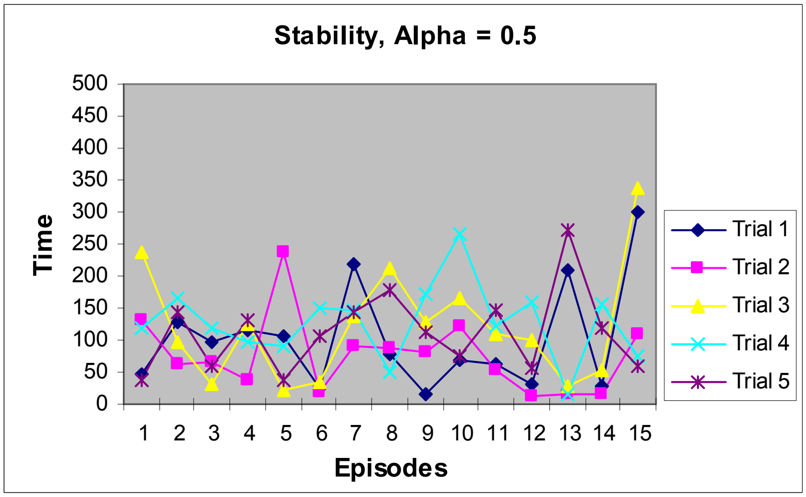


Figure 30. Time results for all trials from stability experiments, $\alpha = 0.5$

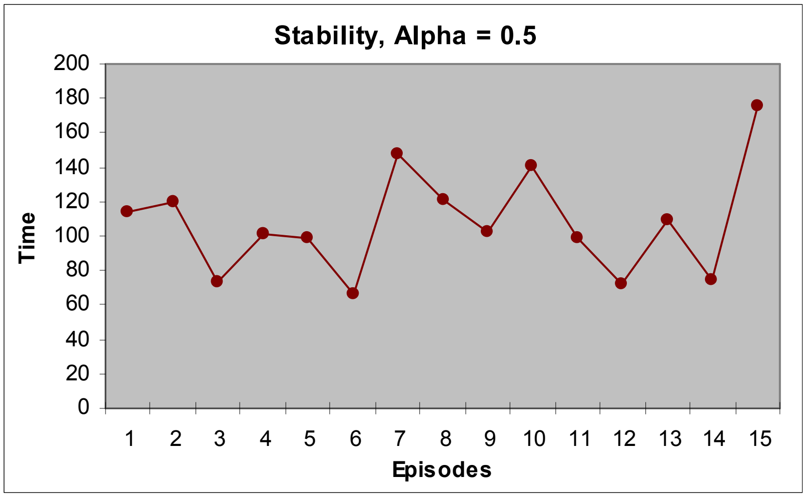


Figure 31. Time results averaged over trials from stability experiments, $\alpha = 0.5$

Figure 32 shows the time measurements per episode for every trial performed in the stability environment with $\alpha = 0.9$, and Figure 33 shows the average of these measurements across trials.

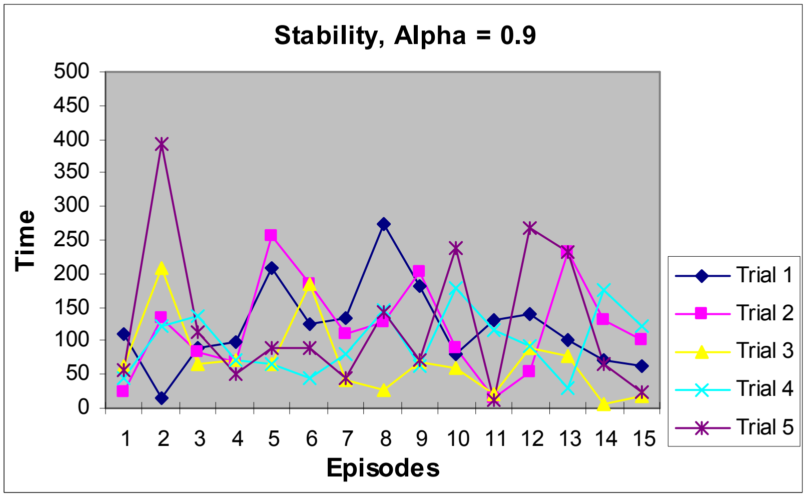


Figure 32. Time results for all trials from stability experiments, $\alpha = 0.9$

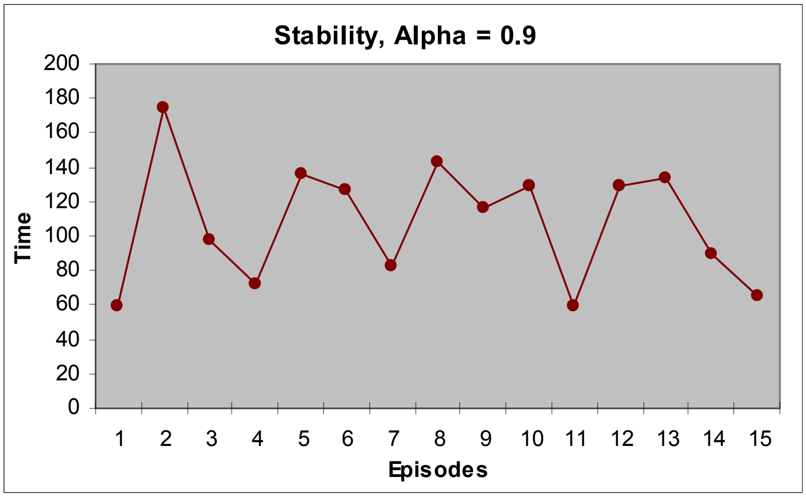


Figure 33. Time results averaged over trials from stability experiments, $\alpha = 0.9$

The next set of figures graph the total accumulated reward over the set of episodes. A record of total reward is summed during an episode, adding the reward received at each step along the episode from the start state to a terminal state. If an episode ends in a goal state, the highly positive reward usually counter-balances the negative reward accrued over the time of the episode. Figure 34 shows the reward measurements per episode for every trial performed in the stability environment with $\alpha = 0.1$, and Figure 35 shows the average of these measurements across trials.

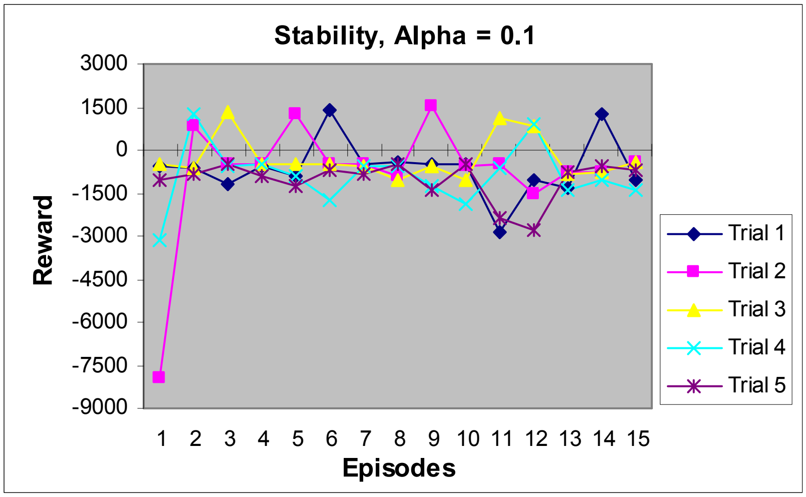


Figure 34. Reward results for all trials from stability experiments, $\alpha = 0.1$

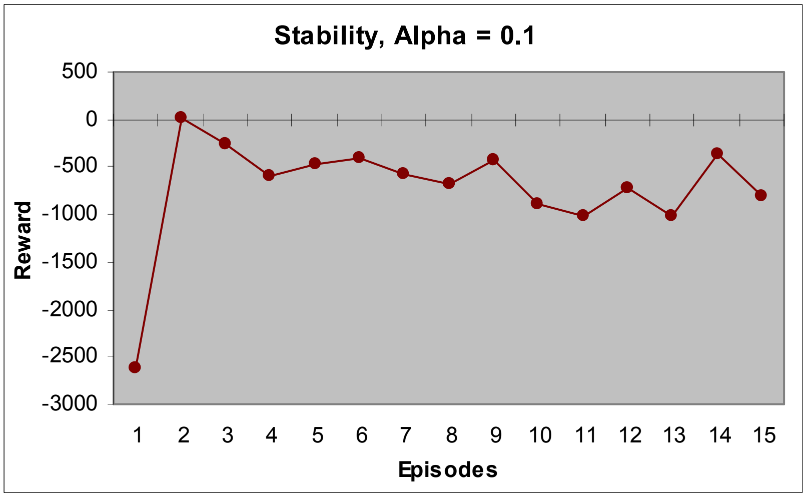


Figure 35. Reward results averaged over trials from stability experiments, $\alpha = 0.1$

Figure 36 shows the reward measurements per episode for every trial performed in the stability environment with $\alpha = 0.5$, and Figure 37 shows the average of these measurements across trials.

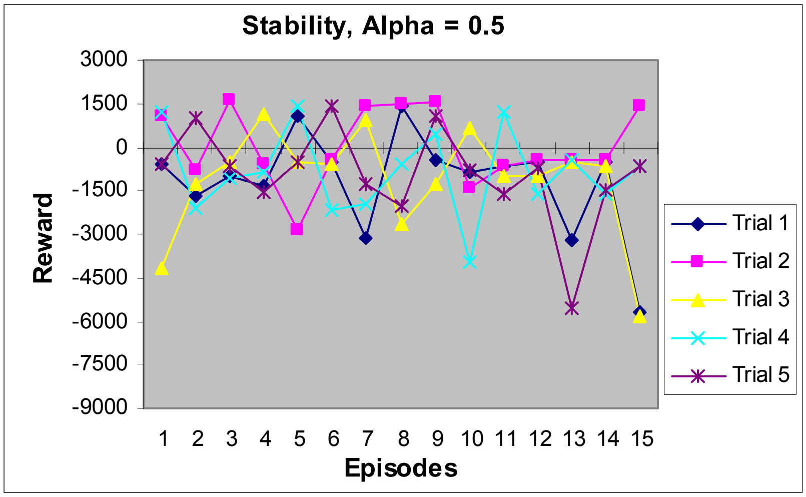


Figure 36. Reward results for all trials from stability experiments, $\alpha = 0.5$

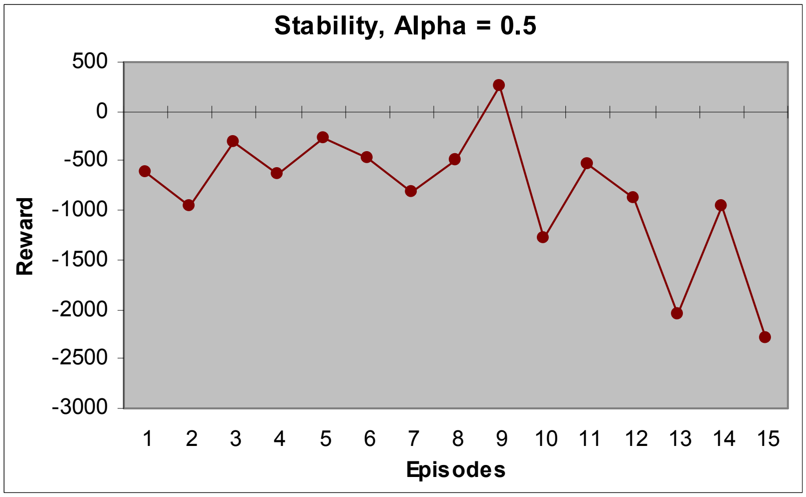


Figure 37. Reward results averaged over trials from stability experiments, $\alpha = 0.5$

Figure 38 shows the reward measurements per episode for every trial performed in the stability environment with $\alpha = 0.9$, and Figure 39 shows the average of these measurements across trials.

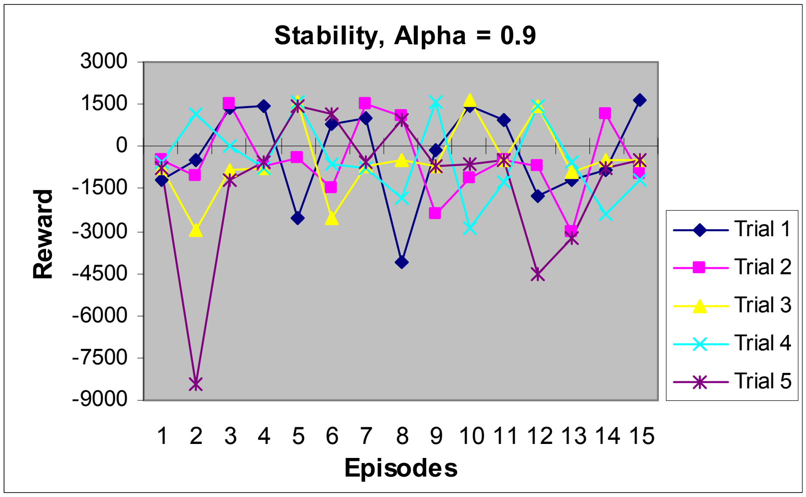


Figure 38. Reward results for all trials from stability experiments, $\alpha = 0.9$

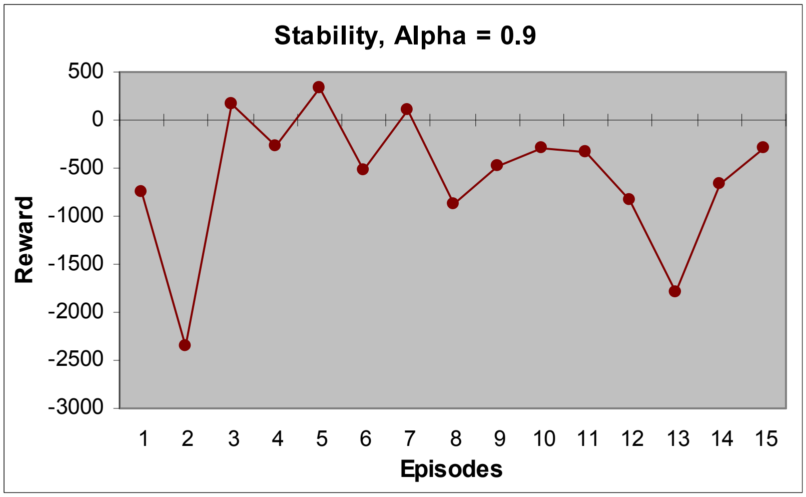


Figure 39. Reward results averaged over trials from stability experiments, $\alpha = 0.9$

As explained before, an episode can end in states other than the goal state. However, the goal is the objective of the task. Therefore, the percent of episodes that ended in a goal state under a particular experimental condition is noteworthy. The percent is taken by summing the number of episodes in a trial that ended in a goal state and dividing that number by 15, the total number of episodes. In a reinforcement learning problem, the theory would lead one to foresee that the goal would be found more often at the end of a trial than at the beginning. Therefore, calculations were made to find the percentage of when a goal state was found in episode 1, 2, 3,...,15, averaged over 5 trials. The overall average of finding a goal state is calculated for comparison between experimental conditions. Figure 40 shows the trial average of percent of times the goal was found, isolated by episode, for the stability experiment with $\alpha = 0.1$.

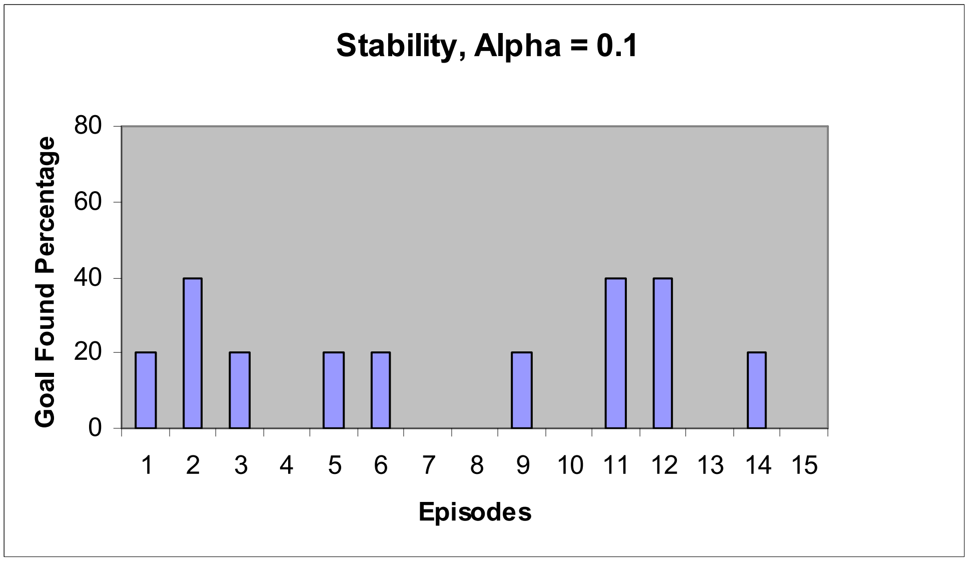


Figure 40. Goal found results averaged over trials from stability experiments, $\alpha = 0.1$

Figure 41 shows the trial average of percent of times the goal was found, isolated by episode, for the stability experiment with $\alpha = 0.5$.

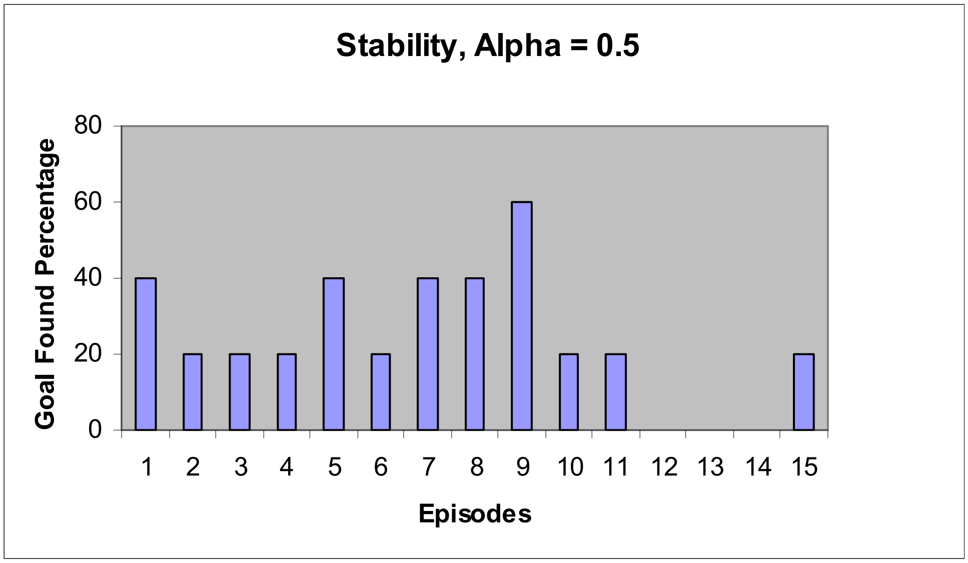


Figure 41. Goal found results averaged over trials from stability experiments, $\alpha = 0.5$

Figure 42 shows the trial average of percent of times the goal was found, isolated by episode, for the stability experiment with $\alpha = 0.9$.

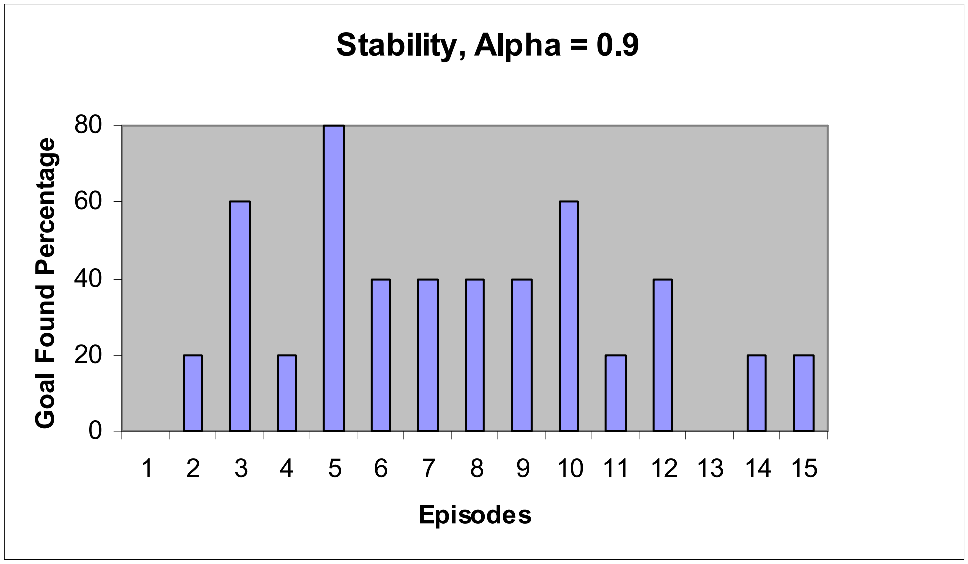


Figure 42. Goal found results averaged over trials from stability experiments, $\alpha = 0.9$

Figure 43 shows the average of finding the goal over all episodes for the stability experiment across values of alpha.

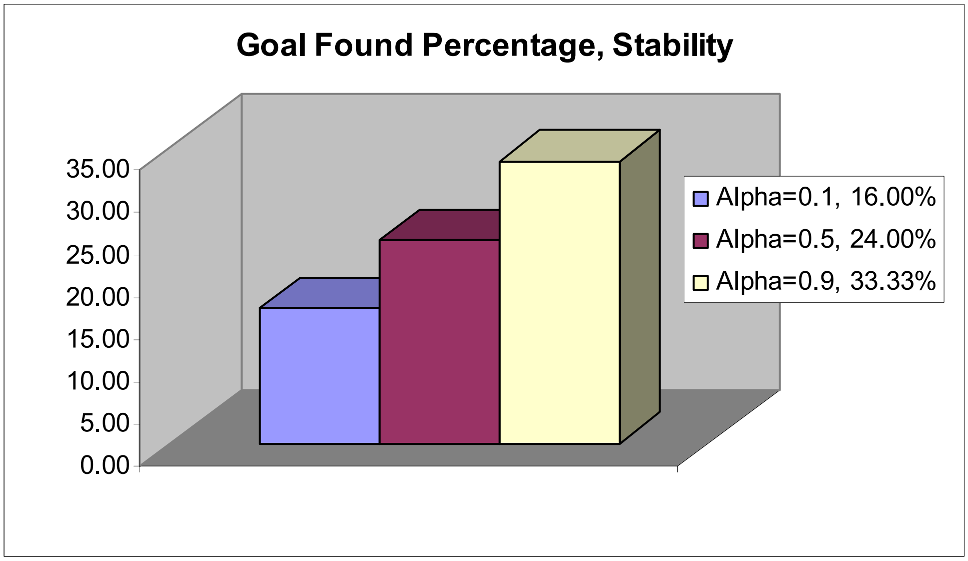


Figure 43. Goal found results from stability experiments for all alpha values

The last graph represents how well the final Q-table at the end of a trial matches the Supervisor Table, based on a set of 36 common states. A percentage is calculated by summing the number of states with matching actions and dividing that number by 36, the total number of states in the comparison set. Figure 44 shows the percent matching results for each value of alpha, averaged over the 5 trials preformed in the stability environment.

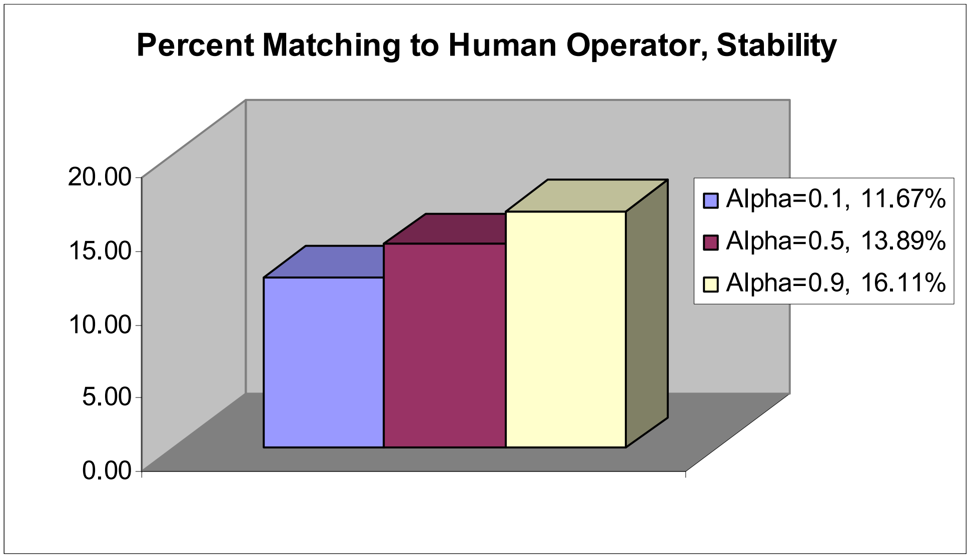


Figure 44. Matching results from stability experiments for all alpha values

Flexibility Experiment Results

The results for the flexibility experiments are processed via the same methods as the stability experiment results. Figure 45 shows the time measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.1$, and Figure 46 shows the average of these measurements across trials.

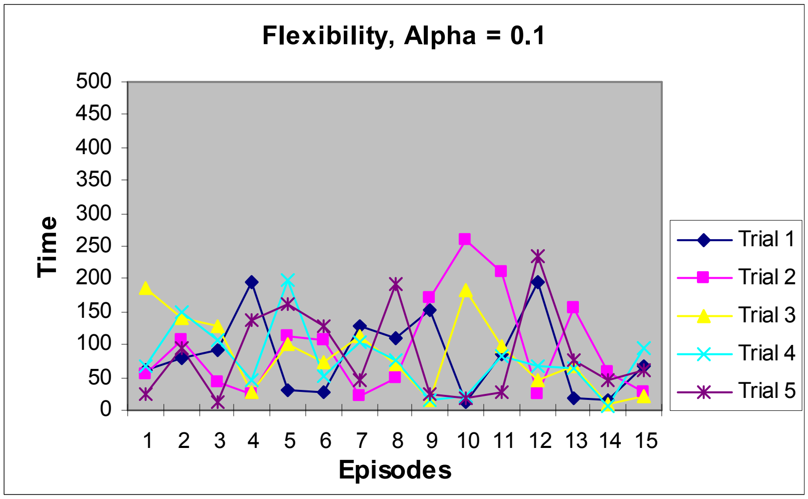


Figure 45. Time results for all trials from flexibility experiments, $\alpha = 0.1$

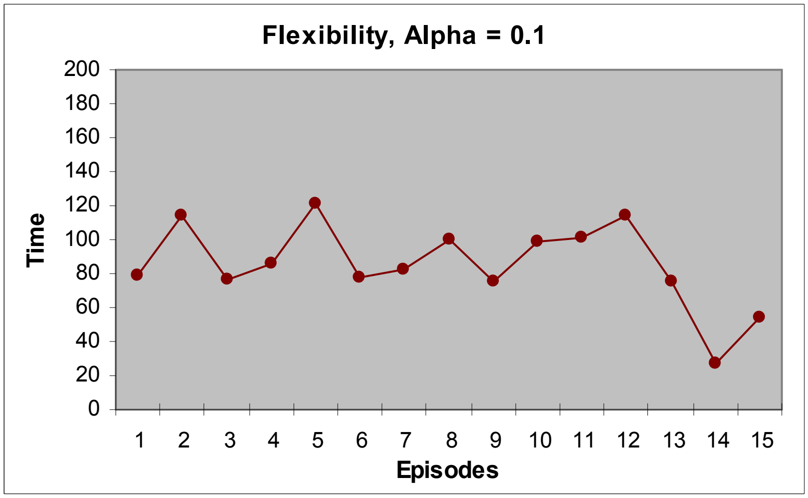


Figure 46. Time results averaged over trials from flexibility experiments, $\alpha = 0.1$

Figure 47 shows the time measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.5$, and Figure 48 shows the average of these measurements across trials.

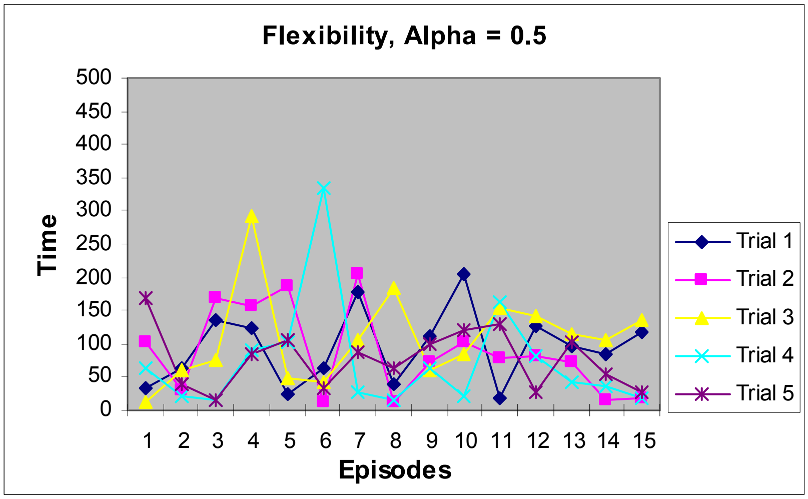


Figure 47. Time results for all trials from flexibility experiments, $\alpha = 0.5$

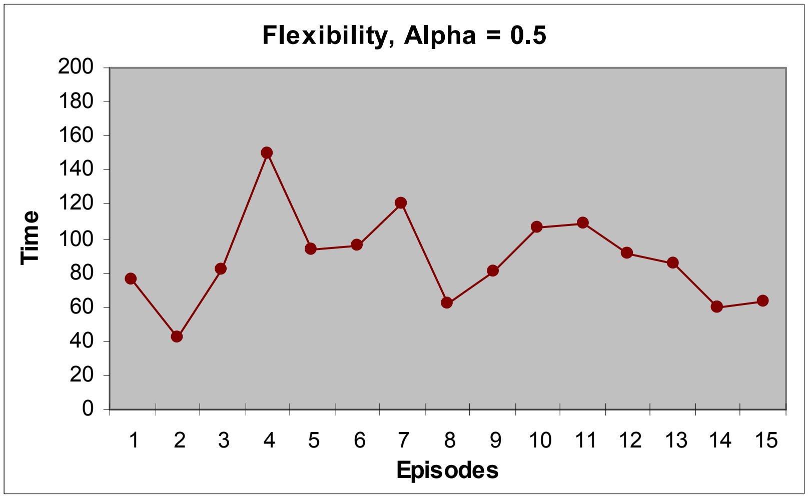


Figure 48. Time results averaged over trials from flexibility experiments, $\alpha = 0.5$

Figure 49 shows the time measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.9$, and Figure 50 shows the average of these measurements across trials.

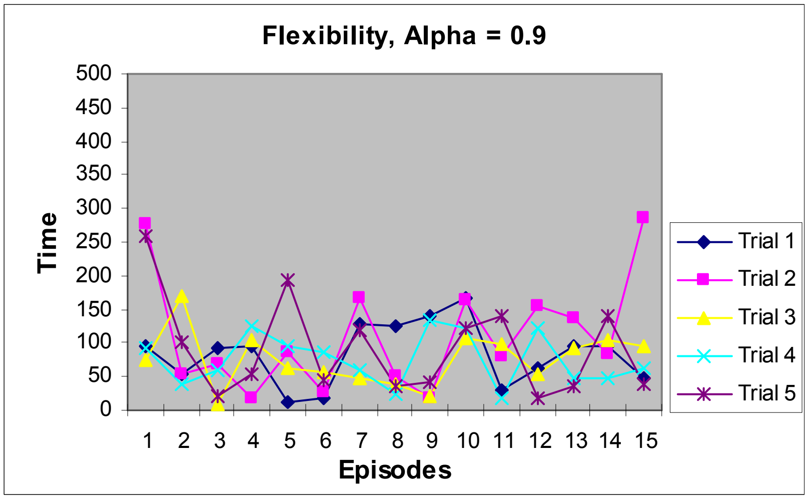


Figure 49. Time results for all trials from flexibility experiments, $\alpha = 0.9$

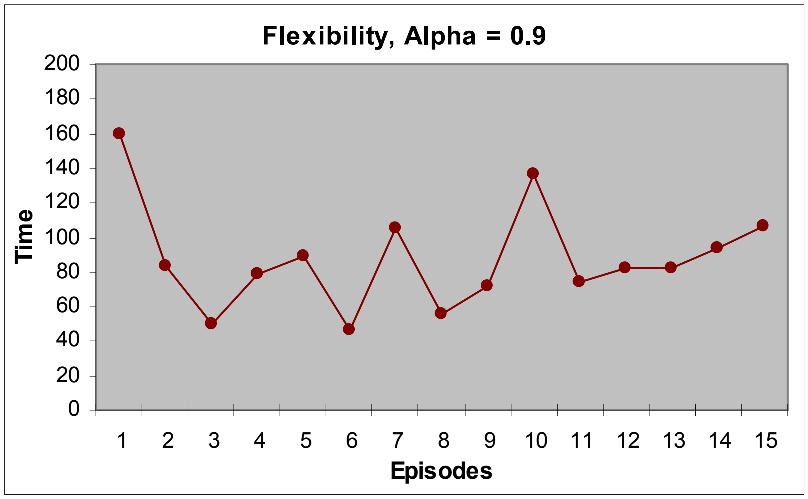


Figure 50. Time results averaged over trials from flexibility experiments, $\alpha = 0.9$

Figure 51 shows the reward measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.1$, and Figure 52 shows the average of these measurements across trials.

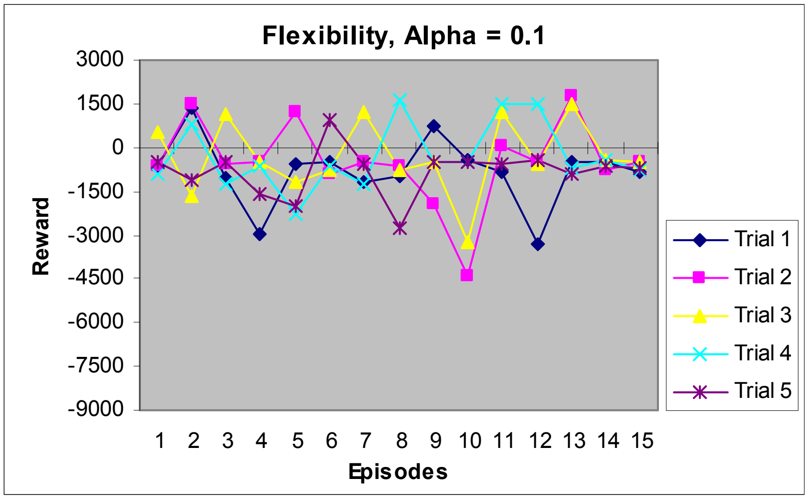


Figure 51. Reward results for all trials from flexibility experiments, $\alpha = 0.1$

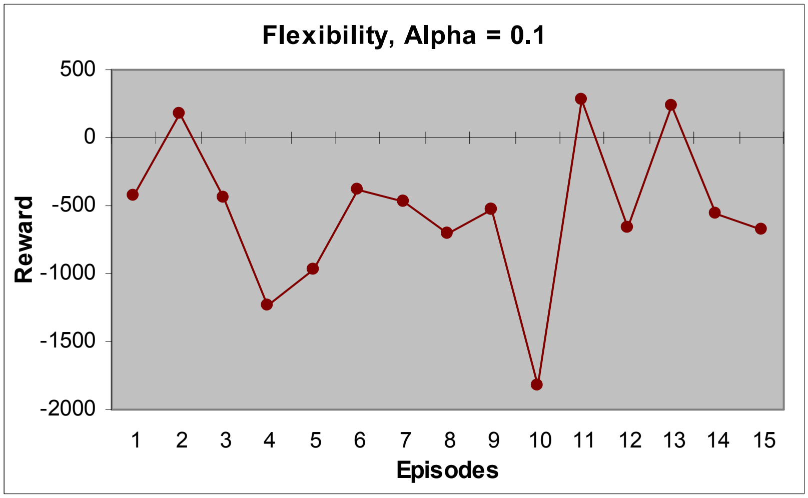


Figure 52. Reward results averaged over trials from flexibility experiments, $\alpha = 0.1$

Figure 53 shows the reward measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.5$, and Figure 54 shows the average of these measurements across trials.

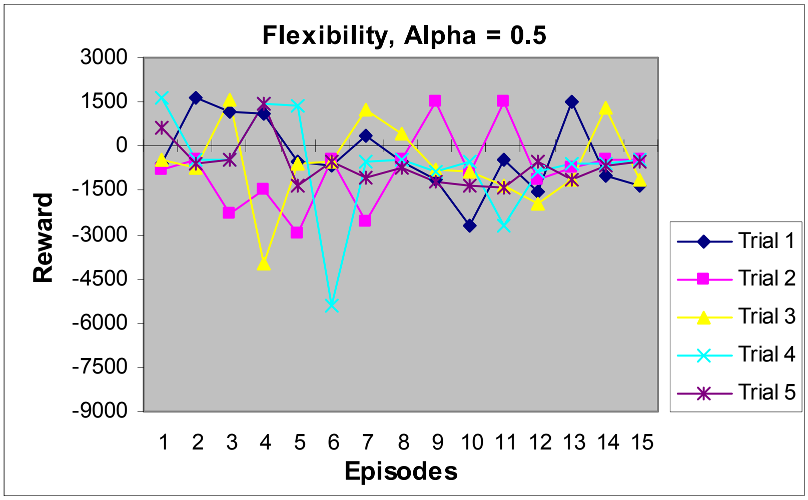


Figure 53. Reward results for all trials from flexibility experiments, $\alpha = 0.5$

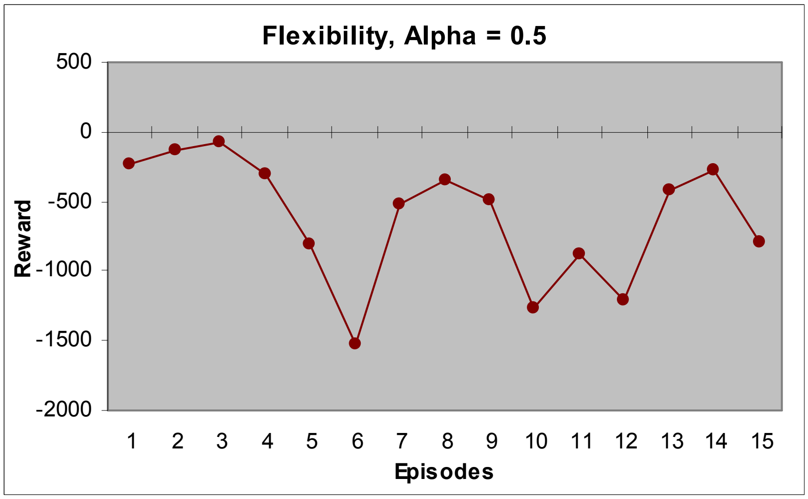


Figure 54. Reward results averaged over trials from flexibility experiments, $\alpha = 0.5$

Figure 55 shows the reward measurements per episode for every trial performed in the flexibility environment with $\alpha = 0.9$, and Figure 56 shows the average of these measurements across trials.

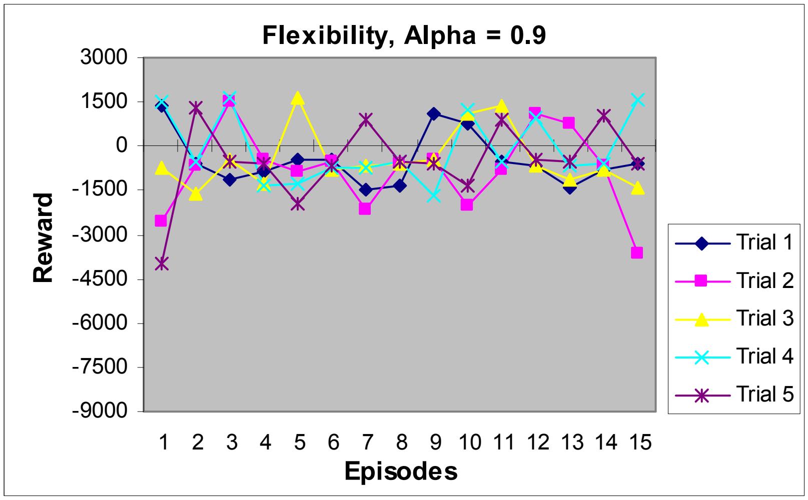


Figure 55. Reward results for all trials from flexibility experiments, $\alpha = 0.9$

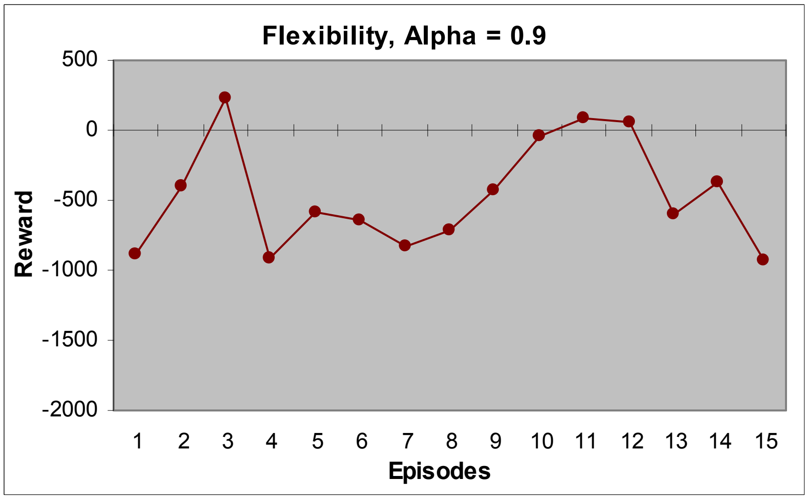


Figure 56. Reward results averaged over trials from flexibility experiments, $\alpha = 0.9$

Figure 57 shows the trial average of percent of times the goal was found, isolated by episode, for the flexibility experiment with $\alpha = 0.1$.

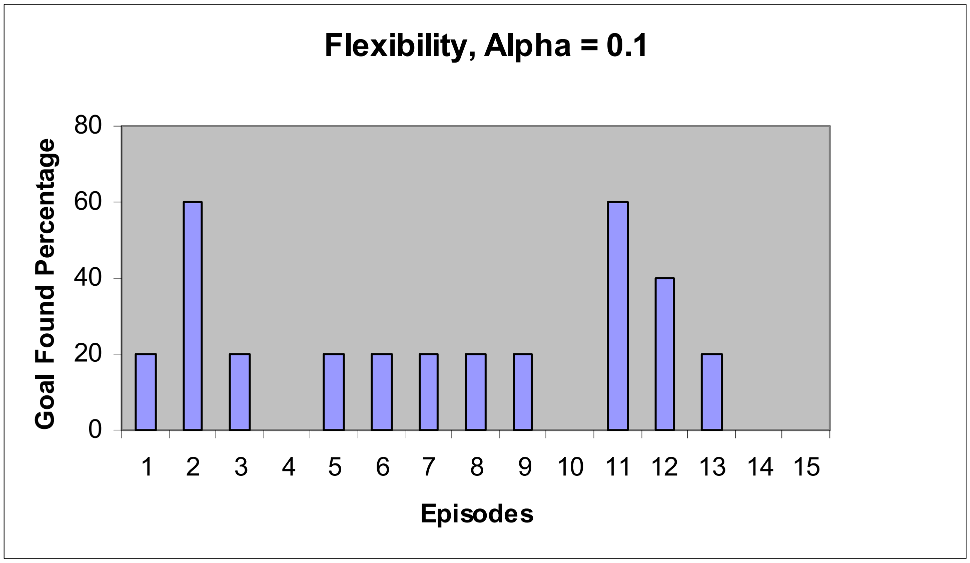


Figure 57. Goal found results averaged over trials from flexibility experiments, $\alpha = 0.1$

Figure 58 shows the trial average of percent of times the goal was found, isolated by episode, for the flexibility experiment with $\alpha = 0.5$.

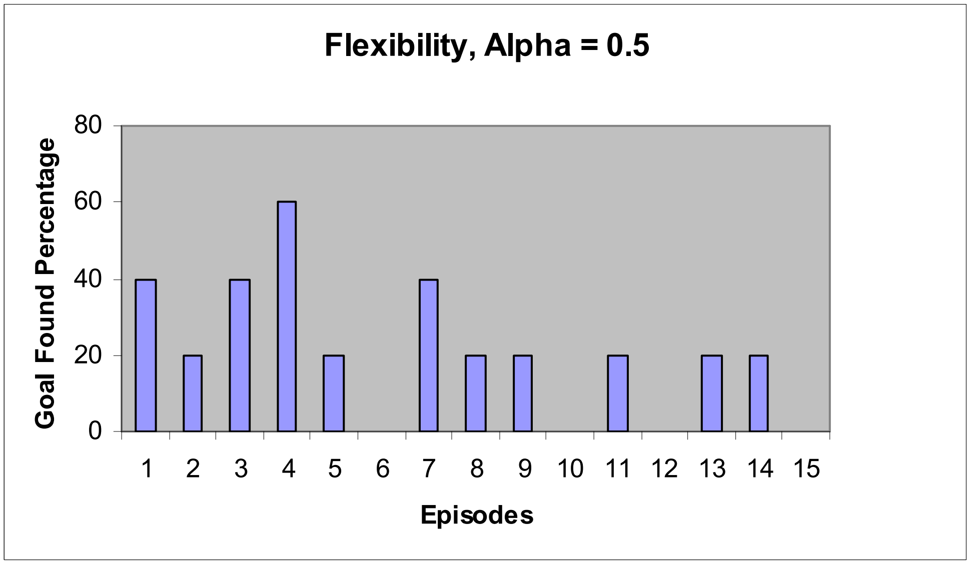


Figure 58. Goal found results averaged over trials from flexibility experiments, $\alpha = 0.5$

Figure 59 shows the trial average of percent of times the goal was found, isolated by episode, for the flexibility experiment with $\alpha = 0.9$.

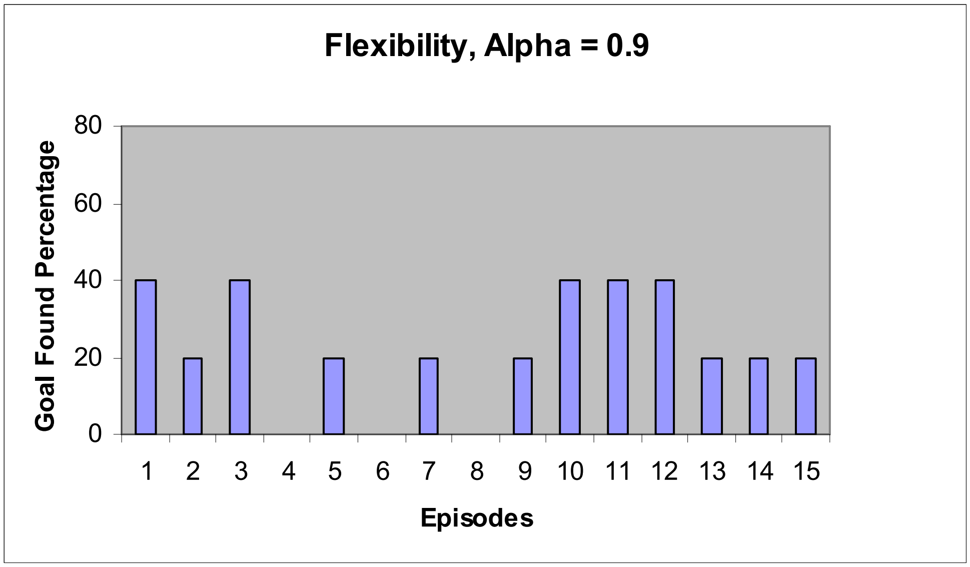


Figure 59. Goal found results averaged over trials from flexibility experiments, $\alpha = 0.9$

Figure 60 shows the average of finding the goal over all episodes for the flexibility experiment across values of alpha.

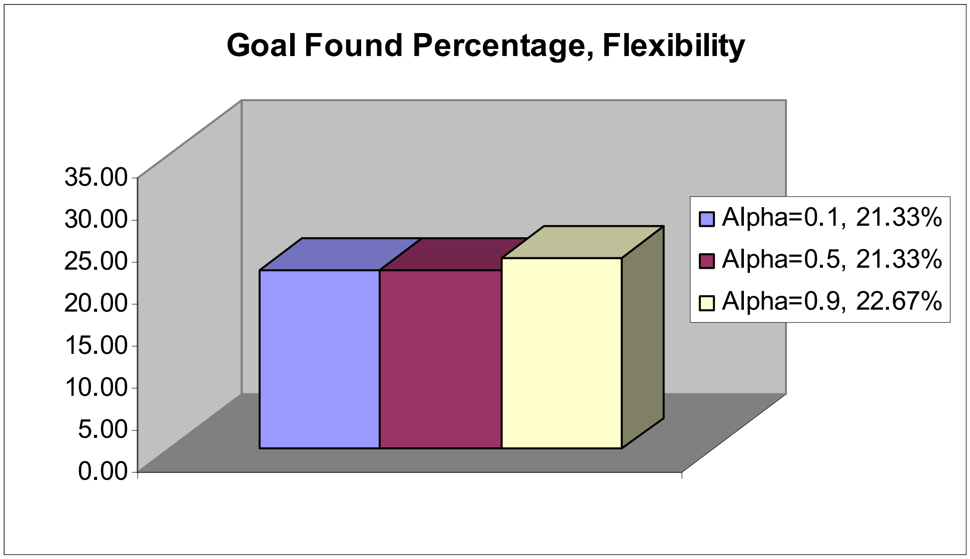


Figure 60. Goal found results from flexibility experiments for all alpha values

Figure 61 shows the percent matching results for each value of alpha, averaged over the 5 trials preformed in the flexibility environment.

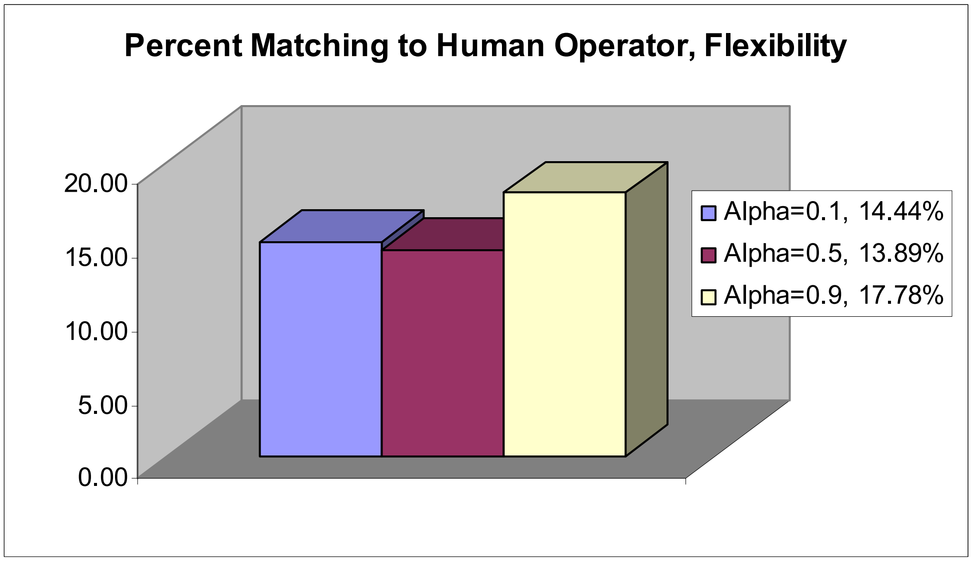


Figure 61. Matching results from flexibility experiments for all alpha values

Comparisons between Experiment Types

To better compare the time results against a baseline, one autonomous episode was run with the policy choosing actions from the Supervisor Table only. This episode found the goal in 58seconds. This information is used to compare how well the episodes faired in finding the goal in the least amount of time. Comparisons are made between the baseline and the single best, based on time, episode that ended in a goal state, for a particular experimental condition. The time measurements taken to find a goal state are then averaged across all trials under the condition for comparison. Figure 62 graphs the best time observed for an episode ending in a goal state against the baseline for each experimental condition, and Figure 63 shows the average time when the goal was found across trials.

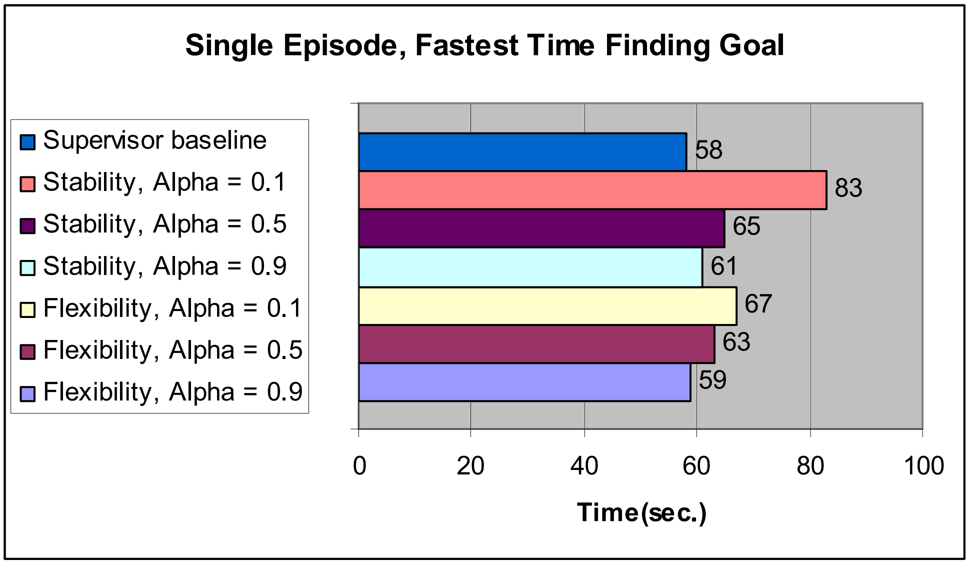


Figure 62. Results from fastest time finding goal for a single episode for all experiments

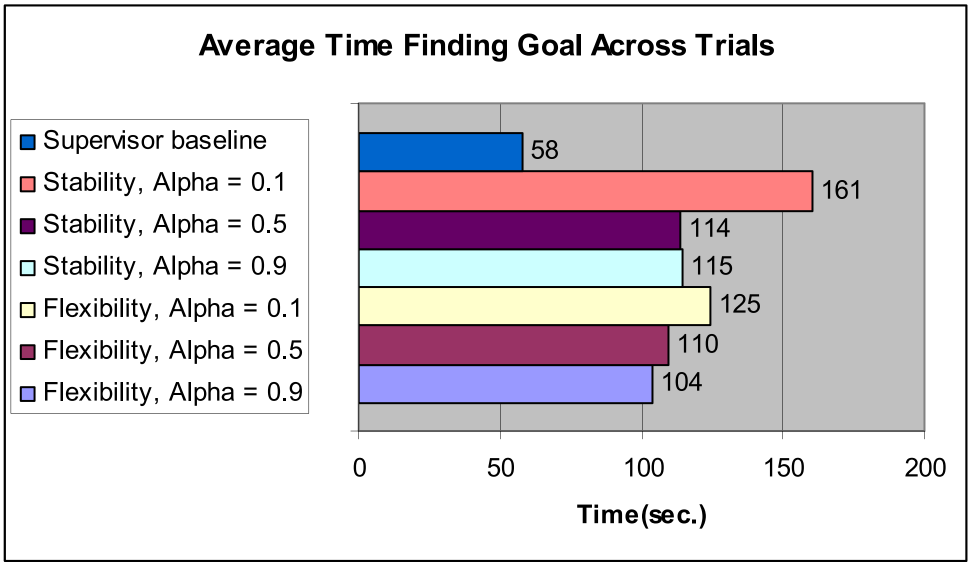


Figure 63. Results from fastest time finding goal over all trials for all experiments

Similar comparisons are made for the accumulated reward during the episode in which actions were used from the Supervisor Table only. This episode amassed a total reward of 1667.5. Comparisons are made between the baseline and the single best, based on reward, episode that ended in a goal state, for a particular experimental condition. The reward measurements taken to find a goal state are then averaged across all trials under the condition for comparison. Figure 64 graphs the best reward observed for an episode ending in a goal state against the baseline for each experimental condition, and Figure 65 shows the average reward when the goal was found across trials.

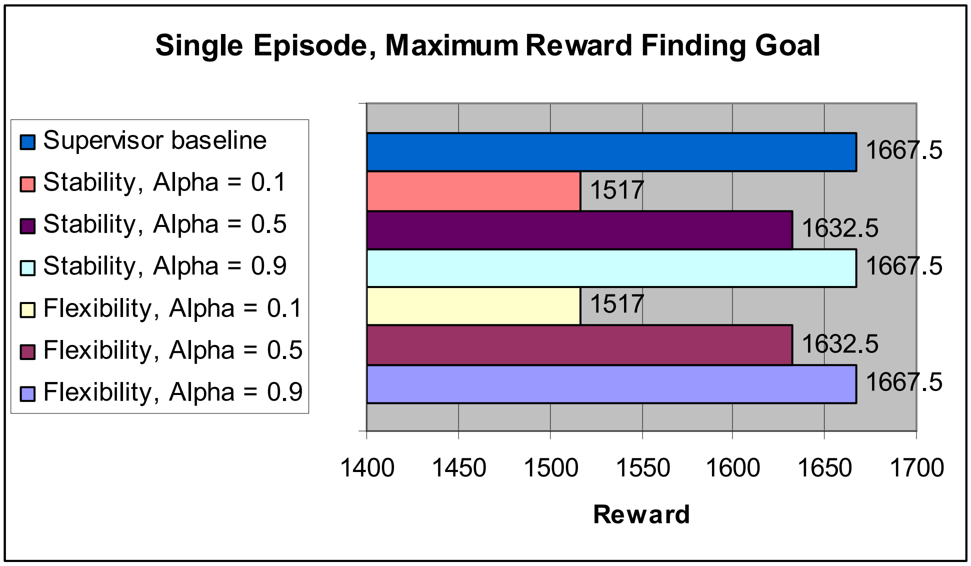


Figure 64. Results from most reward finding goal for a single episode for all experiments

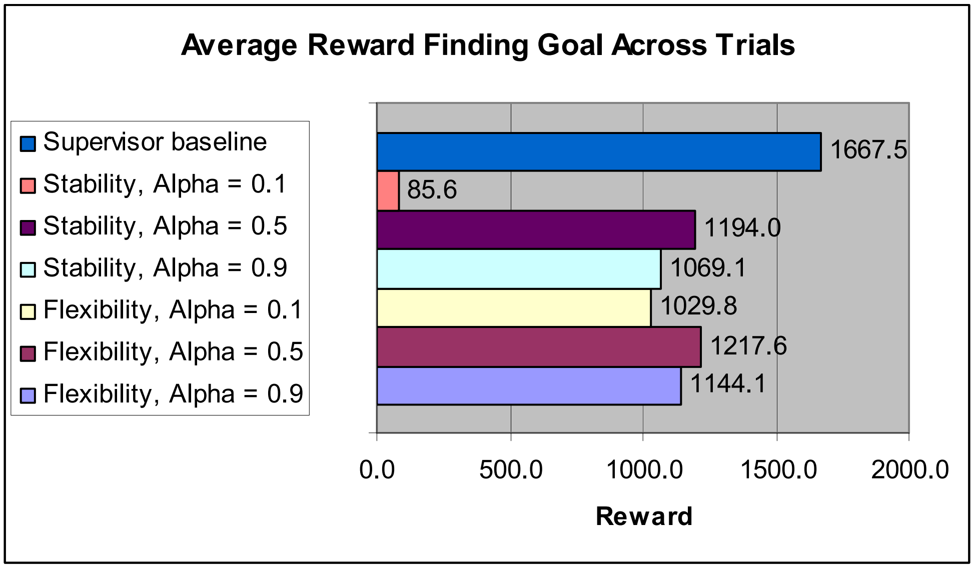


Figure 65. Results from most reward finding goal over all trials for all experiments

Figure 66 shows the trial average of finding the goal over all episodes for each experimental condition.

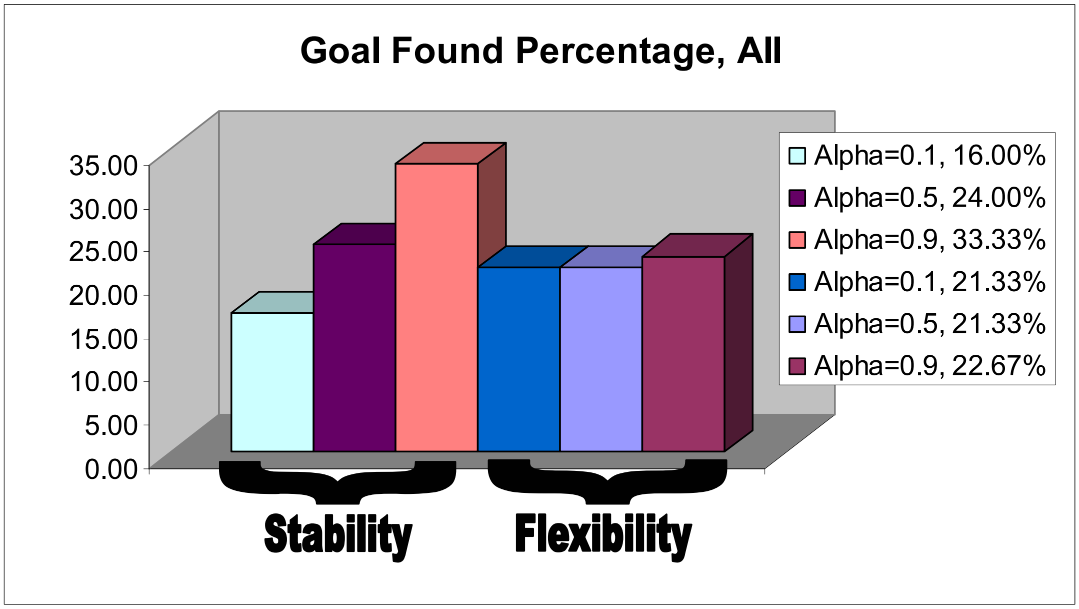


Figure 66. Goal found results for all alpha values for all experiments

Figure 67 shows the trial average of percent matching results for each experimental condition.

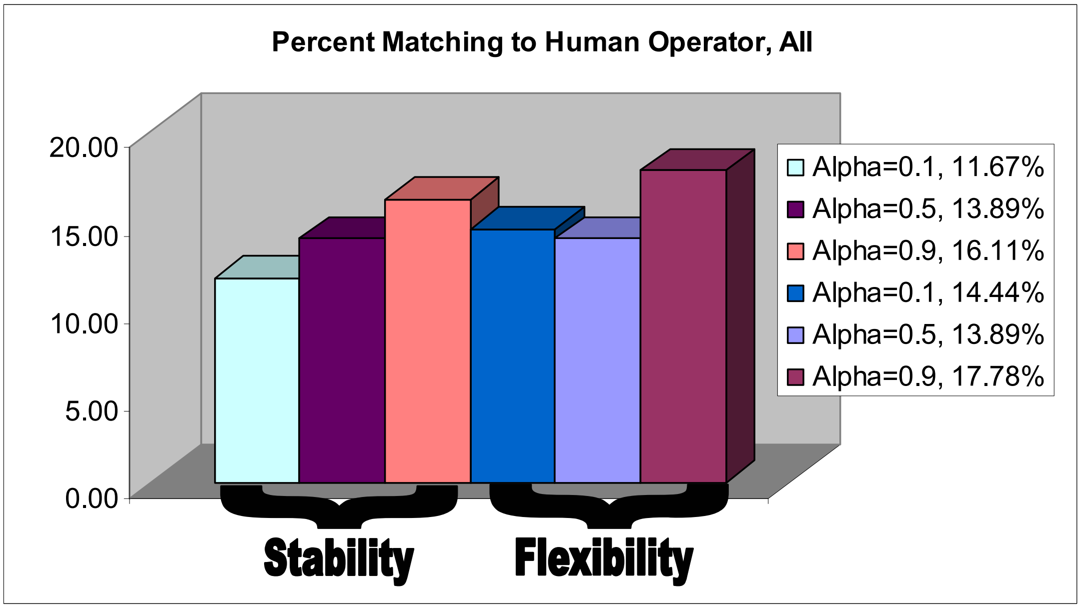


Figure 67. Matching results for all alpha values for all experiments

CHAPTER V

CONCLUSIONS & FUTURE WORK

Determination of Optimal Experimental Conditions

Analyzing the above time and reward graphs for each type of experiment and different value of alpha, one would expect to notice certain trends. The two objectives for the optimization task, which are also directly connected, are to find the goal in the least amount of time and maximize the total accumulated reward. At the beginning of a trial the robot knows little about the task, and the Q-table has only been initialized. As episodes increase over a trial, further evaluations of the Q-table take place. Therefore, one would expect the trend for the time taken to reach the goal to decrease across a trial while the reward increases. This trend was observed in some conditions in these experiments, but not all trials showed a strong adherence to the possible trends. Some reasons why most trials did not follow this trend is due to a high possibility of finding a fail state during any region of the environment and need to evaluate a large number of state-action pairs. The data suffers under the fact that a fail state could be found quicker than a goal state. One example holding to the decreased time and increased reward trend over episodes was observed in experiments in the stability environment with $\alpha = 0.1$. However, this desirable result may not forecast continued superiority on other evaluation parameters, discussed shortly.

Scrutiny was focused more on the overall number of times the robot could find the goal. Testing was conducted to determine which experimental setting yielded the most episodes ending in a goal and Q-table values that lead to the closest match to the Supervisor Table actions. From the conclusions, the Q-table with the best evaluation of the task could be selected for use in completing the task without random actions. Comparing results of how often the goal was found and how well the final Q-table matched the Supervisor Table lead to similar conclusions for both types of experiments. The results from the flexibility experiments were not significantly less productive than results conducted in the stability environment.

The obstacles placed in the environment during the flexibility experiments were meant to challenge the robot's ability to find the goal. Even though the obstacles were not designed to block the robot from reaching the goal, the hypothesis would be that the different environment setting would conclude with less-optimal results than the stability experiments. Actually, the results proved better at finding the goal in the case when $\alpha = 0.1$, as compared to stability experiments when $\alpha = 0.1$. The cases when $\alpha = 0.5$ and $\alpha = 0.9$, the percent for finding the goal is comparable to results from similar stability experiments even though there are additional obstacles in the flexibility environment.

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 stability experiments. In other cases the obstacles created states that encouraged actions toward the goal. Therefore, the comparable results may be balanced by the fact that the obstacles actually narrowed the environment toward the optimal path to the goal. For example, if the robot approached the hallway with the goal in a southeast direction, the trashcan may block the left side of the laser scans, which encourages a right turn towards the middle of the hallway with the goal. Also, the chair is in the goal region, an area in which the robot must be facing east to find the goal. The chair could block the right side of the laser scan, which encourages a left turn, which could lead the robot to turn east in the goal region and most likely find the goal. However, the obstacles can also put the robot in a fail state that would not occur if they were removed. If the robot is close to the west-side wall in the hallway with the goal and turns toward the chair, it is likely that a leg of the chair would cross the safety constraint barriers and signal a fail state. In a stability experiment, the robot would be able to turn or move forward, because there would be 500mm of uninterrupted space that the chair occupies in the flexibility experiments. All in all, the obstacles actually improved performance over the stability trials when $\alpha = 0.1$ and were comparable when $\alpha = 0.5$ and $\alpha = 0.9$.

From the above results, one can conclude that the experiments which were most successful were under conditions with $\alpha = 0.9$, especially for matching the Supervisor Table. This deduction seems reasonable, based on the literature as well. In general, a high value of alpha should lead to quick learning of the optimal policy. Also, this observation could be due to the design of the reward function used in these experiments. A dense reward function was used to describe the objectives of the task to the robot. A high value of alpha leads to drastic changes in the Q-table with each updated evaluation; therefore, most of the value designed in the reward function is used to update the Q-table. The combination of a dense reward function with a high value of α seems to lead to a better evaluation of the optimal actions to take in a given state in order to complete the task. Therefore, both environments report the closest matching of actions to the Supervisor Table when $\alpha = 0.9$.

Future Applications

Reinforcement learning continues to intrigue robotisists for its ability to learn an optimal policy for a task and adapt to changing environments. RL can also be used as a component in other learning architectures and does not have to be the main learning algorithm to aid an agent to learn. Therefore, future experiments could use the algorithm to evaluate parts of a task that are probabilistic but perhaps not every action in a task. For example, these experiments used a small number of simple actions. Future work on mobile robots could include more sophisticated actions. The state space for these experiments was very large compared to the number of episodes that could be run and the number for states necessary to visit in order to find the goal. Therefore, improvements could be made to reduce the state space describing the environment in order to allow the algorithm to evaluate enough state-action pairs to converge toward an optimal policy for finding the goal.

Also, one could modify the safety constraints such that a band of safety remains between the robot and an obstacle but such that fail states are not possible. Then the experiments could be evaluated on how quickly the robot found the goal, because an episode would end only in a goal state. Examples from other RL experiments involve an environment in which the agent can locate the goal from the start position or a sensor is fixated in a heading towards the goal. Such exploitations of the task were not used in these experiments but could be used in future work to quicken learning and quicken evaluation of different parameters.

The experimental parameter studied in these experiments was the learning rate, α, for two different environments. However, there are several variables that could be varied for evaluation of how the change affects performance of finding the goal and evaluating the Q-table. For example, the policy selected actions from the Supervisor Table half of the time. An interesting study would be to investigate if changing this selection percentage to 10% corresponds to an equal drop in performance. Also, a linear progression of ε was used in the policy in these experiments. Adjusts to the value of ε could be tested, which may result in a quicker finding of a goal state in this task.

APPENDIX A

SUMMATION OF NOTATION (adapted from Sutton and Barto, 1998)

t	discrete time step
T	final time step of an episode
s_t	state at t
a_t	action at t
r_t	reward at t, dependent, like s_t, on a_{t-1} and s_{t-1}
R_t	return (cumulative discounted reward) following t
π	policy, decision-making rule
S	set of all nonterminal states
S^+	set of all states, including the terminal state
$A(s)$	set of actions possible in state s
$\Re$	set of real numbers
$P_{ss'}^{a}$	probability of transition from state s to state under s' action a
$V^{\pi}(s)$	value of state s under policy π (expected return)
$V*(s)$	value of state s under the optimal policy
V, V_t	estimates of V^{π} or $V*$
$Q^{\pi}(s,a)$	value of taking action a in state s under policy π
$Q*(s,a)$	value of tacking action a in state s under the optimal policy
Q, Q_t	estimates of Q^{π} or $Q*$

SUMMATION OF NOTATION, continued

δ_t	temporal-difference error at t
γ	discount-rate parameter
ε	probability of random action in ε-greedy policy
α	learning-rate parameter

BIBLIOGRAPHY

*Activ*Media Robotics, Inc. (2001). Pioneer 2/PeopleBot Operations Manual, v9, October.

Beninger, R. J. (1983). The role of dopamine in locomotor activity and learning. *Brain Research Reviews*, 6(2):173-196.

Beninger, R. J. (1989). Dissociating the effects of altered dopaminergic function on performance and learning. *Brain Research Bulletin*, 23:365-371.

Brown, J., Bullock, D., and Grossberg, S. (2002). How the Basal Ganglia and Laminar Frontal Cortex Cooperate in Action Selection, Action Blocking, and Learning. *Department of Cognitive and Neural Systems and Center for Adaptive Systems, Boston University*. [Online]. Available: http://mbi.osu.edu/2002/ws2materials/bullock.pdf

Chevalier, G., Vacher, S., Deniau, J. M., and Desban, M. (1985). Disinhibition as a basic process in the expression of striatal function. I. The striatonigral influence on tecto-spinal/tecto-diencephalic neurons, *Brain Research*, 334:215-226.

CIS, Center for Intelligent Systems. (2005). A Biologically Inspired Adaptive Working Memory System for Efficient Robot Control and Learning. *Cognitive Robotics Laboratory Current Research Projects.* [Online]. Available: http://eecs.vuse.vanderbilt.edu/CIS/crl/wm.shtml

Coulom, R. (2002). Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In *13th International Conference on Algorithmic Learning Theory*, pages 402-413.

Cook, P. A. and Hayes, G. (2003). Could Active Perception Aid Navigation of Partially Observable Grid Worlds?. *Lecture Notes in Computer Science*, Vol. 2837:72-83, March.

Dahl, T.S., Matarić, M. J., and Sukhatme, G. S. (2002). Adaptive Spatio-Temporal Organization in Groups of Robots. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1044-1049, Lausanne, Switzerland.

Dayan, P., and Watkins, C. J. C. H. (2001). Reinforcement learning. *Encyclopedia of Cognitive Science.* MacMillan Press, London, England.

Deniau, J. M. and Chevalier, G. (1985). Disinhibition as a basic process in the expression of striatal functions. II. The striato-nigral influence on thalamocortical cells of the ventromedial thalamic nucleus. *Brain Research*, 334:227-233.

Dreyer, J. (2004). The Basal Ganglia and the Reward Pathway. *University of Fribourg, Switzerland*. [Online]. Available: http://www.unifr.ch/biochem/DREYER/dreyer1.html

Gerkey, B., Vaughan, R., and Howard, A. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the International Conference on Advanced Robotics,* pages 317-323.

Ghory, I. (2004). Reinforcement Learning in Board Games. *Technical Report CSTR-04-004, Department of Computer Science, University of Bristol, May.*

Houk, J., Adams, J., and Barto, A. (1995). *Models of Information Processing in the Basal Ganglia*, Houk, J., Davis, J., Beiser, D., and Schultz, W. editors, MIT Press, Cambridge, Massachusetts.

Hikosaka, O., and Wurtz, R. H. (1983a). Visual and occulomotor functions of monkey substantia nigra pars reticulata. III. Memory contingent visual and saccade responses. *Journal of Neurophysiology*, 49:1268-1284.

Hikosaka, O., and Wurtz, R. H. (1983b). Visual and occulomotor functions of monkey substantia nigra pars reticulata. IV. Relation of substantia nigra to superior colliculus. *Journal of Neurophysiology*, 49:1285-1301.

Huber, M., and Grupen, R. A. (1999). A Hybrid Architecture for Learning Robot Control Tasks. In *AAAI 1999 Spring Symposium: Hybrid Systems and AIModeling, Analysis and Control of Discrete + ContinuousSystems*. Stanford University, CA.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research,* 4:903-910.

Kimura, H. and Kobayashi, S. (1997). Reinforcement Learning for Locomotion of a Two-linked Robot Arm. In *Proceedings of the 6th European Workshop on Learning Robots*, pages144-153.

Mackintosh, N. J. (1983). *Conditioning and Associative Learning.* Oxford University Press, New York.

Marr, D. (1982). *Vision: Computational investigation into human representation and processing of visual information*. W. H. Freeman and Co., San Francisco.

Martínez-Marín, T. and Duckett, T. (2005). Fast Reinforcement Learning for Vision-guided Mobile Robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1425-1430, Barcelona, Spain.

Murphy, R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, Massachusetts.

Provost, J., Kuipers, B. J., and Miikulainen, R. (2004). Self-Organizing Perceptual and Temporal Abstraction for Robot Reinforcement Learning. In *AAAI-04 Workshop on Learning and Planning in Markov Processes*, 79-84.

Randløv, J., Barto, A. G., and Rosenstein, M. T. (2000). Combining Reinforcement Learning with a Local Control Algorithm. In *Proceedings of the 17th International Conference on Machine Learning*.

Robbins, T. W. and Everitt, B. J. (1992). Functions of dopamine in the dorsal and ventral striatum. *Seminars in Neuroscience*, 4:119-127.

Rosenstein, M.T. and Barto, A. G. (2004). Supervised actor-critic reinforcement learning. In *Learning and Approximate Dynamic Programming: Scaling Up to the Real World.* pages 359-380. John Wiley & Sons, Inc., New York.

RTI, Real-Time Innovations, Inc. (2004). Can Ethernet be Real-Time? Network Data Delivery Service (NDDS). *NDDS Data Sheet & Product Brief.* [Online]. Available: http://www.rti.com/products/ndds/literature.html

Russell, S. and Norvig P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.

Schultz, W. (1992). Activity of dopamine neurons in the behaving primate. *Seminal Neuroscience*, 4:129-138.

Schultz, W., Apicella, P., and Lunjberg, T. (1993). Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *Journal of Neuroscience,* 3:900-913.

Schultz, W., Dayan, P., and Montague, R. R. (1997). A neural substrate of prediction and reward. *Science* 275:1593-1599.

Schultz, W. and Romo, R. (1988). Neuronal activity in the monkey striatum during the initiation of movements. *Experimental Brain Research,* 71:431-436.

Segway LLC. (n.d.). About the Segway Robotic Mobility Platform. [Online]. Available: http://www.segway.com/segway/rmp/

Selfridge, O. G. (2000). Comment: AI's greatest trends and controversies. In Hearst, M. A. and Hirsh, H. editors, *IEEE Intelligent Systems & Their Applications*, 15(1).

SICK, Inc. (2005). Product Information – Laser Measurement Systems. [Online]. Available: http://ecatalog.sick.com/Products/

Smart, W. D. and Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9-44.

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

Tangamchit, P., Dolan, J., and Khosla, P. (2002). The Necessity of Average Rewards in Cooperative Multirobot Learning, In *IEEE Conference on Robotics and Automation*.

Tesauro, G. (1995). Temporal-difference learning and TD-Gammon. *Communications of the ACM*, 38(3).

Vaughan, R. T., Howard, A., and Gerkey, B. P. (2002). Stage User Manual 1.3. *Player/Stage Project*. [Online]. Available: http://playerstage.sourceforge.net, November.

Vu, D. (1998). Basal ganglia [*Lecture handouts*]. University of New South Wales. [Online]. Available: http://www.geocities.com/medinotes/basal_ganglia.htm, September, 21st.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD Thesis. University of Cambridge, England.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning,* 8:279-292.