ADAPTIVE TEMPORAL DIFFERENCE LEARING OF SPATIAL MEMORY IN THE WATER MAZE TASK

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

ERIK E. STONE

Dr. Marjorie Skubic, Thesis Supervisor

MAY 2009

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled:

ADAPTIVE TEMPORAL DIFFERENCE LEARNING OF SPATIAL MEMORY IN THE WATER MAZE TASK

presented by Erik E. Stone,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Marjorie Skubic, Ph.D.

Dr. James Keller, Ph.D.

Dr. Yi Shang, Ph.D.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Dr. Marjorie Skubic for her knowledge and guidance in completing this work. I would also like to thank Dr. James Keller for his help in completing this work. Finally, I would like to thank Dr. Yi Shang for being on my thesis committee and reviewing this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem Statement

The goal of this work was to evaluate the performance of a Temporal Difference (TD) approach to the learning of spatial memory for a robot in a dry version of the Morris water maze task. The Morris water maze task is a spatial memory task in which an association between cues from the environment and position must be learned in order to locate a hidden platform. To that end, earlier work performed by Busch et al. [1] was extended from a simulated dry water maze environment into a physical environment on a real robot. This effort presented a number of challenges, but resulted in a system that was capable of learning the necessary action preferences to successfully, and efficiently, navigate to a hidden platform.

Additionally, the TD learning approach was extended to improve its performance in non-stationary environments where the hidden platform location was not fixed. The original TD learning approach was not adaptable in these non-stationary environments, as previously learned action preferences hindered the learning of new action preferences necessary to successfully navigate to a new hidden platform location. Specifically, the TD learning approach was extended by adding the ability to explicitly forget current action preferences based on previous rewards received. This extended version was then evaluated in simulation.

1.2 Overview

Given the ability of animals to navigate and interact with the complex world around them, biological systems have been a source of much inspiration in the field of robotics. The inspiration for this work comes from a behavioral procedure designed by Richard Morris called the Morris water maze task. The Morris water maze task is a spatial memory task in which an association between cues from the environment and position must be learned in order to locate a hidden platform. The task provides an interesting domain in which to study and evaluate the learning of spatial memory.

This work essentially begins where the work conducted by Busch et al. [1] left off. Specifically, Busch et al. developed and tested, in simulation, a Temporal Difference (TD) learning approach to spatial memory for a robot in a dry version of the Morris water maze task. The setup used for these experiments consisted of a simulated 2D world modeled after a dry water maze environment from earlier work by Krichmar et al. [2]. The 2D world was simply a rectangular room with various colored blobs around the edges. Additionally, a hidden platform was placed in the environment that could only be detected by the robot when it was directly over the top of the platform.

A simulated differential drive robot equipped with three cameras capable of detecting the colored blobs was used for the experiments. The goal was for the robot, equipped with the TD learning system, to be able learn the action preferences (mappings from states to actions) necessary for it to successfully locate the hidden platform. The only information available to the robot to achieve this task was the perceptual information about the colored panels obtained from the cameras.

In order to limit the perceptual space, a self organizing map (SOM) was used. The SOM discretized the perceptual space into a useable number of possible states, which made the task of learning the action preferences associated with each state possible for the TD learning system. The SOM was trained by first collecting a large number of perceptual vectors from the simulated environment.

Testing of the system consisted of letting the robot complete "runs" through the environment, then observing changes in its performance. Each run consisted of a maximum of 50 steps. At each step, the robot would observe its current perceptual state, determine which of the possible SOM nodes it was closest to, and then execute the move that was currently favored by the TD learning system for that SOM node. During and after each run the TD learning system would receive scalar rewards based on its performance. As noted in [1], this approach yielded good results that not only allowed the robot to learn the correct action preferences to locate the hidden platform, but allowed the TD system to outperform, given enough training episodes, a probabilistic graph search method.

Given these encouraging results, the work presented here was focused on extending what was achieved in this simulated environment to a physical environment, with a real robot, and on evaluating and possibly improving the performance of the TD system in environments where the hidden platform was not stationary, as was case in the experiments conducted by Busch et al. [1].

In a first step towards those goals, a physical environment modeled after both the simulated environment used by Busch et al. [1], and the environment used by Krichmar et

al. [2], was constructed. Additionally, a Pioneer 3-DX robot was equipped with the necessary hardware for the task.

A number of challenges not present in the simulated world were also addressed in order get the physical implementation of the dry water maze environment working. Specifically, problems with obtaining the correct perceptual state from the environment using the cameras mounted on the robot had to be overcome. Additionally, the problems of limited battery life and imperfect odometry information necessary for resetting the robot after each training episode had to be addressed. Ultimately, the physical implementation and testing of the TD system for learning spatial memory were successful, and a number of experimental trails were conducted. These results are presented later in this work.

With the physical implementation of the TD learning system successful, work then began on evaluating the performance of the system in environments with non-stationary platform locations. In order to speed the evaluation, these tests were conducted in the simulated environment used by Busch et al. [1]. Based on initial testing, it was soon apparent that the TD learning system, as implemented, performed unsatisfactorily in non-stationary environments.

Work was then conducted with the goal of improving the performance of the system in such non-stationary environments. Ultimately, a framework which allowed the TD learning system to actively forget its current action preferences based on past rewards received was implemented which allowed for much improved performance of the system in non-stationary environments.

Chapter 2

Background and Related Work

2.1 Morris Water Maze

The Morris water maze [3] is a novel behavioral procedure originally designed by Richard Morris for studying spatial localization in the rat. In the typical Morris water maze experiment, a rat is placed into a circular pool of water from which the only escape is a raised platform. The raised platform is positioned just below the water's surface, and the water is made opaque to hide the platform from view of the rat, thus ensuring no local cues from the platform are available to aid escape behavior. Although no local cues from the platform are available, only a few trials are required before a normal rat learns to swim directly toward the platform, given that it remains in a fixed location, even with a unique starting location in the environment. However, if the platform location is varied randomly for each trial, the rat cannot learn to find it. This provides evidence that rats escape by learning the spatial position of the platform relative to distal cues [4].

Studies have shown that single cells in the hippocampus respond during spatial learning, and that certain cells only fire when animals are in a specific area of a familiar environment [4]. In [4], Morris et al. used the water maze procedure described above to study the effect of hippocampal lesions on the navigational ability, and spatial learning, of rats. Through various experiments, they found that lesions to the hippocampus result in deficits in spatial learning and memory, and that total hippocampal lesions cause a lasting place-navigational impairment.

Specifically, Morris et al. [4] compared the performance of normal and brain-lesioned rats using both the normal conditions of the Morris water maze task and with a fixed but visible platform. For one study, both lesioned and unlesioned rats were trained, using the water maze task, with the platform hidden. It was found that rats with hippocampal lesions showed significantly decreased performance as compared to unlesioned rats. Next, the platform was left in the same location, however it was made visible. Soon, the performance difference between the two groups, lesioned and unlesioned, effectively disappeared. However, when the platform was then re-hidden the performance difference reappeared, even though the platform remained in the same location. These results provided evidence for the fact that lesions of the hippocampus result in deficits in spatial learning and memory. Other researchers have used the water maze procedure to evaluate the effect of drugs and cerebral neurotoxins on the spatial learning of rats [3].

The Morris water maze task has also been used by a number of researchers to study computational models of the hippocampus and spatial learning. Redish and Touretsky [5] used a simulated version of the water maze task to evaluate a computational model of the hippocampus. In addition, Brown and Sharp [6] used a simulated water maze environment to investigate how spatial behavior could be guided by spatial information in the hippocampal formation. Their model learned mappings between the firings of simulated hippocampal place and head direction cells and particular movements of a simulated rat to find the hidden platform location.

Foster et al. [7] used Temporal Difference (TD) learning to model how hippocampal place cells might be used for spatial navigation by rats. First, they simulated

a reward based navigational approach based solely on input from place cells. Second, they simulated a combined approach using input from place cells and information about the rats' self motion to acquire a goal independent coordinate system. Like Brown and Sharp [6], they used simulated place cells to provide a representation of the current position of the rat, as opposed to direct visual perceptual cues from the environment.

Krichmar et al. [2] constructed a dry version of the water maze task to assess the spatial memory of a brain-based device called Darwin X, whose behavior was guided by a simulated nervous system modeled on the anatomy and physiology of the vertebrate nervous system. A 16 by 14 foot rectangular room was used as the water area, with a hidden circular platform made of reflective paper. Darwin X was equipped with a color camera for vision, odometry for self-movement information, an IR sensor for platform detection, and IR sensors for obstacle avoidance.

Based on Krichmar's work, Busch et al. [1] used a simulated water maze environment to compare an attributed probabilistic graph search navigational approach and a TD learning navigational approach based solely on visual cues from the environment. The simulated robot was equipped with three cameras to gather perceptual information from the environment and used a Self-Organizing Map (SOM) [8] to discretize the perceptual space. This work showed that given sufficient training the TD learning navigational approach was actually able to outperform the probabilistic graph search method.

2.2 Temporal Difference Learning

In this work, Temporal Difference (TD) learning [9] is used to learn the action preferences necessary to successfully locate the hidden platform in the water maze environment. TD learning is a reinforcement learning procedure which is driven by the difference between temporally successive predictions. In general, reinforcement learning is the process of learning how to map states to actions to maximize a reward signal. Reinforcement learning generally differs from other forms of computational intelligence or machine learning in that this reward signal is delayed. Furthermore, a given action may impact not only the immediate reward, but all subsequent rewards [10].

Fundamentally, TD learning is a bootstrapping method for estimating a state value function using experience gathered following a given policy, π. The simplest TD method, *TD(0)*, can be formalized as [10]:

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + V(s_{t+1}) - V(s_t)]$$

where V is the state value function being learned, α is a learning rate parameter, and r is the reward signal. Ultimately, the estimated value of a given state, s_t is updated based on the reward, r_{t+1}, received after taking whatever action is specified by policy π for state s_t, along with the difference between the estimated value of the state s_t, and the estimated value of the following state, s_{t+1}.

For most of the experiments conducted in this work (those making use of the Working Memory Toolkit), however, an off policy TD control algorithm known as Q-learning was used [11]. At its heart, Q-learning attempts to learn a state-action pair value

function independent of the policy used to select actions. In its simplest form, one step Q-learning can be formalized as [10]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where Q is the state-action pair value function that is to be learned, s_t is the state at time t, a_t is the selected action at time t, α is a learning rate, r is the reward signal, and γ is a discount rate. Similar to *TD(0)* described above, Q-learning updates the estimated value of a given state-action pair based on the reward, r_{t+1}, received after taking whatever action is specified, by whatever policy is currently being followed, for state s_t; however, whereas in *TD(0)* the estimated state value was also updated based on the difference between the estimated value of the state s_t, and the estimated value of the following state, s_{t+1}, it can be been seen that with Q-learning, the state-action pair value is updated not by the difference between $Q(s_{t+1},a_{t+1})$ and $Q(s_t,a_t)$, but by the difference between $\max_a Q(s_{t+1}, a)$ and $Q(s_t,a_t)$. That is, following a given action, a_t, the value of the previous state-action pair is updated based on the reward received after taking the action a_t, along with the difference between the estimated value of the previous state-action pair and the maximum estimated value of all the state-action pairs for state s_{t+1}. Simply stated, Q-learning attempts to estimate the state-action pair value function for the greedy policy, although this policy is not necessarily the policy being used to select actions.

Although the two formalizations laid out above give a basic overview of TD methods, they are both single step methods. Specifically, an error signal for a given time step, called the TD error, denoted by δ_t and (for TD learning) characterized as:

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

affects only the value estimate of state s_t. In many situations, however, a number of previous states, or in the case of Q-learning, state-action pairs, could be "responsible" for the TD error at time *t*. Thus, in order to better assign the TD error seen at a given time step, eligibility traces are used

Eligibility traces act as a method to identify which state, or state-action pair, values should be updated when a TD error signal is received. Furthermore, eligibility traces do not simply identify which values should be updated; they determine the degree to which a value should be affected by a given TD error signal. Specifically, in the TD case, in addition to storing the estimated value of each state, an additional eligibility trace parameter, *e(s)*, is stored for each state. At the start of an episode, all of the eligibility trace values are set to zero. Throughout an episode, the eligibility trace values are updated, at each time step, as follows [10]:

$$e_t(s) = \begin{cases} \lambda e_{t-1}(s) & if\ s \neq s_t \\ \lambda e_{t-1}(s) + 1 & if\ s = s_t \end{cases}$$

where λ is the temporal credit assignment parameter, such that $0 \leq \lambda \leq 1$. As can be seen, at a given time step *t*, all of the eligibility trace values are decayed by λ, and the eligibility trace value for the state s_t is incremented by one. Thus, the temporal credit assignment parameter, λ, determines how quickly the eligibility of a state for updating decays

following its activation. Setting λ to zero effectively nullifies the use of the eligibility traces, and yields the *TD(0)* formulation detailed earlier. Thus, the use of eligibility traces, and the temporal credit assignment parameter λ, gives rise to the *TD(λ)* notation.

Similar to the TD case, eligibility traces can also be applied to the Q-learning algorithm, giving rise to *Q(λ)*. In *Q(λ)*, an eligibility trace value, *e(s,a)*, is stored along with each state-action pair, and the eligibility trace values are updated just as in the TD case described above. Implementation of *Q(λ)* can thus be formalized with the following equations, for a given time step *t*:

$$Q(s,a) = Q(s,a) + \alpha\delta_t e_t(s,a) \qquad \forall s,a \in S,A$$

$$\delta_t = [r_t + \gamma \max_a Q(s_t,a) - Q(s_{t-1},a_{t-1})]$$

$$e_t(s,a) = \begin{cases} \lambda e_{t-1}(s,a) & if\ s,a \neq s_{t-1},a_{t-1} \\ \lambda e_{t-1}(s,a) + 1 & if\ s,a = s_{t-1},a_{t-1} \end{cases}$$

where *S* is the set of all possible states, and *A* is the set of all possible actions.

Although Q-learning attempts to estimate the state-action pair value function for the greedy policy, the policy being followed may involve exploratory actions; that is, actions not deemed optimal by the policy being followed, but taken for the purpose of exploring the entire state-action space. These exploratory actions cause the use of eligibility traces to be less then straightforward. This can be illustrated with the decision of whether a large negative value, received after executing an exploratory action, should be passed back to the state-action pair values that unadjusted eligibility traces would suggest are responsible for the outcome. It is certainly unclear how, following an

exploratory action, previous state-action pair values should be impacted by future TD error signals.

Different methods have been proposed about how to handle eligibility traces intelligently for this situation, specifically Watkins's *Q(λ)* [12] and Peng's *Q(λ)* [13]. However, the simplest method for dealing with this problem is, essentially, to ignore it. Specifically, just update the eligibility traces as would be done assuming there were never any exploratory actions. This method is dubbed naïve Q-learning [10], and is used for all of the experiments presented in this work.

Lastly, the formulations given above seem to indicate that TD learning and Q-learning essentially learn value functions mapping states, or state-action pairs, to scalar values. Thus, a straightforward implementation of the *TD(λ)* or *Q(λ)* algorithm is in the form of a single perceptron, where the number of inputs is equal to the number of states, or state-action pairs, and where the weight connecting each input to the perceptron is effectively the value measurement for the given state, or state-action pair (in addition, an eligibility trace value is stored along with each weight). However, it is possible to extend the *TD(λ)*, or *Q(λ)*, algorithm to map states, or state-action pairs, to vectors, and/or to extend the representation to a multi-layer perceptron (MLP). For the general case of *TD(λ)*, or *Q(λ)*, backpropagation in conjunction with eligibility traces can be used to correctly distribute the TD error signals throughout the network. Detailed equations for training a MLP using TD learning are presented in [14].

2.3 Working Memory Toolkit (WMtk)

For most of the experiments conducted in this work, an implementation of Temporal Difference (TD) learning in the form of a single linear perceptron, called the Working Memory Toolkit (WMtk) [15], was used. The WMtk was developed at Vanderbilt University, and is based on the biology of the prefrontal cortex and the midbrain dopamine system. The system has been used in various contexts [16], [17], [18], with a primary goal being adaptive robot control.

At its heart, the WMtk implements a naive Q-learning, *Q(λ)*, system in the form of an optimistic critic whose representation is a single perceptron with a linear activation function. The system is capable of learning preferences, or weights, for different combinations of state-item groupings. The basic input elements of the WMtk are chunks, and the current state. A chunk is simply an abstract data structure that can hold any information desired by the user. Similarly, the state is an abstract data structure that holds whatever information is necessary to describe the current state. Ultimately, neither the chunks nor the state are given directly to the perceptron as input. Two user defined functions, the *chunk mapping function* and the *state mapping function*, perform the task of translating a given chunk or state data structure, respectively, into a real valued feature vector.

The objective of the working memory system is to learn the correct valuation for each state-item grouping. At each time step, the system evaluates all the possible combinations of state-item groupings presented to it, and selects the grouping with the highest valuation from the perceptron as the winner. The items from this winning state-item combination are then placed into the "working memory store." The size of the

working memory store is defined before the system is initialized (typically less then 5 to 7 items). Additionally, at each time step, the current items, or chunks, held in the working memory store are automatically included in the list of items to evaluate for the current time step.

A given chunk feature vector, or multiple chunk feature vectors depending on the size of the working memory store, and the current state feature vector are combined using one of four methods to form what is termed an aggregate feature vector (AFV), which is then presented to the perceptron. The four methods for combining chunk feature vector(s) and the state feature vector are: concatenation, state conjunctive, chunk conjunctive, and complete conjunctive. In the first case, concatenation, the state feature vector and the chunk feature vector(s) are simply concatenated. In the second case, state conjunctive, the chunk feature vectors(s) are first concatenated, and the state feature vector is then conjunctively coded with concatenated chunk vector. In the third case, chunk conjunctive, the chunk feature vector(s) are conjunctively coded, and the state feature vector is then concatenated with the resultant vector. In the fourth case, complete conjunctive, all of the chunk feature vector(s) are conjunctively coded, and the resultant vector is then conjunctively coded with the state feature vector. The selection of the method for forming the AFV thus dictates what can be learned by the working memory system. Specifically, for example, if the concatenation method is used for the formation of the AFV, then it will be impossible for the system to learning anything that depends on the combination of a state and a chunk; as the system will only be able to learn a preference for a given state, or a given chunk, not a combination of the two.

At each time step, chunks are added to a candidate chunk list which is then used by the system to form all the possible state-item groupings. The user can add chunks to the candidate chunk list, additionally all the chunks currently in the working memory store are automatically added. The working memory system then evaluates all the possible permutations of the chunks in the given candidate chunk list, selected at the number of chunks that can be held in the working memory store at a time. Like any Q-learning system, the WMtk learns to output the expected final reward that will be received for each state-item grouping.

A number of parameters control the behavior of the system, namely: the learning rate, the exploration percentage, the temporal credit assignment value, the mean of the initial weights, and the reward function. The learning rate, of course, impacts the magnitude of updates to the weights. The exploration percentage controls the chance that the working memory system will ignore the best valued state-item grouping and instead will chose one of the groupings at random. The temporal credit assignment value affects the number of time steps that a reward is propagated back. The mean of the initial weight controls the initialization point of the system. Finally, the reward function determines the behavior of the system, and ultimately describes the state-item valuations that the system learns to predict.

Chapter 3

Design

3.1 Simulation

The typical Morris water maze environment consists of a one to two meter circular pool of opaque water from which the only escape is a raised platform positioned slightly below the water's surface, ensuring no local cues from the platform are used to guide behavior. The essential elements from this typical Morris water maze setup, combined with information about the setup used by Krichmar et al. [2] to assess the spatial memory of a brain-based device called Darwin X, form the basis for the simulated Player/Stage [19] environment developed by Busch et al. [1]. The simulated environment is shown in Fig. 3.1.

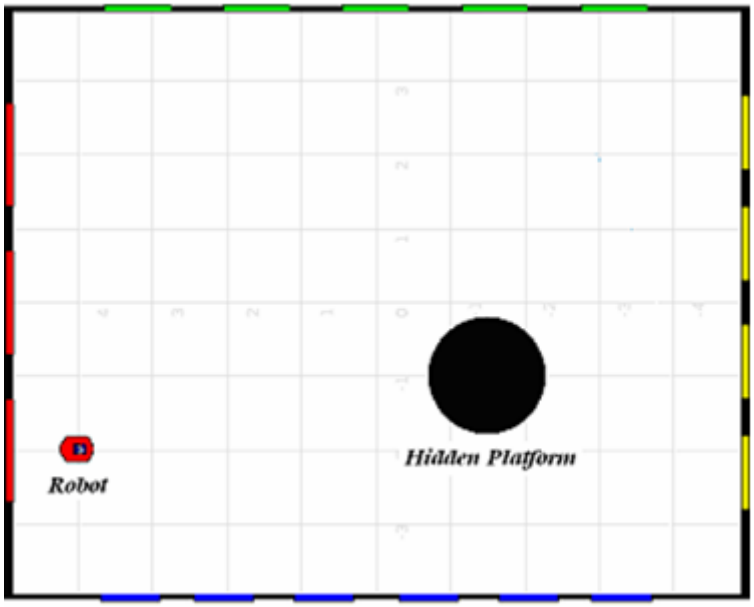

Figure 3.1: Simulated Player/Stage dry water maze environment used by Busch et al. [1]. Gridlines spaced at one meter increments.

The environment is an 8x10 meter rectangle. There are 18 colored panels of varying widths located along the walls. Specifically, there are six blue panels on one wall, five green panels on the wall opposite the wall with the blue panels, four yellow panels on the wall to the right of the blue panels, and three red panels on the wall opposite the yellow panels. These colored panels are used to form the perceptual cues observed by robot.

In addition to the colored panels, a hidden circular platform of radius 0.8 meters is positioned in the simulated environment. This hidden platform represents the only "escape," or goal location, from the simulated environment and is only detectable by the robot when it is directly over the top of the platform.

The robot used in the experiments is a simulated version of a Pioneer P3-DX; a commercially available differential drive robot. The simulated robot is equipped with three cameras, blob finders in Player/Stage, each with a 60 degree field of view. The purpose of the cameras is to detect the colored panels in the simulated environment. In addition, the robot is equipped with a laser range finder that is used for obstacle avoidance. Lastly, the robot is equipped with an infrared sensor which allows it to detect the hidden circular platform. In practice, this is implemented by simply testing if the robot's position is within the platform radius of the goal location. Nevertheless, the robot cannot detect the hidden platform unless it is directly over it.

Using its differential drive system, there are five possible actions, motions, the robot can execute at a given time step: hard left, left, forward, right, and hard right. During these possible actions, the translational speed of the robot is set to 0.3 meters per second, and the rotational speed of the robot is set to 0.4, 0.2, 0.0, -0.2, and -0.4 radians

per second during the hard left, left, forward, right, and hard right actions respectively. Each action is executed for a time period of one second, at which time the motion of the robot is stopped.

The three cameras, blob finders, with which the robot is equipped, allow the robot to detect the colored panels along the edges of the simulated environment. The blob finders are defined to have a maximum range of 14 meters, allowing them to detect the colored panels, if positioned in the camera's field of view, no matter where in the environment the robot may be located. The simulated cameras are specified to have a resolution of 160 pixels by 120 pixels. Blobs found by the blob finders contain information specifying the bounding box, in pixels, of the blob in the simulated camera image space. This information about the bounding box of the blob is used in the formation of the current perceptual feature vector. The formation of this feature vector is described in detail in section 3.5.

The robot uses a simulated laser range finder for obstacle avoidance behavior. This obstacle avoidance behavior is activated if the robot comes within 0.4 meters of a wall in the environment. When this behavior occurs, the robot rotates in place until its directional axis is 30 degrees beyond parallel with the wall. After executing this avoidance behavior, the motion of the robot is stopped, ending the current time step. Given that the maximum number of allowed actions has not been reached, the robot then goes through the sequence of once again acquiring a perceptual vector and selecting another action.

Finally, the hidden platform, as shown in Figure 3.1, is implemented by simply observing the position of the robot using odometry information available in Player/Stage.

If after executing an action, the position of the robot is found to be within the platform radius distance of the specified location of the hidden platform, it is assumed that the robot has found the platform. It should be noted that the odometry information is not used by the Temporal Difference (TD) learning system for locating the hidden platform; it is simply used to implement the infrared sensor functionality in the simulated environment.

3.2 Physical Implementation

In order to implement a physical version of the dry water maze environment used in simulation, a number of features of the environment needed to be adapted. Additionally, a number of challenges had to be overcome to obtain the same functionality of many of the components available in the simulated environment in the physical world. This section details those adaptations, and the solutions to those challenges.

3.2.1 Environment

The simulated dry water maze environment needed to be adapted slightly for implementation in the physical world. Firstly, the environment had to be modified to fit the available space in the lab. Whereas the simulated environment was an 8 by 10 meter rectangle, the physical environment had to be created within the space confines of the lab. Specifically, the physical environment ended up being a 5.26 by 6.06 meter rectangle. Consequently, due to the change in the environment size, the size of the hidden platform was also changed. In simulation the hidden platform had a radius of 0.8 meters and covered 2.5% of the total space. In the physical environment, the hidden platform was

given a radius of 0.41 meters and thus covered 1.6% of the total space. The selection of the smaller platform size relative to the total percentage of the environment space was due to two factors: firstly, less of the total environment space would available to the robot in the physical environment then in the simulation due to the need for a larger obstacle avoidance distance; and, secondly, to more closely match the size of the hidden platform used by Krichmar et al. [2]. These environmental specification differences are laid out in Table 3.1.

TABLE 3.1

SIMULATION AND REAL WORLD ENVIRONMENT STATS

	Simulation	*Real World*
Dimensions (m)	8x10	5.26x6.06
Total Area (m2)	80	31.88
Goal Area (m2)	2.01	0.52
Goal Area /Total Area (%)	2.5	1.63
Avoidance Distance (m)	0.4	0.8

In addition to the size of the environment, the size and positioning of the colored panels along the edge of the environment were also adapted slightly from the simulation. Due to the materials used to construct the colored panels, all of the panels were made to have essentially the same width. Specifically, the blue and red panels have widths of approximately 28 cm, and the green and yellow panels have widths of approximately 25

cm. A diagram of the physical environment with the layout of the colored panels is shown in Figure 3.2. Additionally, pictures taken from the physical environment are shown in Figure 3.3.

Lastly, as can be seen in Figure 3.3, black strips were placed above the green and yellow colored panels in the physical setup. These strips perform no function other then to help with the panel segmentation procedure, as described in section 3.4. Ultimately, besides the size difference of the environment, goal, and colored panels, the physical environment closely matches the simulated environment with respect to the general layout.

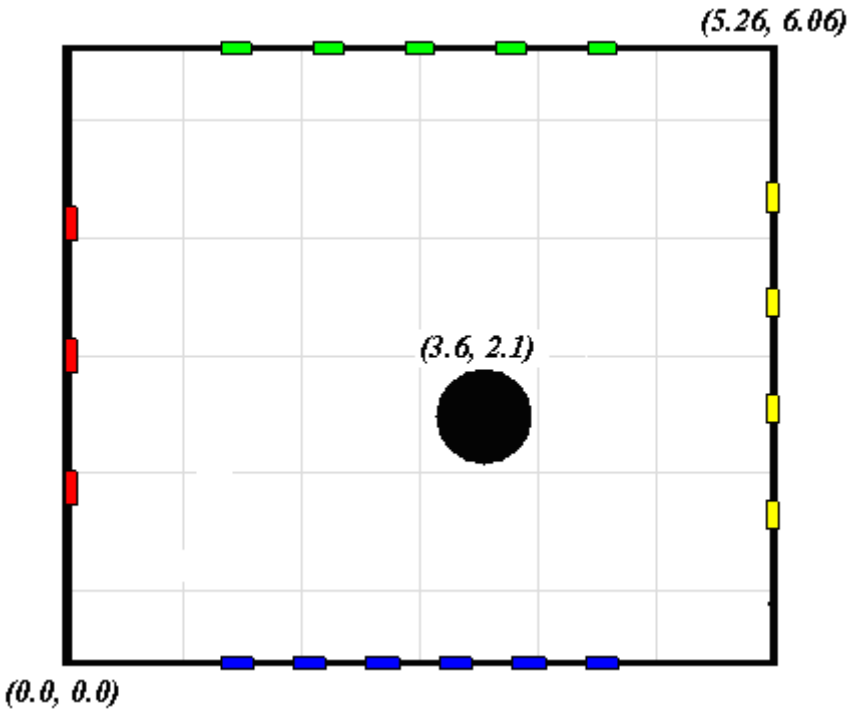

Figure 3.2: Diagram of physical environment and layout of colored panels.

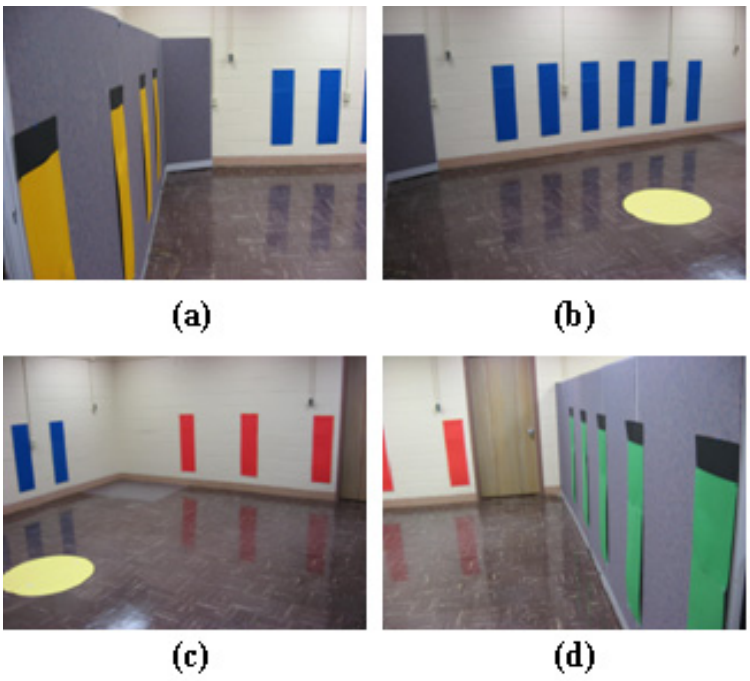

Figure 3.3: Pictures taken from the physical environment, showing the colored panels and hidden platform. Pictures (a) through (d) were taken from left to right around the enclosure.

3.2.2 Robot

As in the simulation, a Pioneer P3-DX was used in the physical setup. However, a number of details about the specific pieces of equipment attached to the robot needed to be adapted from the simulation for implementation on the physical robot. Firstly, whereas in the simulation the robot was equipped with three simulated cameras, the physical robot was equipped with two FireWire web cams with wide angle lenses. Secondly, whereas the simulated robot was equipped with a laser range finder for obstacle avoidance, the physical robot simply used sonar sensors already integrated into the frame for obstacle avoidance. Finally, the robot was equipped with an assembly of four infrared sensors

attached to a Handy Board for detection of the hidden platform. A picture of the Pioneer P3-DX used in the experiments, with the attached equipment described above, is shown in Figure 3.4.

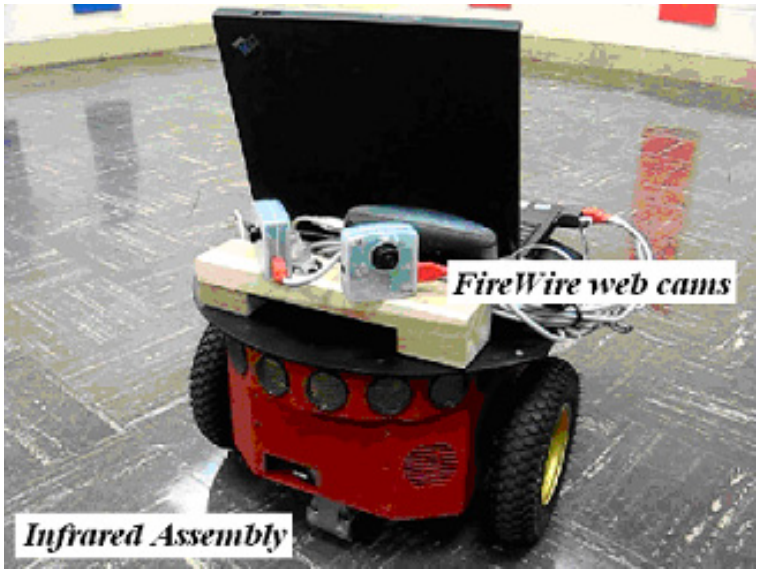

Figure 3.4: Picture of the Pioneer P3-DX robot used in the physical experiments. Two FireWire web cameras are mounted to a wooden base on the front of the robot. An infrared sensor assembly can be seen beneath the front of the robot.

3.2.2.1 Motion

As a result of using a P3-DX robot in the simulated environment, the possible motions of the robot did not need to be altered for implementation in the physical environment. As in the simulated environment, the physical robot can execute one of five possible actions at a given time step: hard left, left, forward, right, and hard right. In the simulation, during these possible actions, the translational speed of the robot was set to

0.3 meters per second, and the rotational speed of the robot was set to 0.4, 0.2, 0.0, -0.2, and -0.4 radians per second during the hard left, left, forward, right, and hard right actions respectively. These same specifications were used to define the five possible actions the physical robot could take at a given time step.

In the simulated environment, each action was executed for one second, at which time the motion of the robot was stopped while the robot acquired a new perceptual vector and determined what new action to execute. However, whereas this perceptual acquisition and processing took only milliseconds in the simulated environment; this same process required slightly less then one second in the physical environment. Thus, due to approximately one second of processing time being needed to acquire and act on a new perceptual vector, in the physical environment each action was executed for 50 milliseconds (during which time obstacles could be detected) plus this approximately one second needed for the perceptual acquisition and processing. Furthermore, the motion of the robot was not stopped between actions. These changes resulted in a much smoother motion from the physical robot, and, ultimately, approximately the same time required per action, or step, as in the simulated environment.

3.2.2.2 Cameras

The cameras used on the physical robot were Unibrain Fire-I FireWire digital color cameras. The cameras capture 640 by 480 color images in uncompressed RGB format. Wide angle lenses were used on the cameras which gave an approximate horizontal field of view of 90 degrees for each camera. The cameras, mounted on the P3-DX robot, are shown in Figure 3.4.

3.2.2.3 Ultrasonic Sensors

The Pioneer P3-DX robot comes equipped with a ring of 16 sonar sensors. Given that space on top of the robot was needed for cameras, and that the laser range finder requires a large amount of power, it was decided that the sonar sensors on the robot would be used for obstacle avoidance. Specifically, the front eight sonar sensors are used for basic obstacle avoidance. Due to the latency of the readings from the sonar sensors, the obstacle avoidance behavior of the robot is triggered by a reading of 0.8 meters or less from any of the front eight sonar sensors.

As accurate position and orientation information is not available to the physical robot, the obstacle avoidance behavior from the simulation, detailed in section 3.1, is approximated using readings from the sonar sensors. Specifically, the robot rotates away from a detected obstacle (known to be a wall in the environment) until readings from its sonar sensors indicate it is angled at least 30 degrees away from the obstacle.

3.2.2.4 Infrared Sensor

To detect the hidden platform in the physical environment, an infrared sensor assembly attached to a Handy Board was used. The infrared assembly consists of four individual infrared emitter-detector sensors, all of which need to be activated to constitute a detection of the hidden platform. The Handy Board was connected to the laptop computer used to control the robot through a serial connection, and simply transmits a notification signal to the laptop whenever all four infrared sensors are simultaneous activated. The infrared assembly is visible in Figure 3.4.

3.3 Position Fix

In the simulated environment, where perfect odometry information is readily available, resetting the robot after each episode is as simple as telling the robot to move to a given position. In the physical environment, however, the robot's odometry suffers from considerable error. Although the odometry is accurate enough to keep a rough path for display purposes of the robot's movements during an episode, it is not accurate enough for use in repositioning the robot. Furthermore, as the sonar sensors used for obstacle avoidance are not suitable for localization, another solution to repositioning the robot after each episode had to be developed.

The solution relies on the fact that the robot's odometry is accurate enough that the estimated orientation after a single episode is within plus or minus approximately 10-15 degrees. Given this level of accuracy, following the end of an episode, the robot is instructed to turn toward an absolute heading of 135 degrees, and is then instructed to go forward until a wall is encountered. Depending on readings from the robot's sonar sensors, the robot then follows the detected wall towards the right or left. This causes the robot to "funnel" itself into the corner of the environment. This process is illustrated in Figure 3.5. Once the robot has successfully navigated into the corner, the sonar sensors are used to obtain an accurate position fix to within a few inches on the x and y axis, and to within 5-7 degrees on the orientation. This level of localization precision is accurate enough for the robot, in general, to successfully reset its position after every run.

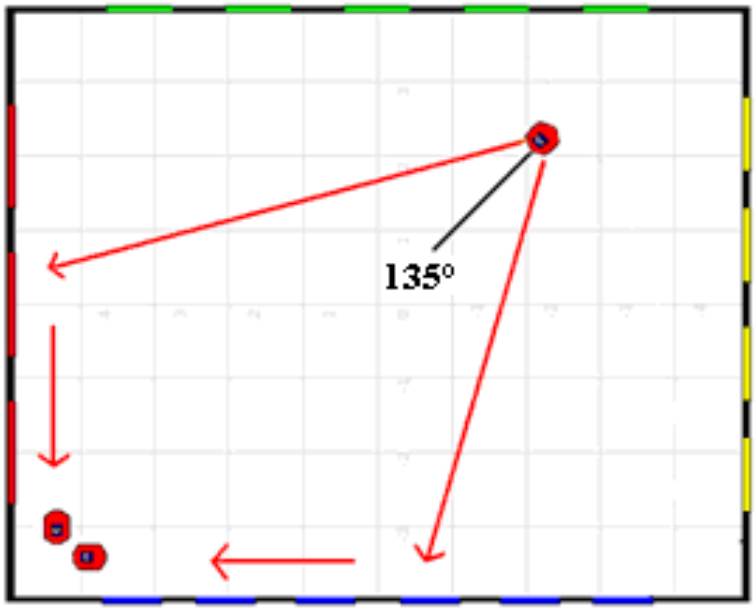

Figure 3.5: Diagram of the position fix procedure.

Although the solution is straightforward, there are two main drawbacks. First, a sizable amount of time is wasted between episodes while the robot fixes it position. Second, although the robot gets a reasonably good position estimate, the error is still large enough to cause a noticeable amount of variability in the positioning of the robot for a given starting location. As compared to the simulated environment, this is an additional source of error.

Simulations, detailed in section 4.2.1.2, were conducted to evaluate the possible impact of this inaccuracy in starting location positioning. Specifically, simulations were performed which added randomly distributed errors of plus or minus 0.5 meters and plus or minus 5 degrees to the x/y position and orientation of the robot's starting positions respectively. Results of these simulations showed that the inclusion of such error in the robot's starting positions significantly increased the average number of moves needed to

find the hidden platform given 100 training episodes. Analysis of the results seemed to indicate that this decrease in performance was a result of the fact that even small changes to a starting position cause the robot to observe different SOM nodes during the initial step of the episode. Thus, variability in the starting locations requires the robot to learn correct action preferences for a larger number of SOM nodes from which it might start; whereas in the simulated environment the robot always begins each episode from the same SOM node for a giving starting position.

3.4 Panel Segmentation

In the simulated environment, reliable detection of the colored panels is not an issue. The simulated cameras do not suffer from the problems encountered using real cameras in a physical environment. These problems include limited resolution, lens distortion, changing lighting conditions, and the task of ultimately segmenting the panels in the images. Due to these problems, developing a reliable panel extraction process itself proved to be a challenging task.

On the physical robot, the panel detection process consists of first removing lens distortion from the captured images, discarding the bottom halves of the images, performing median filtering on the images, and using a combination of rules in multiple color spaces for segmentation. Finally, the knowledge that the panels are vertical in the images is used to identify and extract the number of panels. A number of functions from the Open Source Computer Vision Library [20] are used for the image processing steps.

A large amount of distortion is present in the captured images due to the use of wide angles lenses. Therefore, in order to obtain accurate information about the height of

the colored panels (necessary for the creation of the perceptual feature vectors as described in section 3.5), the first step of the panel extraction process involves adjusting the captured images for lens distortion. To achieve this, a rough estimate of the lens distortion parameters, as modeled in OpenCV, was obtained by adjusting the parameters until lines in a handful of test images were no longer distorted.

After adjusting the captured images for distortion, the bottom halves of the images are discarded, and only the top halves are used for panel detection. The discarding of the bottom halves of the images helps to eliminate panel detection problems due to reflections from the floor, or the color of the hidden platform. In addition, the bottom edges of the panels are always located in the bottom half of the images, meaning they always extend past the bottom of the top half. This information is later exploited in the panel segmentation process.

Repeated median filtering is then performed to smooth the images while preserving edges. After this preprocessing step, the images are transformed into the Hue Saturation Value (HSV) color space, and a set of RGB and HSV rules are used to segment the images. Next, using the fact that the panels, if present, extend to the bottom of the images (now only the top half of the original images), and that they are vertical in the images, the sides of any panels in the images are identified. Given the location of the sides of each panel, and the color, the top edges of the panels are identified by moving up from the bottom of the image towards the top until the average color no longer satisfies the RGB and HSV rules defining the color of the panel.

An illustration of the panel segmentation process is shown in Figure 3.6.

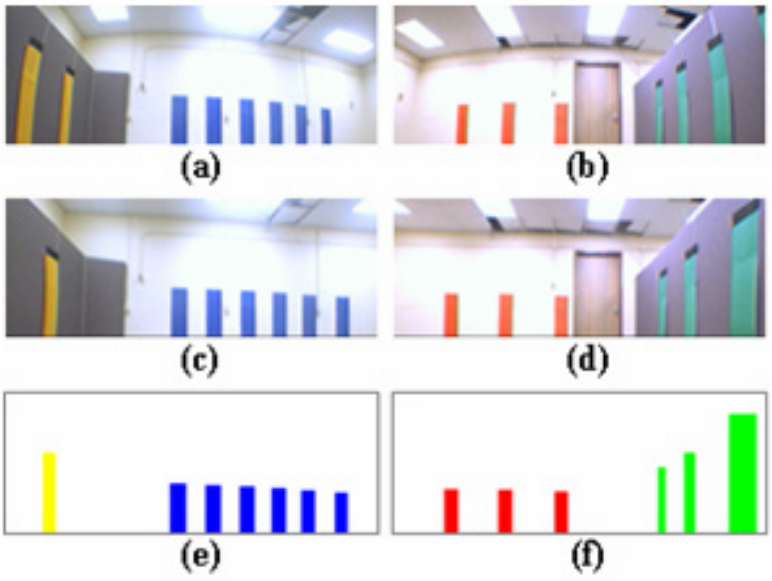

Figure 3.6: Illustration of the panel detection process from the two FireWire webcams on the P3-DX robot. Images (a) and (b) are the top halves of raw images captured from the left and right cameras respectively. Images (c) and (d) are the corrected and filtered images from the left and right cameras respectively. Finally, images (e) and (f) show the resulting panel segmentation.

3.5 Self-Organizing Map

In [1], Busch et al. used a Self-Organizing Map (SOM) to discretize the perceptual space. The SOM allowed the large number of possible perceptual states to be reduced to a useable number for the tested spatial memory systems.

Initially, panel segmentation information collected from each of the cameras is used to generate a feature vector. As each camera has the possibility to encounter any of the 18 colored panels in the environment, the vector formed for each camera consists of 18 bins. If a panel of a certain color is detected, the height of that panel in pixels is stored in the first empty bin corresponding to the color of the panel, as the robot has no way to determine which of the panels of the given color it is observing. Panels are processed

from left to right across an image. Thus, in the simulated environment, the robot is equipped with three cameras, resulting in a 54 dimensional feature vector. In the physical environment, the robot is equipped with two cameras, resulting in a 36 dimensional feature vector.

The SOM(s) used in the simulated experiments were trained from 10,000 perceptual vectors collected from random points in the environment. The SOFM(s) used in the real world experiments were trained from approximately 7,400 perceptual vectors collected by letting the robot randomly roam throughout the environment with an obstacle avoidance behavior for approximately two hours. For the experiments conducted in the simulation and physical environments presented in this work, unless otherwise stated, an 8x8 SOFM was used, resulting in 64 possible perceptual states.

3.6 Working Memory Toolkit Usage

As previously stated, for the majority of the experiments conducted in this work, the Temporal Difference (TD) learning implementation found in the Working Memory Toolkit (WMtk), detailed in section 2.3, was used. This section details the usage of the WMtk in the context of the dry water maze environment.

Firstly, it should be stated that in the case of the dry water maze environment, a spatial memory task, the learning of spatial memory, and not working memory, was being investigated. Thus, although the WMtk was used, it was used purely from the TD implementation aspect, and not as a general working memory system. Due to this fact, the WMtk was used with a working memory store of size one.

Finally, it should be noted that the use of the WMtk for the dry water maze environment, as documented in this section, was used by Busch et al. [1].

3.6.1 State Mapping Function

One of the steps necessary in the use of the WMtk is the creation of a state mapping function which maps an abstract state data structure to a real valued feature vector. Given the discretized perceptual space resulting from the use of a Self Organizing Map (SOM), see section 3.5, the creation of this state mapping function is relatively straightforward.

Firstly, the abstract state data structure simply holds the number of the SOM node to which the current perceptual feature vector maps. Secondly, the state mapping function simply generates a real valued state feature vector with a dimensionality equal to the number of possible states, such that the current state, as indicated by the state data structure, is set to 1, while all the other values are set to zero. Thus, if, for example, a 64 node SOM is used, then the state mapping function generates a state feature vector with dimensionality 64, where the dimension corresponding to the current state, as indicated by the abstract state data structure, is set to 1, while all the other values are set to zero.

3.6.2 Chunk Mapping Function

In addition to the state mapping function, it is also necessary, for the use of the WMtk, to create a chunk mapping function which maps an abstract chunk data structure to a real valued feature vector. In the case of a control task, these chunks generally take the form of actions. In the case of the dry water maze environment, there are five possible

actions that the robot can take at a given time step, thus yielding five possible chunk values.

Firstly, in the dry water maze environment, an abstract chunk data structure simply holds the number (1-5) of the action it represents. Secondly, the chunk mapping function simply generates a real valued feature vector of dimension five, such that the dimension corresponding to the action held in the chunk data structure is set to 1, while the other dimensions are set to zero.

3.6.3 Aggregate Feature Vector (AFV) Formation

Given that the state and chunk mapping functions have been defined, it is also necessary to define how the real valued state and chunk feature vectors output by the two functions are combined before being presented to the perceptron that is the TD system. In the case of the dry water maze environment, where the working memory store size was set to one, state conjunctive coding was used for the formation of an AFV. Specifically, to generate an AFV, a given chunk feature vector and the current state feature vector are conjunctive coded together to yield an AFV. Thus, if, for example, the chunk feature vector length is five, and the state feature vector length is 64, the resulting AFV that is presented to the TD system will be of length 320; where 320 corresponds to the total number of possible state-action pairs, and for any given AFV only one of those 320 dimensions will be non zero. The use of this method for the formation of AFVs effectively allows the system to learn valuations of all possible state-action pairs.

3.6.4 Reward Function

When the Working Memory system is first initialized, the preferences, or weights, of the system are set randomly. During training, a reward signal is given to the system, both during and at the end of training episodes, which the system attempts to maximize by adjusting its state-action preferences (as described in sections 2.2 and 2.3). This signal is represented by a reward function that is called at the end of each time step, and, for most of the experiments described in this work, is defined as:

$$\text{reward} = \begin{cases} -5.0 & \text{if } c = 0 \\ -5.0 & \text{if obstacle detected} \\ 1.0 + \dfrac{(m-n)}{10} & \text{if goal detected} \\ -5.0 & \text{if } n \geq m \\ 0.0 & \text{otherwise} \end{cases}$$

where c is the current number of chunks in the Working Memory system's store, m is the maximum number of moves allowed per episode, and n is the number of the current move in the current episode.

Thus, the robot is rewarded at the end of each training episode depending on whether it has or has not found the hidden platform. If the platform has been found, a positive reward inversely proportional to the number of moves required to find the platform is given. If the hidden platform has not been found, then a fixed negative reward is given. In addition to the delayed rewards, the robot is given an immediate negative reward when the obstacle avoidance behavior is initiated, and when the learning system selects none of the five possible actions (c = 0). (When this occurs, one of the five

possible actions is chosen at random.) These sparse measures of performance are the only feedback the robot receives in learning a mapping between SOM nodes and actions.

In a handful of experiments detailed in later sections, variations of this reward function are used. When this is the case, the specific reward function used will be described.

3.7 Modification of WMtk for "Forgetting"

As detailed later in section 4.2.2, the original implementation of Temporal Difference (TD) learning, as encapsulated by the WMtk, was found to be severely lacking in its ability to adapt in non-stationary environments. For this reason, the learning algorithm of the WMtk was modified to achieve better performance in non-stationary environments. This modification came in the form of active "forgetting" based on past rewards received.

Specifically, a "short term" reward and a "long term" reward, calculated based on rewards received at the end of episodes during the current trail, are used to control a "forgetting" process. This process can be formalized into the equations below. Essentially, whereas the original learning equation, in terms of weight updates, implemented in the WMtk is:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha\delta_t e_{ij}(t)$$

where α is the learning rate, δ_t is the TD error at time t, and $e_{ij}(t)$ is the eligibility trace for w_{ij} at time t; with "forgetting" the weight update equation becomes:

$$w_{ij}(t+1) = \left[w_{ij}(t) + \alpha\delta_t e_{ij}(t) - \mu\right]\left(1 - f_p\right) + \mu$$

where μ is the initial mean of the weights for the system, and f_p is defined as follows:

$$f_p = \begin{cases} 0 & if\ r_s \geq r_l - \varepsilon \\ (r_l - r_s)\eta_f & otherwise \end{cases}$$

where r_l and r_s are the long term and short term rewards respectively, ε is a small positive constant, and η_f is the forgetting rate.

Essentially, under normal circumstances, where $r_s \geq r_l - \varepsilon$, the weight update is unchanged from that originally implemented in the WMtk. However, if $r_s < r_l - \varepsilon$ then the system actively "forgets" by moving all weights closer to the initial mean value μ. Furthermore, the degree to which this forgetting takes place is controlled by both the difference between r_l and r_s, and the forgetting rate η_f.

The long term and short term rewards, r_l and r_s, are calculated based on rewards received at the end of each episode during a trial. Specifically, the long and short term rewards are calculated by taking an average of sliding medians over some number of past rewards received:

$$r_{l/s} = \frac{1}{N_{l/s} - M_{l/s} + 1} \sum_{k=0}^{N_{l/s} - M_{l/s}} median(R_{t-N_{l/s}+k}, \ldots, R_{t-N_{l/s}+k+M_{l/s}})$$

where N is the number of episodes over which past rewards will be used to calculate the reward, M is the size of the window over which medians will be calculated, R is the reward signal consisting of rewards received at the end of previous episodes, and t is the current episode.

For experiments conducted in section 4.2.2.2, N was set to 100 and 30 for the long and short term rewards respectively, M was set to 20 for both rewards, ε was set to 0.5, and η_f was set to 0.09.

3.8 TD(λ) Multi Layer Perceptron Implementation

In addition to the Working Memory Toolkit (WMtk), a Multi Layer Perceptron (MLP) implementation of the *TD(λ)* algorithm, suitable for use with *Q(λ)* algorithm as well, was developed and tested for the dry water maze environment. The implementation is as described in [14], and could be instantiated for any number of hidden layers, with any number of nodes (including any number of output nodes).

The main purpose of implementing a MLP version of the *TD(λ)* algorithm was to evaluate the ability of such a system to learn to find the hidden platform in the dry water maze environment, without the need for a SOM discretization of the perceptual space. Specifically, the MLP was implemented to test the idea of using the raw perceptual vectors obtained from the cameras on the robot as input of the current state.

Results from various experiments making use of this MLP implementation are presented in section 4.2.1.5.

3.9 General Experimental Procedure

For the experiments presented in this work, in general, simulation episodes proceeded as follows: at each time step, the robot determines its current perceptual state, as represented by a 54 dimensional feature vector, using information about the colored blobs visible to its blob finders. (This process in described in detail in section 3.1.) This perceptual feature vector is then compared to the SOM nodes to determine the closest node. The temporal difference learning system then uses the closest SOM node as the current state of the robot. Based on this state, the TD system then selects one of the five possible actions, which are described in section 3.1. This process is then repeated for a maximum of 50 moves or until the robot detects the hidden platform, at which time the current episode is ended, and the robot's position is reset. During each such episode, the system receives rewards as defined in section 3.6.4. The general procedure for experiments conducted in the physical environment is effectively the same as described above for experiments conducted in the simulated environment.

Performance of the system is evaluated by either observing the performance of the system during training trails, specifically the number of moves per episode, or by performing some number of evaluation episodes, that is episodes during which learning and exploration are inhibited, both before and after training in order to measure improvement.

Chapter 4

Experimental Results and Analysis

4.1 Physical Environment

Two experiments were developed to evaluate the ability of the physical robot to learn the necessary associations between SOM nodes and actions to locate the hidden platform in the dry water maze environment. These two experiments were a single corner experiment and a four corner experiment. For both of these experiments, each training episode consists of a maximum of 51 moves. For each move the robot determines which SOFM node it is currently in, and then selects one of five possible actions, or a random action if it selects none of the five, to take. If the robot finds the goal before 51 moves have been made, the run is ended.

In addition to tracking the performance of the robot during training, in a handful of experiments evaluation episodes were also used to observe the performance of the system. For these evaluation episodes, the exploration and learning of the system were inhibited.

4.1.1 Single Corner

For this experiment, each training episode consists of starting the robot at a single fixed starting location, as illustrated in Figure 4.1. During each episode, the robot is allowed a maximum of 51 moves to locate the hidden platform. Once 51 moves have been executed, or the hidden platform has been found, the episode is ended. The robot is then repositioned to the single starting location for the next episode.

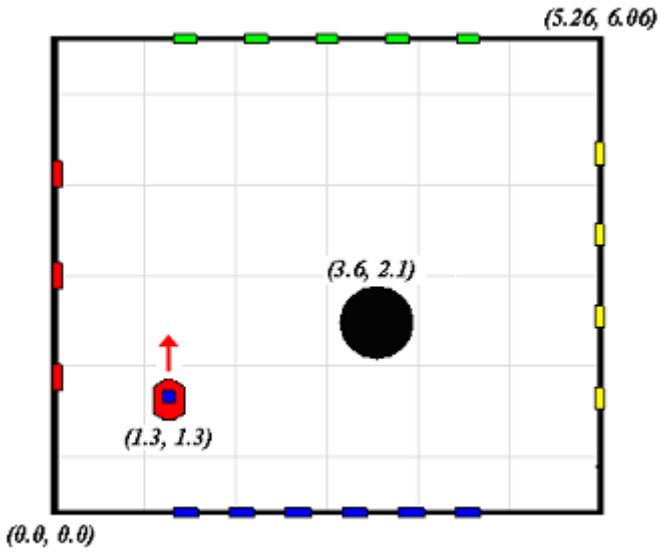

Figure 4.1: Diagram of the single corner water maze task showing the starting location and positioning of the robot, along with the location of the hidden platform. Coordinates are in meters.

The results of one single corner water maze experiment are shown in Figure 4.2 and Table 4.1, and are typical of those obtained during other tests. Figure 4.2 (a) shows the number of moves per episode during a training sequence of 100 episodes. As can be seen, the robot fails to find the platform during most of the early training episodes. Figure 4.2 (b) shows an example path of the robot during one of these early training episodes. Within approximately 20 training episodes, however, the robot appears to learn a path to the platform. Figure 4.2 (c) shows the path of the robot during episode 22. As the training sequence continues the robot does fail to locate the platform during certain episodes. Figure 4.2 (d) shows the path of the robot during episode 64, an episode in which it fails to find the platform. The path in Figure 4.2 (d) is typical of many failed runs later in the

training process. The robot generally follows a successful path to the platform but just misses it. More information is needed to identify the exact reason. However, these failures could be due to the exploration of the TD system, or could be caused by some combination of parameters such as goal size, SOM size, sensory uncertainties, etc. Clearly, though, the frequency of episodes during which the platform is not found appears to decrease as the training process progresses.

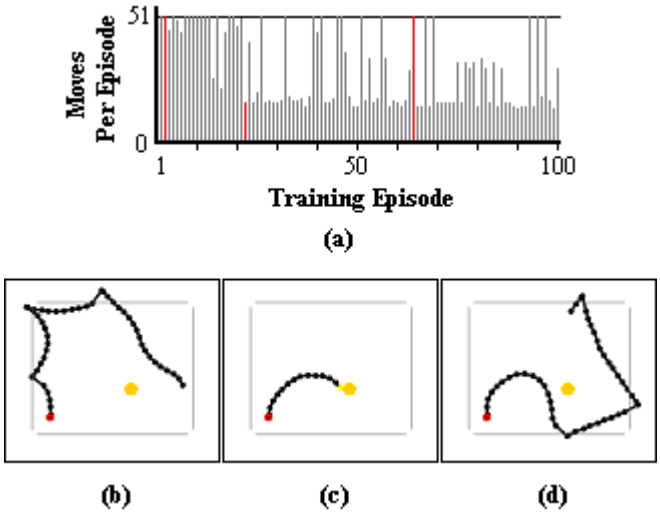

Figure 4.2: (a) Plot of moves per episode during 100 training episodes for the typical single corner water maze task presented here. The maximum number of moves allowed per episode is 51. (b) Path of robot during episode 2. (c) Path of robot during episode 22. (d) Path of robot during episode 64. The displayed paths are highlighted in red in (a). The inner box in (b), (c), and (d) shows the distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training and evaluation episodes. This information is not used by the learning system.

To better characterize the improvement of the system, the robot was evaluated both before and after the training episodes. During the evaluation episodes the exploration and learning of the TD system are inhibited. Table 4.1 shows the results of these evaluations. Noticeable improvement can be seen after the 100 training episodes as compared with the results obtained before training, indicating the robot has indeed learned at least some set of state-action associations necessary for navigating to the platform.

TABLE 4.1
SINGLE CORNER EVALUATION RESULTS

	Before Training	*After 100 Training Episodes*
Evaluation Episodes	40	40
Average Moves per Episode	48.5	23.6
Episodes Platform not Found	32	8

Results from additional single corner training sequences can be found in Appendix A.

4.1.2 Four Corner

For this experiment, each training episode consists of starting the robot at one of four starting locations, as illustrated in Figure 4.3. The four starting locations are visited in the following sequence: (1.3, 1.3), (4.76, 1.3), (4.76, 3.96), and (1.3, 3.96). During each episode the robot is allowed up to 51 moves to locate the hidden platform. Once 51 moves have been executed, or the hidden platform is found, the episode is ended. At the end of each episode the robot is repositioned to the next starting location in the sequence as described above.

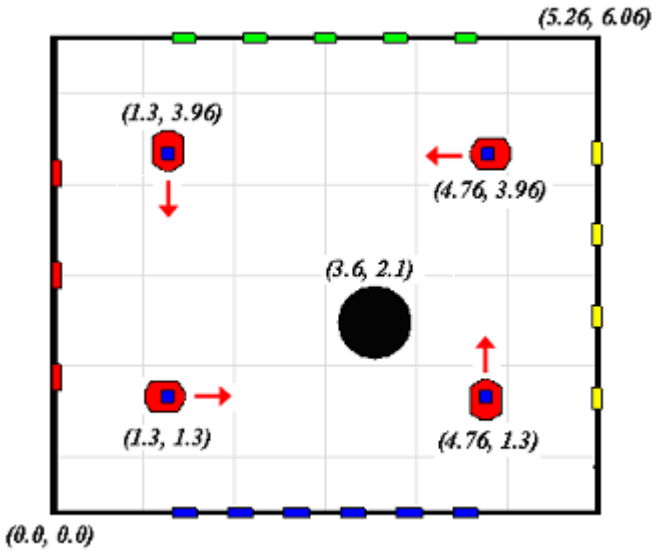

Figure 4.3: Diagram of the four corner water maze task showing the starting locations and positions of the robot, along with the location of the hidden platform. Coordinates are in meters.

The results of one four corner water maze experiment are shown in Figure 4.4 and Table 4.2, and are typical of those obtained during other tests. Figure 4.4 (a) shows a moving average of the number of moves per episode during a trial of 100 episodes. The moving average is taken over the current episode and the previous three. A moving average of four episodes is used to display the change in the number of moves per episode over all four starting locations.

As in the single corner task, the robot fails to find the platform during many of the early training episodes. Figure 4.4 (b) shows the paths of the robot from each starting location for episodes 2-5. After approximately 70 training episodes, the robot appears to learn a path to the platform from each of the starting locations. Figure 4.4 (c) shows the paths of the robot from each starting location for episodes 69-72. Here again, however, as the training sequence continues the robot does fail to locate the platform during certain episodes. Figure 4.4 (d) shows the paths of the robot from each starting location for episodes 91-94, in which the robot fails to locate the platform from the lower right starting location during episode 94. As before, these failures could be due to the exploration of the system, or could be caused by some combination of parameters such as goal size, SOM size, sensory uncertainties, etc.

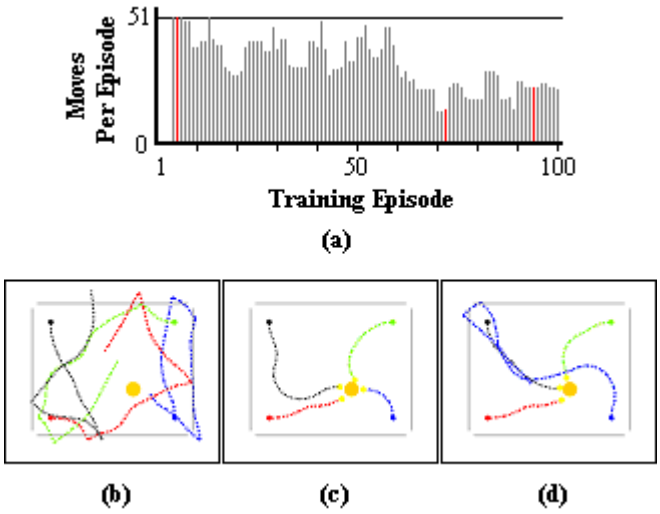

Figure 4.4: (a) Plot of the four episode moving average of moves per episode during 100 training episodes for the typical four corner water maze task presented here. The moving average is of the current episode and three previous episodes, and is used to show the change in the number of moves from the four combined starting locations. The maximum number of moves allowed per episode is 51. (b) Paths of robot during episodes 2-5. (c) Paths of robot during episodes 69-72. (d) Paths of robot during episodes 91-94. The average of the displayed paths is highlighted in red in (a). The inner box in (b), (c), and (d) shows the distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training and evaluation episodes. This information is not used by the system.

As in the single corner task, the robot was evaluated before and after the training episodes to monitor improvement. During the evaluations episodes the exploration and learning of the TD system are inhibited. For the four corner task, the evaluation episodes are carried out from all four starting locations in the same sequence as during training. The evaluation consists of 40 total episodes, thus 10 evaluation episodes are conducted from each corner. Table 4.2 shows the results of these evaluations. Here again, noticeable improvement can be seen after the 100 training episodes as compared with the results

obtained before training, indicating the robot has correctly learned at least some set of node/action associations necessary for navigating to the platform.

TABLE 4.2
FOUR CORNER EVALUATION RESULTS

	Before Training	*After 100 Training Episodes*
Evaluation Episodes	40	40
Average Moves per Episode	46.0	20.1
Episodes Platform not Found	30	3

Results from additional four corner training sequences can be found in Appendix B.

4.2 Simulation

The simulated environment used by Busch et al. [1], described in section 3.1, was used to conduct a number of experiments for the purpose of evaluating and testing various parameters and ideas related to the TD implementation. Generally, the experiments can be broken down into two groups: those conducted in a stationary environment, i.e. an environment in which the hidden platform location is fixed, and those conducted in a non-stationary environment, sections 4.2.1 and 4.2.2 respectively.

During the experiments, unless otherwise noted, the robot was positioned at one of four starting locations at the beginning of each episode, as illustrated in Figure 4.5. During each episode the robot was allowed a maximum of 50 moves to locate the hidden platform. Once 50 moves have been executed, or the hidden platform is found, the episode is ended.

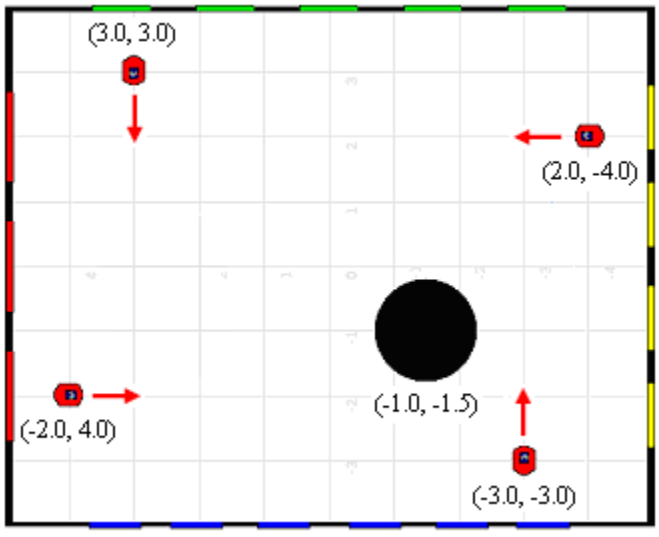

Figure 4.5: Diagram of starting positions and goal location for experiments conducted in section 4.2.

4.2.1 Stationary Environments

Initially, a large number of experiments were conducted in the simulated environment with the goal of determining the optimum values for the parameters affecting the performance of the Working Memory Toolkit (WMtk) and the optimum size of Self Organizing Map (SOM) used to discretize the perceptual space. Significant testing of these parameters was not conducted as part of the work in [1]. The results from these experiments are detailed in section 4.2.1.1.

Section 4.2.1.2 details the results of experiments done to evaluate the effect of imprecise starting positions, as experienced in the physical environment, on the performance of the temporal difference (TD) system.

Finally, sections 4.2.1.3 - 4.2.1.5 describe the results of experiments done to evaluate the performance of the system given the following modifications: a constant reward function, incorporation of simulated hippocampal place cells into the formation of the state feature vector, and the use of a multi layer perceptron (MLP) implementation of the TD(λ) algorithm to eliminate the need for a SOM.

4.2.1.1 SOM Size and WMtk Parameter Evaluation

A large number of experiments were conducted to evaluate the affect of various parameters on the performance of the TD system in the simulated dry water maze environment. The parameters tested included: SOM size, temporal credit assignment value, learning rate, and exploration percentage.

The size of the SOM used to discretize the perceptual space determines the number of states in the water maze environment. For the experiments conducted in [1], a

20 by 20 SOM was used which resulted in 400 perceptual states. As the number of weights in the TD system, and thus the complexity, is directly proportional to the number of states, a smaller SOM, size 8 by 8, was tested to determine whether it yielded better performance in the water maze environment.

The temporal credit assignment value, described in section 2.2, determines how rewards given to the system are propagated back to weights activated during a training episode. Specifically, the eligibility trace value of an activated weight is decayed by the temporal credit assignment value at each time step. For the experiments conducted in [1], the default WMtk value of 0.7 was used. For this work, six temporal credit assignment values from 0.7 to 0.95 were tested to evaluate their affect on the performance of the TD system in the water maze environment.

The learning rate, described in section 2.2, determines the magnitude of the updates to the weights of the TD system. Specifically, the eligible TD error calculated at a time step for a given weight is multiplied by the learning rate before being applied to the weight. For the experiments conducted in [1], a learning rate of 0.01 was used. For this work, three learning rate values, 0.001, 0.01, and 0.1, were tested to evaluate their affect on the performance of the TD system in the water maze environment.

The exploration percentage, described in section 2.3, determines how often the TD system will ignore its current action preference and instead choose a different action. In [1], the default WMtk exploration percentage of 0.05 was used. For this work, two exploration percentages, 0.05 and 0.10, were tested to evaluate their affect on the performance of the TD system in the water maze environment.

In order to evaluate the impact of these four variables on the performance of the TD system, 25 trials, each consisting of 100 training and 100 evaluation episodes, were conducted for all 72 possible combinations of values of the variables described above. For these experiments, during both the training and evaluation phases, the robot was randomly positioned at one of the four starting locations shown in Figure 4.5 at the beginning of each episode. Although the sequence of starting locations was random, it was controlled such that during every 100 episodes the robot started from each location 25 times. Results of these experiments are shown in Tables 4.3 and 4.4. Each entry represents the number of moves per episode, during evaluation, averaged over 25 trials.

TABLE 4.3

PARAMETER EVALUATION RESULTS - 20 X 20 SOM

(RESULTS ARE AVERAGE MOVES PER EVALUATION EPISODE)

Exploration Percentage = 0.05							
		Temporal Credit Assignment Value					
		0.70	**0.75**	**0.80**	**0.85**	**0.90**	**0.95**
Learning Rate	**0.001**	21.5	18.3	16.8	14.3	13.7	14.8
	0.010	16.0	16.8	13.8	13.6	15.0	15.5
	0.100	20.2	20.8	21.6	20.1	18.4	32.9
Exploration Percentage = 0.10							
		Temporal Credit Assignment Value					
		0.70	**0.75**	**0.80**	**0.85**	**0.90**	**0.95**
Learning Rate	**0.001**	20.8	22.2	17.3	18.0	13.1	15.6
	0.010	16.4	15.7	16.2	15.0	17.4	16.9
	0.100	18.5	22.4	20.6	18.7	19.2	33.7

TABLE 4.4

PARAMETER EVALUATION RESULTS – 8 X 8 SOM

(RESULTS ARE AVERAGE MOVES PER EVALUATION EPISODE)

Exploration Percentage = 0.05							
		Temporal Credit Assignment Value					
		0.70	**0.75**	**0.80**	**0.85**	**0.90**	**0.95**
Learning Rate	**0.001**	18.0	16.9	15.1	12.7	11.5	11.6
	0.010	17.1	15.5	13.9	14.7	13.0	14.8
	0.100	15.2	16.4	17.4	15.9	15.3	33.4
Exploration Percentage = 0.10							
		Temporal Credit Assignment Value					
		0.70	**0.75**	**0.80**	**0.85**	**0.90**	**0.95**
Learning Rate	**0.001**	22.5	18.9	13.9	15.1	12.7	11.9
	0.010	15.1	14.1	***12.9***	***13.6***	14.4	14.1
	0.100	17.5	14.4	15.6	17.0	16.3	33.7

From the results shown in Tables 4.3 and 4.4, perhaps the most noticeable result is the performance of the system given the largest learning rate and the largest temporal credit assignment value. In all four cases of this combination, the resulting performance of the system, an average of approximately 33 moves per episode during evaluation, was essentially equal to results obtained before training, with random weights; which seems to indicate that this combination of values makes the system unstable. Furthermore, it appears that the combination of the smallest learning rate and the smallest temporal credit assignment value, though not yielding unstable performance such as the previous combination, also yields poor results in all four cases. Ultimately, besides these to noticeable effects, the performance of the system seems rather invariant, for the most part, to the changes in parameter values. That said, it does appear that overall, the 8 by 8 SOM tended to yield better results then the 20 by 20 SOM.

As a result of the lack of conclusive evidence demonstrating a clear and obvious favorite for the best combination of parameter values, the following values for the tested parameters were used in all subsequent experiments: a SOM size of 8 by 8, an exploration percentage of 0.10, a learning rate of 0.01, and a temporal credit assignment value of 0.83.

4.2.1.2 Starting Position Variability Evaluation

As described in section 3.3, the lack of error free odometry in the physical environment leads to imprecise positioning of the robot at the beginning of each episode, a source of error not found in the simulated environment. In order to determine the impact of this imprecise positioning on the performance of the TD system, experiments were conducted in the simulated environment during which uniform random errors of plus or minus 0.5 meters and plus or minus 5 degrees were added to the robot's starting position at the beginning of each episode. Specifically, 25 trials, each consisting of 100 training and 100 evaluation episodes, were conducted during which the error described above was added to the robot's starting positions. During these experiments, for both the training and evaluation phases, the robot was positioned, before the addition of error, at one of the four starting locations shown in Figure 4.5 in the following order: (-2,4), (3.0,3.0), (2,-4), (-3.0,-3.0). Results of the experiments are shown in Table 4.5, along with typical results obtained without the starting position error for comparison purposes. Each entry represents the number of moves per episode, during evaluation, averaged over 25 trials.

TABLE 4.5
STARTING POSITION VARIABILITY RESULTS

	With Variability	*No Variability*
Average Number of Moves per Evaluation Episode	17.12	13.4

As can be seen from the results in Table 4.5, the introduction of error into the positioning of the robot at the beginning of each episode seems to have a significant negative impact on the performance of the system after 100 training episodes. As one might expect, this result seems to indicate that the physical robot starts out with a noticeable handicap in its ability to quickly learn the necessary action preferences necessary to locate the hidden platform as compared to the error free simulation.

4.2.1.3 Constant Reward Function Evaluation

The reward function, used by the TD system to generate the reward signal, is largely responsible for the ultimate performance of the system. In [1], the reward function described in section 3.6.4 was used and seemed to perform quite well. That reward function, though a superposition of multiple simple rewards, essentially yields an end of episode reward, assuming the robot locates the hidden platform, inversely proportional to the number of moves required to find the hidden platform.

For this work, a reward function that simply yields a constant scalar value in the case of the hidden platform being found was implemented and tested. This constant reward function can be formalized as:

$$\text{reward} = \begin{cases} -5.0 & \text{if } c = 0 \\ -5.0 & \text{if obstacle detected} \\ 5.0 & \text{if goal detected} \\ -5.0 & \text{if } n \geq m \\ 0.0 & \text{otherwise} \end{cases}$$

The use of the constant reward function is based on the theory that the eligibility traces built into the TD system should be capable of determining which state-action pairs are responsible, and to what degree, for the outcome of an episode, and that the reward function need not explicitly favor shorter paths. Specifically, the eligibility traces inherent to the system should effectively handle the responsibility of distributing rewards received with the goal of maximizing the reward signal, and, thus, the system must consequently end up learning short paths to the platform location.

In order to evaluate the impact of this constant reward function on the performance of the TD system, experiments were conducted in the simulated environment. Specifically, 25 trials, each consisting of 100 training and 100 evaluation episodes, were conducted using the constant reward function specified above. During these experiments, for both the training and evaluation phases, the robot was positioned at one of the four starting locations shown in Figure 4.5 in the following order: (-2,4), (3.0,3.0), (2,-4), (-3.0,-3.0). Results of the experiments are shown in Table 4.6, along with results obtained using the original reward function for comparison purposes. Each entry represents the number of moves per episode, during evaluation, averaged over 25 trials.

TABLE 4.6
CONSTANT REWARD FUNCTION RESULTS

	Constant Reward	*Original Reward*
Average Number of Moves per Evaluation Episode	11.9	13.4

As can be seen from the results in Table 4.6, although the performance is slightly better given the constant reward as compared to that obtained using the original reward, the difference in performance does not appear to be all that significant. However, what is significant is the fact that the performance of the system seems to be on par, if perhaps a bit improved, given a reward function that does not favor shorter paths. This result seems to provide evidence for the theory, described above, that the reward function need not explicitly favor shorter paths in order for the system to learn to favor them. It should be noted, however, that due to the relatively insignificant difference in performance the constant reward function was not adopted for future experiments.

4.2.1.4 Simulated Place Cell Evaluation

In much of the previous work done using a simulated or physical version of the Morris water maze task to evaluate computation models for the learning of spatial memory [5-7], simulated firings of hippocampal place cells were used to represent the state space. These hippocampal place cells are generally simulated by distributing a pre-determined number of such cells in a grid fashion over the entire environment, and

having the 'firing' of each of the place cells be modeled as some function of the distance of the robot to the location of the place cell. Thus, the firing of each place cell is based on the robot's spatial positioning in the environment.

For this work, the idea of hippocampal place cells that fire when the robot is in certain spatial locations was incorporated into the existing design to evaluate the performance of the system given not only the state information gained from the perception to SOM node mapping, but from a combination of perceptual localization and direct spatial localization. This combination was realized in the form of the perception based SOM nodes and a grid of place cells whose firings are directly tied to the robot's spatial location in the environment.

In order to evaluate the impact of incorporating this place cell grid into the TD system, experiments were conducted in the simulated environment. Eight trials, each consisting of 1,000 training and 1,000 evaluation episodes, were conducted in which the current state of the robot was a function not only of the current SOM node, but also of the firing of a simulated grid of hippocampal place cells. Specifically, 20 hippocampal place cells were evenly distributed over the simulated environment as shown in Figure 4.6. As the TD system can only handle a discrete number of possible states, the firing of the hippocampal place cells was taken to be '1' if the place cell was the closest place cell to the robot, and '0' otherwise. Thus, the total number of possible states is equal to the number of place cells times the number of SOM nodes. For the experiments conducted here, a 4 by 4 SOM, with 16 total nodes, was used in conjunction with the grid of 20 place cells, yielding a total of 320 states; roughly equal to the number of possible states given only an 8 by 8 SOM. Furthermore, for these experiments, two sets of 1,000

uniform random starting positions in the simulated environment were used; one set was used for the training episodes, and one set was used for the evaluation episodes. The large number of uniform random starting positions was used, as opposed to the four starting positions shown in Figure 4.5, to evaluate whether the spatial information gained from the grid of hippocampal place cells yielded better localization, and thus better performance, over the entire environment space then a SOM alone. Results of the experiments are shown in Table 4.7, along with results obtained using the original system with an 8 by 8 SOM. Each entry represents the number of moves per episode, during evaluation, averaged over 8 trials.

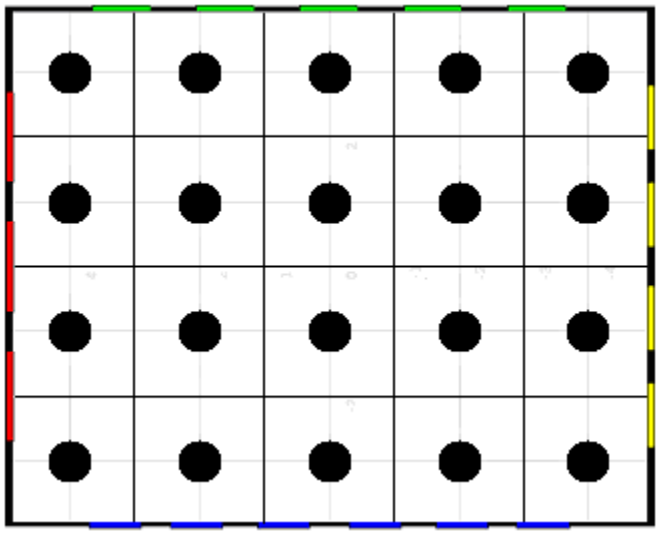

Figure 4.6: Diagram of hippocampal place cell layout in the simulated environment, used for experiments conducted in section 4.2.1.4.

TABLE 4.7
SIMULATED PLACE CELL RESULTS

	With Place Cells	*Without Place Cells*
Average Number of Moves per Evaluation Episode	16.6	19.1

As can be seen from the results, the performance of the system does seem to be better given the SOM and place cell combination, as compared to the SOM alone, perhaps providing evidence for the theory that inclusion of self motion information, i.e. odometry, could yield improved performance. However, both setups seem to perform well. It would certainly be interesting, given more time, to further investigate the performance of the system given different combinations of place cells and SOMs.

4.2.1.5 TD(λ) Multi Layer Perceptron Evaluation

In all the experiments presented in this work, the Self Organizing Map (SOM) used to discretize the perceptual space is an integral part of the setup. However, the SOM requires that a large number of perceptual vectors be collected from the environment before experiments are conducted; an advantage not given to rats in the typical Morris water maze environment. Therefore, the use of Multi Layer Perceptron (MLP) implementation of the TD(λ) algorithm, described in section 2.2, was explored for the purpose of eliminating the need for the SOM. Specifically, the idea of directly using the 18 bin perceptual vectors obtained from the robot's cameras for the current state information was tested.

Two different MLP structures were evaluated for this purpose: the first structure consisted of 54 inputs (three cameras each with an 18 dimensional feature vector) each directly tied to five output nodes (one for each possible move); the second structure consisted of 270 inputs (three cameras each with an 18 dimensional feature vector, multiplied by the number of possible moves) connected to a hidden layer of five nodes, which were then connected to a single output node. In the case of the first structure, the current perceptual state of the robot was observed, and the output node with the highest value determined the move that was executed. In the case of the second structure, each possible state-action pair was applied to the network, and the pair that resulted in the highest value from the single output node was taken as the winner. Finally, a small percentage chance for exploration was allowed in both cases, and naïve eligibility traces were used in the implementation.

In order to evaluate the performance of these two MLP structures trained with temporal difference learning, experiments were conducted in the simulated environment. Specifically, 25 trials, each consisting of 100 training episodes, were conducted using the basic reward function described in section 3.4. During these experiments, the robot was positioned at one of the four starting locations shown in Figure 4.5 in the following order: (-2,4), (3.0,3.0), (2,-4), (-3.0,-3.0). Simply stated, the second MLP structure never yielded results that seemed to indicate it had learned. Similarly, the first MLP structure, although it did occasionally seem to yield results indicating it had at least adjusted its performance to the environment, never ultimately gave any results indicative of actual convergence. Based on this, the use of a MLP implementation of the TD(λ) algorithm was abandoned.

However, given more time, it would be interesting to further investigate the use of a MLP approach and its ability to possibly eliminate the need for a SOM.

4.2.2 Non-stationary Environments

One of the main motivations for using the temporal difference (TD) learning approach for the water maze task is the ability for online learning and the supposed adaptability of the system in dynamic, non-stationary, environments. In order to test the adaptability of the TD approach in such environments, experiments were conducted in simulation. Specifically, the adaptability of the TD approach was tested by first letting the system train for a number of episodes given a fixed platform location in the dry water maze environment. After this training period, the platform location was moved, and the system was again allowed to train for a number of episodes given the new goal location. By plotting the average number of moves per episode during the entire training sequence, the ability of the system to adapt to new goal locations could be observed.

4.2.2.1 Original System

Initially, a set of experiments were conducted to evaluate the adaptability of the original TD learning approach, as implemented in the WMtk, in non-stationary environments. The results of these experiments are presented here.

These experiments consisted of changing the goal location during training trials. Firstly, the goal location was moved from (-1.0, -1.5) to (0.0, 2.5) after 50, 100, and 400 training episodes, at which point the robot was then given 400 more training episodes with the new goal location. At the beginning of each training episode, the robot was

positioned at one of four starting locations. A diagram of the goal locations and starting positions can be seen in Fig. 4.7. The starting locations were visited in the following order: (-2,4), (3.0,3.0), (2,-4), (-3.0,-3.0). The robot was allowed a maximum of 50 moves per episode to find the hidden platform.

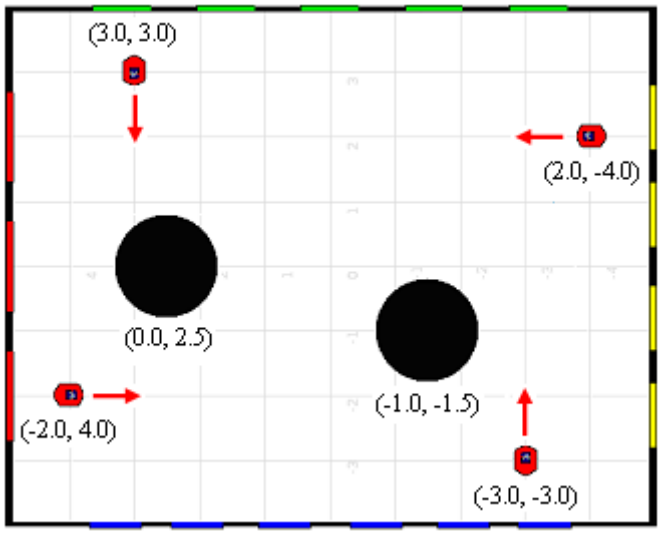

Figure 4.7: Diagram of goal locations and starting positions for experiments conducted in section 4.2.2.1.

Figure 4.8 shows the results of simply setting the goal location to (0.0, 2.5) and letting the robot train for 400 episodes. This is shown for comparison purposes versus the experiments in which the goal is moved to (0.0, 2.5) only after some number of training episodes with a prior goal location. Figure 4.9, Figure 4.10, and Figure 4.11 show the results of experiments in which the goal was moved to (0.0, 2.5) from (-1.0, -1.5) after 50, 100, and 400 training episodes respectively. Each graph is a plot of the four episode

moving average of the number of moves per episode, as averaged over multiple training trials.

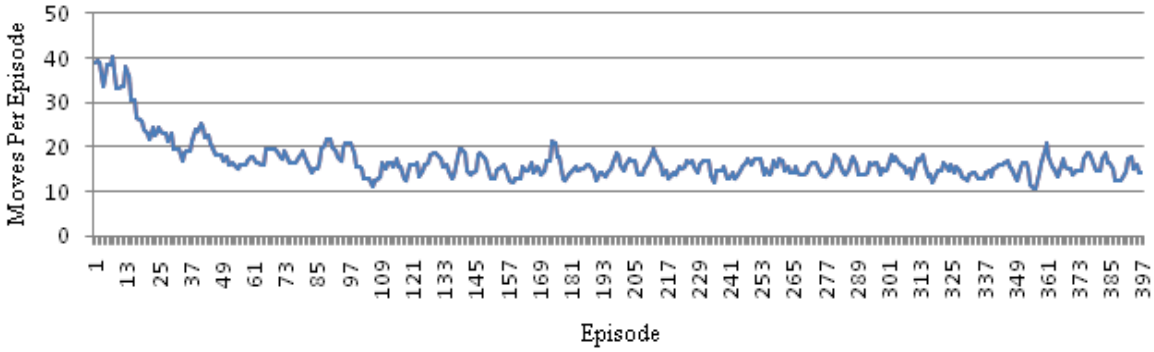

Figure 4.8: Graph of the four episode moving average of the number of moves per episode, as averaged over six trials. The goal location was set to (0.0, 2.5) for the entirety of each trial. The maximum allowed number of moves per episode is 50.

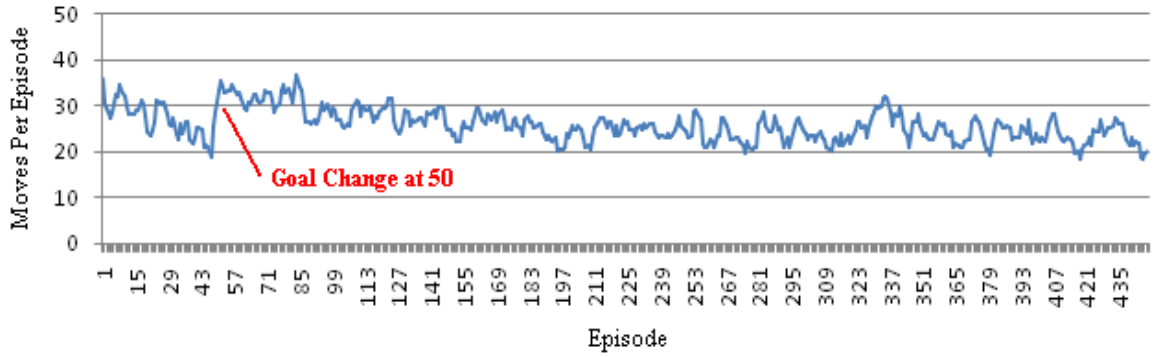

Figure 4.9: Graph of the four episode moving average of the number of moves per episode, as averaged over six trails. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 50 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 400 more training episodes. The maximum allowed number of moves per episode is 50.

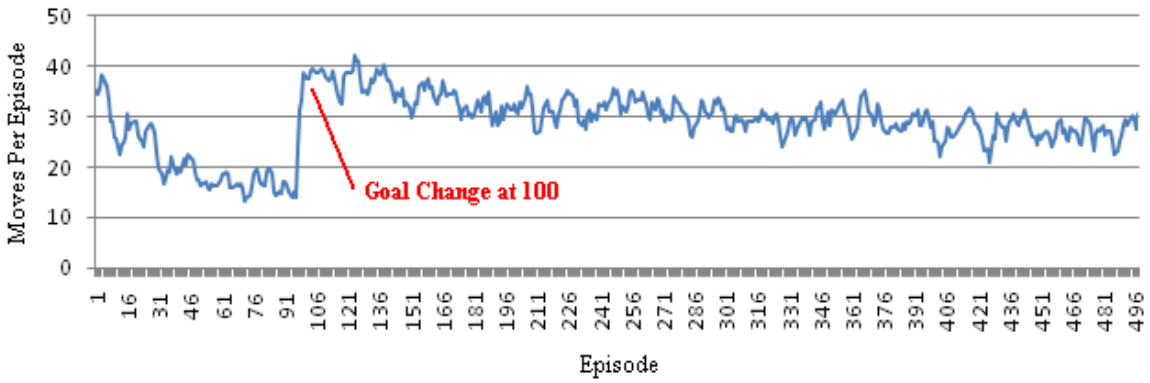

Figure 4.10: Graph of the four episode moving average of the number of moves per episode, as averaged over six trials. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 100 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 400 more training episodes. The maximum allowed number of moves per episode is 50.

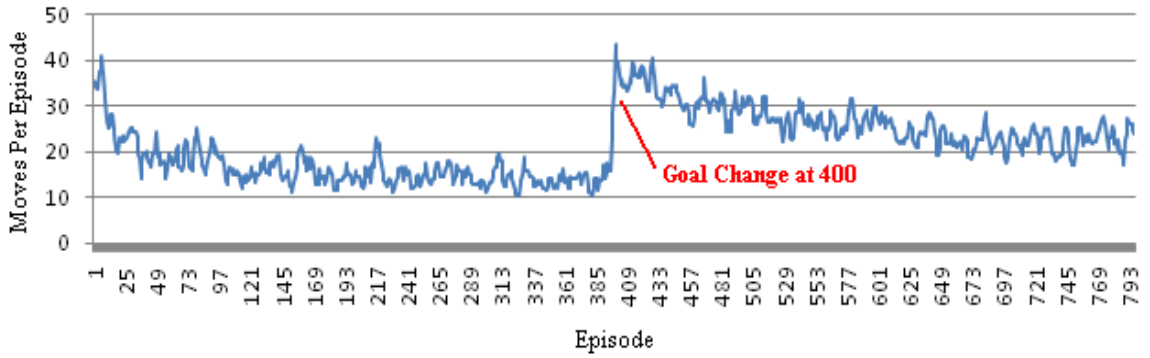

Figure 4.11: Graph of the four episode moving average of the number of moves per episode, as averaged over six trials. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 400 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 400 more training episodes. The maximum allowed number of moves per episode is 50.

As can be seen from the results, after training for even as few as 50 episodes before moving the goal location, the robot fails to learn paths to the new goal location that are as efficient as those learned if the system is only trained using the new goal location. In addition, the length of initial training does not seem to have a significant impact on the ability of the system to adapt to the second goal location. In the 50, 100, and 400 episode cases, after 400 training episodes with the new goal location the robot seems to find a set of paths that average in the 20-30 move range, as opposed to the set of paths that average in the 10-20 move range when the system is only trained on the second goal location for 400 episodes.

Next, a second batch of experiments, which consisted of letting the robot train for a larger number of episodes after the repositioning of the goal location, were conducted to determine if the performance of the TD system would eventual reach the same level as when it was only trained on the second goal location. Specifically, the robot was allowed to train for 4,000 episodes after the repositioning of the goal location from (-1.0, -1.5) to (0.0, 2.5). The results are shown in Figure 4.12. Here again, the plot is the four episode moving average of the number of moves per episode, as averaged over multiple training trials.

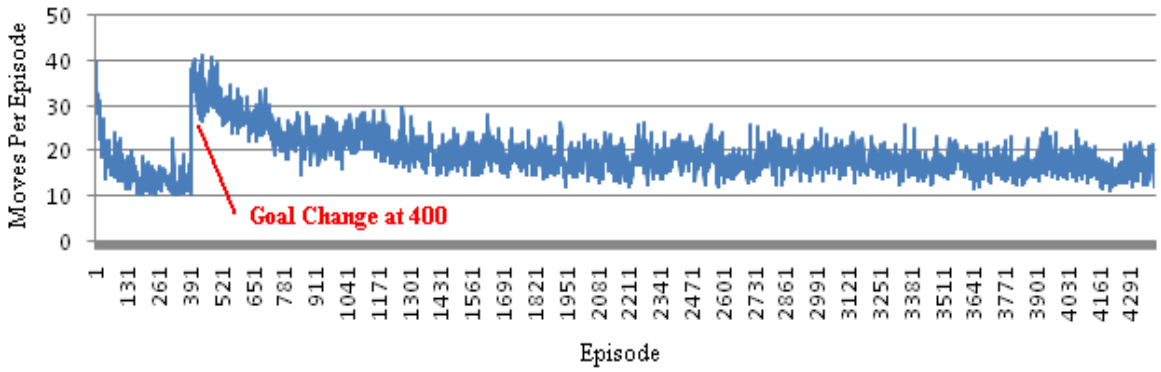

Figure 4.12: Graph of the four episode moving average of the number of moves per episode, as averaged over five trials. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 400 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 4,000 more training episodes. The maximum allowed number of moves per episode is 50.

As can be seen from Figure 4.12, the system does appear to keep improving over the entire sequence of training episodes, albeit slowly. After 4,000 training episodes the system's performance does reach approximately the same level as when it is trained only on the second goal location for 400 episodes. This seems to imply that the learning which takes place during the initial 400 episodes is not "forgotten" until after a large number of training episodes with the new goal location. Furthermore, this initial learning seems to hinder the convergence of the system toward the best paths for the new goal location.

Finally, a set of experiments was conducted which consisted of moving the goal location twice. In the first experiment, the robot was allowed to train for 400 episodes with the platform at (0.0, -2.0). The goal location was then moved to (0.0, 2.0) and the robot was allowed to train for 400 episodes given the new location. Lastly, the goal location was moved to (0.0, 0.0), and the robot was allowed to complete 400 more training episodes. The results of this experiment are shown in Figure 4.13. Once again,

the plot is the four episode moving average of the number of moves per episode, as averaged over multiple training trials.

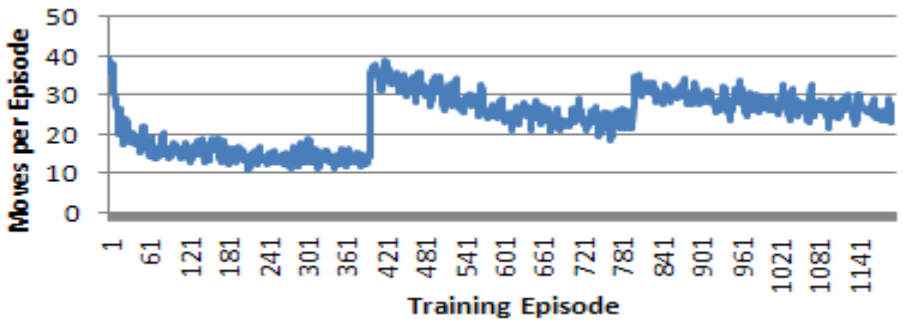

Figure 4.13: Graph of the four episode moving average of the number of moves per episode, as averaged over ten trials. At the beginning of each trial, the goal location was set to (0.0, -2.0). After 400 training episodes the goal location was moved to (0.0, 2.0) and the robot was allowed to complete 400 more training episodes. Finally, the goal was moved to (0.0, 0.0) and the robot was allowed to complete another 400 training episodes. The maximum allowed number of moves per episode is 50.

Additionally, a second experiment was conducted during which the goal location was moved twice. For this experiment, the robot was first allowed to train for 400 episodes with the platform at (-1.0, -1.5). The goal location was then moved to (0.0, 2.5) and the robot was allowed to train for 400 episodes with the new location. Finally, the goal location was moved back to where it originally started at (-1.0, -1.5), and the robot was allowed to complete 400 more training episodes. The results are shown in Figure 4.14. Once again, the plot is the four episode moving average of the number of moves per episode, as averaged over multiple training trials.

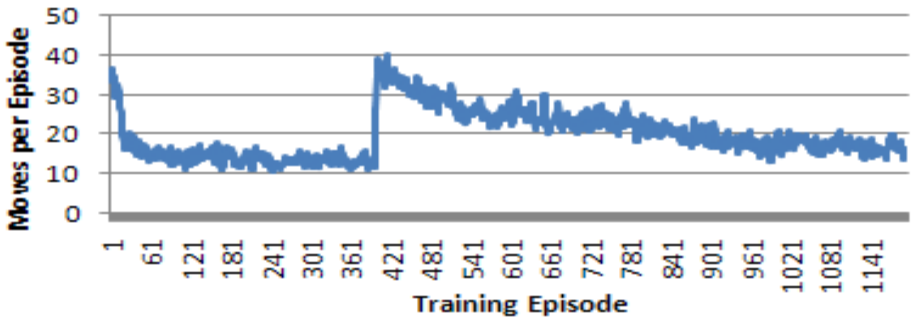

Figure 4.14: Graph of the four episode moving average of the number of moves per episode, as averaged over ten trials. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 400 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 400 more training episodes. Finally, the goal was moved back to (-1.0, -1.5) and the robot was allowed to complete another 400 training episodes. The maximum allowed number of moves per episode is 50.

As can be seen from the graph in Figure 4.14, there is no spike in the number of moves per episode when the platform is moved back to where it original started after episode 800. This seems to imply that a significant amount of what was learned during the initial 400 training episodes has been retained, and not "forgotten", even after 400 training episodes with the new goal location. However, what is learned during the 400 training episodes after the initial movement of the goal does seem to slow the rate of convergence of the system after the goal is moved back to where it originally started. Lastly, the graph from Figure 4.13 seems to indicate that the performance of the system seems to further degrade when the platform is moved to a third unique location. These results clearly seem to demonstrate an inability on the part of the TD system to adequately adapt in non-stationary environments.

4.2.2.2 Adaptive Learning and Exploration Rates

With the goal of obtaining better adaptability in non-stationary environments, the use of adaptive learning and exploration rates was explored. Specifically, the process of adjusting the learning and exploration rates of the TD system, based on measures of past performance in the form of a "short term" and "long term" reward as described in section 3.7, was explored. The following equations illustrate how the learning rate, α, and exploration rate, γ, were adapted based on the short term and long term rewards r_s and r_l:

$$\alpha(t) = \begin{cases} k_\alpha \left(\left| r_s(t) - r_s(t-d) \right| \right)^{q_\alpha} & \text{if } r_s(t) \geq r_l(t) - \varepsilon \\ \alpha_{\max} & \text{otherwise} \end{cases}$$

$$\gamma(t) = \begin{cases} k_\gamma \left(\left| r_s(t) - r_s(t-d) \right| \right)^{q_\gamma} & \text{if } r_s(t) \geq r_l(t) - \varepsilon \\ \gamma_{\max} & \text{otherwise} \end{cases}$$

where k_α, q_α, k_γ, q_γ, and d are constants. Effectively, if the short term reward is less then the long term reward minus ε, as would be the case following a change in hidden platform location, then the learning and exploration rates are set to their maximum allowed values to encourage the TD system to explore and learn. For the experiment shown in Figure 4.15, $\alpha_{\max}$ was set to 0.03 and $\gamma_{\max}$ was set to 0.25; values significantly greater then those used when the parameters were fixed.

However, if the short term reward is greater then or equal to the long term reward minus ε, as would be the case during the process of learning a new goal location (after some initial exploration) or retaining information about a current goal location, then the learning and exploration rates are adjusted to be exponentially proportional to an estimate

of the absolute value of the derivative of the short term reward; where the parameters k and q control the relationship. This relationship has the effect of forcing the learning and exploration rates to zero as the performance of the system plateaus. For the experiment shown in Figure 4.15, k_α was set to 0.05, k_γ was set to 0.25, q_α was set to 1.6, q_γ was set to 1.6, and d was set to 15.

In order to evaluate the performance of the TD system with the addition of adaptable learning and exploration rates, experiments were conducted during which the robot was first allowed to train for 400 episodes with the platform at (-1.0, -1.5). The goal location was then moved to (0.0, 2.5) and the robot was allowed to train for 400 episodes with the new location. The results of a typical set of these experiments are shown in Figure 4.15. Once again, the plot is the four episode moving average of the number of moves per episode, as averaged over multiple training trials.

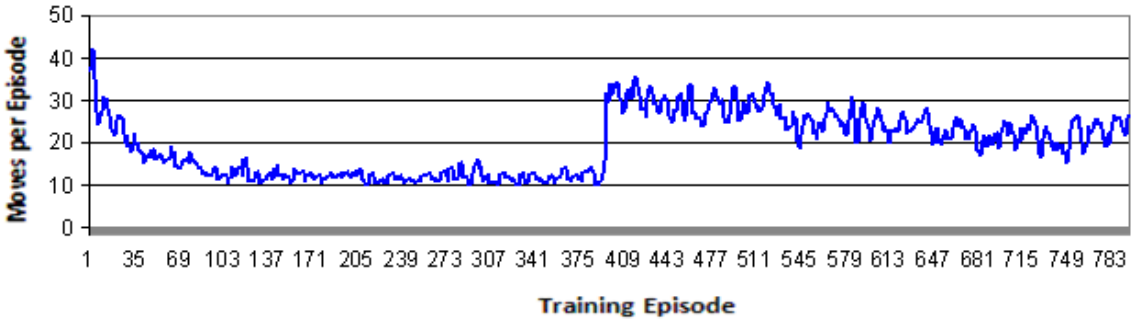

Figure 4.15: Graph of the four episode moving average of the number of moves per episode, as averaged over six trials, with adaptive exploration and learning rates. At the beginning of each trial, the goal location was set to (-1.0, -1.5). After 400 training episodes the goal location was moved to (0.0, 2.5) and the robot was allowed to complete 400 more training episodes. The maximum allowed number of moves per episode is 50.

As the results in Figure 4.15 illustrate, the use of adaptive exploration and learning rates proves to be an unsatisfactory method for achieving better adaptability in non-stationary environments. Adjustments to these parameters based on the short and long term performance measures tend to have little impact on the performance of the system, giving results similar to those shown in Figure 4.11 obtained by the original system. Ultimately, the use of adaptive learning and exploration rates does not effectively deal with the problem of prior learning hindering the performance of the TD system to adapt to new hidden platform locations.

4.2.2.3 Forgetting

In order to obtain better adaptability in non-stationary environments by reducing the hindrance of prior learning, the effect of adding "forgetting" to the TD system was explored. As detailed in section 3.7, the "forgetting" modification is based on keeping track of past performance, namely rewards received, in the form of a "short term" reward and a "long term" reward. These two rewards are then used to control when, and to what extent, "forgetting" takes place.

Figure 4.16 shows the results of the same experiment as conducted in Figure 4.13, but with the addition of "forgetting" incorporated into the TD system. Recall that in this experiment, the robot was allowed to train for 400 episodes with the hidden platform at (0.0, -2.0). The goal location was then moved to (0.0, 2.0) and the robot was allowed to train for 400 episodes given the new location. Lastly, the platform location was moved to (0.0, 0.0), and the robot was allowed to complete 400 more training episodes.

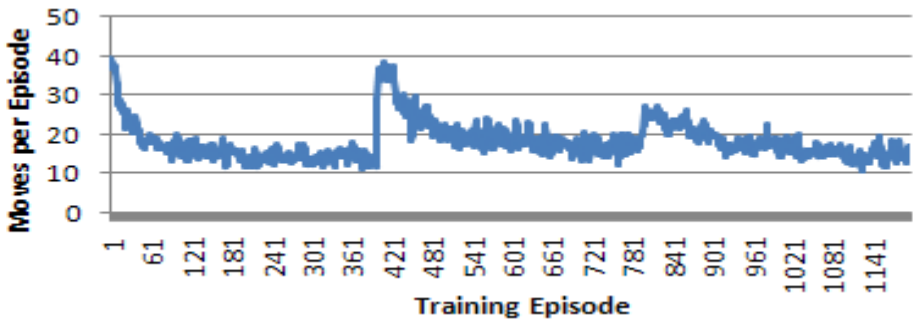

Figure 4.16: Graph of the four episode moving average of the number of moves per episode, as averaged over ten trials, on the same experiment conducted in Figure XX, but with the addition of "forgetting" incorporated into the TD system.

As can be seen from the comparison of the results shown in Figures 4.16 and 4.13, with "forgetting" incorporated into the TD system, it is effectively able to adapt to the new platform locations as though it had not undergone any previous learning, thus making it much more adaptable in non-stationary environments.

Additionally, Figure 4.17 shows the results of the same experiment as conducted in Figure 4.14, but, once again, with the addition of "forgetting" incorporated into the TD system. Recall that for this experiment, the robot was first allowed to train for 400 episodes with the platform at (-1.0, -1.5). The goal location was then moved to (0.0, 2.5) and the robot was allowed to train for 400 episodes with the new location. Finally, the goal location was moved back to where it originally started at (-1.0, -1.5), and the robot was allowed to complete 400 more training episodes.

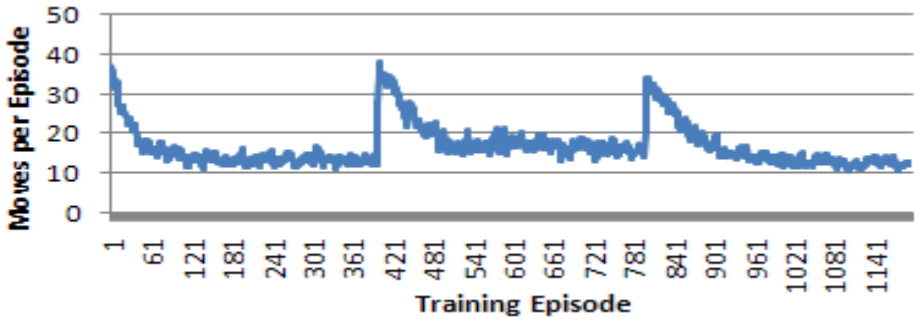

Figure 4.17: Graph of the four episode moving average of the number of moves per episode, as averaged over ten trials, on the same experiment conducted in Figure XX, but with the addition of "forgetting" incorporated into the TD system.

Here again, from the comparison of the results shown in Figures 4.17 and 4.14, it appears that with "forgetting" incorporated into the TD system, it is effectively able to adapt to the new platform location, and re-adapt to the original platform location, as though it had not undergone any previous learning, thus making it much more adaptable in non-stationary environments.

The essential benefit inherent to the method of incorporating a controlled process of "forgetting" into the TD system, as oppose to simply using the "short term" and "long term" rewards to decide when to reset the system, is that at the moment the forgetting process starts, all of the information contained in the TD system representation is not lost, as would be the case if the system were to simply be reset. Thus, the system with "forgetting" should, theoretically, be less susceptible to noise in the reward signal then a method based solely on resetting the entire system. However, no specific experiments were conducted to evaluate the benefit of the "forgetting" method over a complete system reset. It would be interesting to investigate the impact of not simply throwing away everything that has been learned, as the "forgetting" method allows, perhaps using an

experiment where the change in goal location is minimal, or done in such a way that some of the information already learned by the system could be applied to the new context.

Chapter 5

Discussion

The objective of this work was to evaluate the performance of a biologically inspired Temporal Difference (TD) approach to the learning of spatial memory for a robot in a dry version of the Morris water maze task. In order to achieve this objective, a number of experiments were conducted in both a simulated environment, and in a physical environment, as detailed in Chapter 4. The results of these experiments yielded a significant amount of information about both the capabilities and limitations of the proposed TD approach, and its ability to learn the state-action preferences necessary to successfully navigate to the hidden platform in the Morris water maze task.

From the results obtained by Busch et al. in [1], it is clear that the TD learning approach, as implemented in the Working Memory Toolkit (WMtk), is capable of learning the necessary action preferences to locate the hidden platform in a simulated version of the Morris water maze task. However, the TD learning approach was never extended from the simulated environment to a real world environment with a physical robot. Furthermore, a detailed set of experiments was never conducted to observe the effect of various parameters on the performance of the system, or the performance of the system given a more complex, non-stationary environment. This work achieved those objectives.

In extending the TD approach to a physical robot, a number of issues not present in the simulated environment needed to be addressed, as discussed in section 3.2, such as segmentation of colored panels from images captured from the cameras mounted on the robot, and repositioning of the robot at the start of each episode. Although these issues

introduced new sources of error not present in the simulated environment, experiments, as detailed in section 4.1, showed the TD approach was still capable of learning the necessary action preferences to successfully navigate to the hidden platform. Additionally, the results of these experiments provide evidence that the system is quite robust in its ability to deal with occasional erroneous sensor readings which result in incorrect state feature vectors. Ultimately, the results obtained by extending the TD approach to a physical robot were quite similar to those obtained in the simulated environment, and demonstrate that the TD approach is capable of dealing with the additional uncertainties that a physical environment introduces.

In addition to extending the TD approach to a physical robot, a large number of experiments were conducted in the simulated environment to observe the impact of various parameters on the performance of the system, as detailed in section 4.2.1. The list of parameters tested included the size of the Self-Organizing Map (SOM) used to discretize the perceptual space; the learning rate, exploration percentage, and temporal credit assignment value of the TD system; and a constant reward function, as opposed to the path length dependent reward function described in section 3.6.4. The results of these experiments seemed to indicate that, in general, the performance of the TD approach was rather invariant to changes in these parameters; once again illustrating the robustness of the TD approach.

Finally, a large amount of work was done to both evaluate, and ultimately improve, the performance of the TD approach in more complex, non-stationary environments. Specifically, the simulated Morris water maze environment was adapted such that the location of the hidden platform could be changed during learning trials.

This change allowed for the performance of the TD approach to be observed for environments where the hidden platform location was not stationary. Results from an initial set of experiments, as detailed in the section 4.2.2.1, clearly showed the inability of the TD approach, as implemented in the WMtk, to adapt in non-stationary environments, thus significantly hindering its performance on subsequent goal locations, as compared to the performance obtained on those goal locations in the absence of prior learning.

In order to address the inability of the TD approach to adapt in non-stationary environments, experiments were conducted to evaluate the affect of adaptive learning and exploration rates based on short and long term measures of performance, as described in section 4.2.2.2. Ultimately, the results from these experiments indicated that the performance of the system in non-stationary environments was not significantly impacted by making these parameters of the TD system adaptive; as the prior learning of the system was still a major hindrance to the learning of efficient paths to new goal locations.

Experiments were then conducted to evaluate the impact of integrating active "forgetting," once again based on short and long term measures of performance, into the TD system, as described in section 3.7, on its ability to adapt in non-stationary environments. The results of these experiments, detailed in section 4.2.2.3, clearly show the improved adaptability of the TD approach in such environments. Whereas the ability of the TD system, as originally implemented in the WMtk, to learn efficient paths to new hidden platform locations was significantly hindered in the presence of prior learning, with the integration of active "forgetting" the TD system is able to achieve performance on new hidden platform locations effectively equivalent to that obtained by the original implementation in the absence of prior learning. Additionally, the gradual impact of

active “forgetting” completely eliminates the possibility of accidentally destroying all prior knowledge due to noise in the short and long term performance measures, as oppose to an actual change in hidden platform location, as would be a distinct possibility if “hard resets” were used to address the problem of adaptability of the TD approach in non-stationary environments.

Although not investigated in this work, and thus a possible avenue for additional investigation, it would be interesting to observe the performance of the TD approach, both with and without active “forgetting,” in non-stationary environments that introduced a less severe change in hidden platform location. The changes in hidden platform location used for the experiments conducted in this work effectively required the complete elimination, or “forgetting,” of knowledge related to previous hidden platform locations in order to achieve good performance on subsequent locations. However, it seems likely that in non-stationary environment scenarios consisting of a small change to the hidden platform location, that the retention of some amount of prior knowledge may be beneficial in reducing the time required to learn efficient paths to new hidden platform locations.

In conclusion, the experiments conducted in this work showed both the ability of the TD approach to learn the necessary state-action preferences to efficiently navigate to the hidden platform in a dry version of the Morris water maze task. Additionally, these experiments helped identify both the robustness of the original TD approach to noise in the perceived state, as well as the limitations of that approach in non-stationary environments. Finally, these experiments clearly illustrated the benefit of integrating active “forgetting” into the TD approach to improve the adaptability of the TD system.

Chapter 6

Summary and Conclusion

This work presented the results of extending a biologically inspired temporal difference (TD) approach to the learning of spatial memory from a simulated environment to a physical robot, and also presented the results of testing the adaptability of that approach in non-stationary environments. Experiments showed that the robot is able, using the TD approach, to successfully learn the necessary associations between perceptual states and actions to successfully locate the hidden platform. In addition, as seen from the results of the experiments conducted in simulation, with the addition of "forgetting" the system is able to achieve good performance in non-stationary environments.

Bibliography

[1] M.A. Busch, M. Skubic, J.M. Keller, and K.E. Stone, "A Robot in a Water Maze: Learning a Spatial Memory Task," In *2007 IEEE International Conference on Robotics and Automation*, Rome, Italy, 10-14 April 2007, pp. 1727-1732.

[2] J.L. Krichmar, D. A. Nitz, J.A. Gally, and G. M. Edelman, "Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task," In *Proc National Academy of Science USA*, 2005, vol. 102, pp. 2111-2116.

[3] Morris, R. "Development of a water-maze procedure for studying spatial learning in the rat," *Journal of Neuroscience Methods,* 1984, vol. 11, pp. 47-60.

[4] R. Morris, P. Garrud, J. Rawlins, and J. O'Keefe, "Place navigation impaired in rats with hippocampal lesions," *Nature*, 1982, vol. 297, pp. 681-683.

[5] A.D. Redish and D.S. Touretsky, "The role of the hippocampus in solving the Morris water maze," Neural Computation, 1998, vol. 10, no. 1, pp. 73-111.

[6] M.A. Brown, and P.E. Sharp, "Simulation of spatial learning in the Morris water maze by a neural network model of the hippocampal formation and nucleus accumbens," *Hippocampus*, 1995, vol. 5, pp. 171-188.

[7] D.J. Foster, R.G.M. Morris, and Peter Dayan, "A Model of Hippocampally Dependent Navigation, Using the Temporal Difference Learning Rule," *Hippocampus*, 2000, vol. 10, pp. 1-16.

[8] Kohonen, T. 1990. The self-organizing map. Proc. IEEE 78, 1464-1480.

[9] Sutton, R.S. "Learning to predict by the methods of temporal differences," *Machine Learning*, 1988, vol. 3, pp. 9-44.

[10] R.S. Sutton, and A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, Massachusetts, 1998.

[11] Watkins, C.J.C.H., Learning from Delayed Rewards, PhD thesis, Cambridge University, Cambridge, England, 1989.

[12] Watkins, C.J.C.H., and Dayan, P., "Q-learning," *Machine Learning*, 1992, vol. 8, pp. 279-292.

[13] Peng, J. and Williams, R. J, "Incremental multi-step Q-learning," *Machine Learning*, 1996, vol. 22, issue 1-3, pp. 283-290.

[14] Sutton, R.S. “Implementation details of the TD(lambda) procedure for the case of vector predictions and backpropagation,” GTE Laboratories Technical Report TR87-509.1, as corrected August 1989. GTE Laboratories, 40 Sylvan Road, Waltham, MA 02254.

[15] J.L. Phillips and D.C. Noelle, “A Biologically Inspired Working Memory Framework for Robots,” In *Proc. of the 27th Annual Meeting of the Cognitive Science Society*, Stresa, Italy, July 2005.

[16] M. Skubic, D. Noelle, M. Wilkes, K. Kawamura, J. Keller, “A Biologically Inspired Adaptive Working Memory for Robots,” AAAI 2004 Fall Symposium, Workshop on The Intersection of Cognitive Science and Robotics: From Interfaces to Intelligence, Washington, D.C., Oct. 21-24, 2004.

[17] R.H. Luke, J.M. Keller, M. Skubic and S. Senger, “Acquiring and Maintaining Abstract Landmark Chunks for Cognitive Robot Navigation,” in *Proc. of the IEEE Intl. Conf. on Robots and Intelligent Systems (IROS)*, Edmonton, Alberta, Canada, Aug., 2005.

[18] K. Kawamura, W. Dodd, P. Ratanaswasd, and R.A. Gutierrez, "Development of a Robot with a Sense of Self", *Proc. of the 6th IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation (CIRA),* Espoo, Finland, June, 2005.

[19] Gerkey, B.P., Vaughan, R.T. and Howard, A. 2003. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In Proc. IEEE Intl. Conf. Advanced Robotics, Coimbra, Portugal.

[20] G. Bradski, et. al. Intel Open Source Computer Vision Library: http://www.intel.com/technology/computing/opencv/

Appendix A

Additional Single Corner Task Results from Physical Environment

A.1 Additional Single Corner Task - Sequence 1

This single corner task training sequence consisted of 126 episodes and was conducted with the following parameters: 20x20 SOM size, exploration rate of 0.05, learning rate of 0.01, temporal credit assignment value of 0.7, initial mean weight of 2.0, and reward function as described in section 3.6.4. Figure A.1 shows a plot of moves per episode during the training sequence. Figure A.2 shows example paths of the robot from the training sequence.

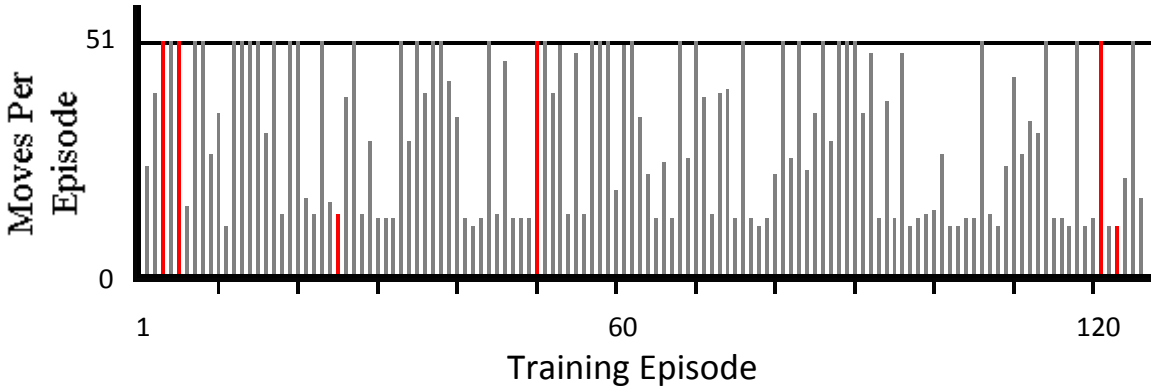

Figure A.1: Plot of moves per episode during training for the single corner water maze task described in this section. The maximum number of moves allowed per episode is 51. The paths followed by the robot for the episodes highlighted in red are shown in Figure A.2.

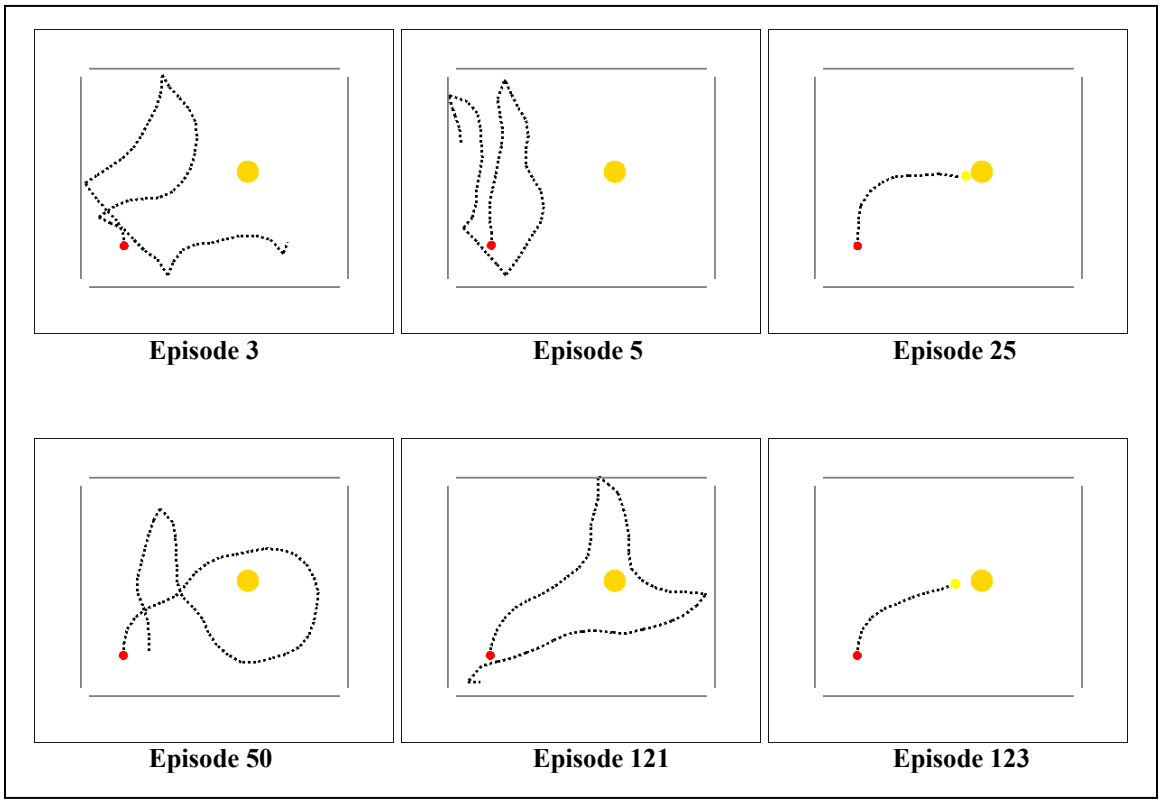

Figure A.2: Paths of robot during training episodes highlighted in red in Figure A.1. The inner box shows the approximate distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training episodes. This information is not used by the system.

A.2 Additional Single Corner Task - Sequence 2

This single corner task training sequence consisted of 135 episodes and was conducted with the following parameters: 20x20 SOM size, exploration rate of 0.05, learning rate of 0.01, temporal credit assignment value of 0.7, initial mean weight of 2.0, and reward function as described in section 3.6.4. Figure A.3 shows a plot of moves per episode during the training sequence. Figure A.4 shows example paths of the robot from the training sequence.

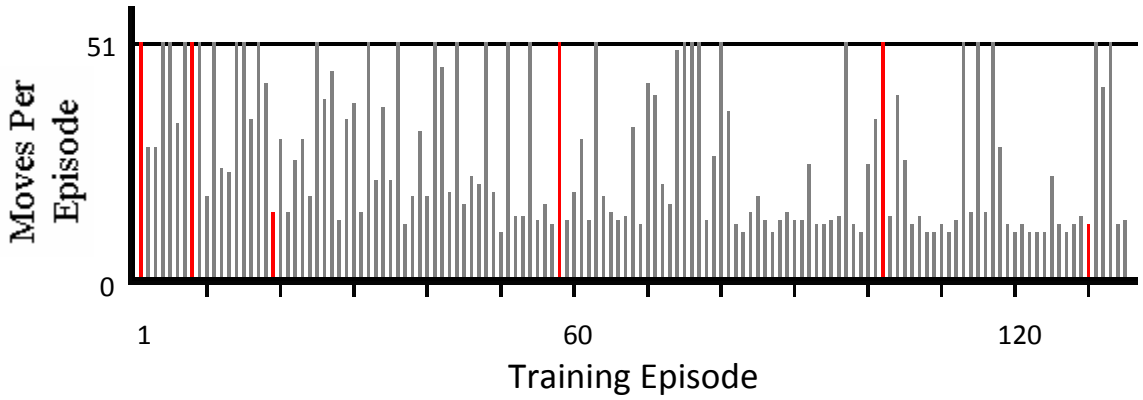

Figure A.3: Plot of moves per episode during training for the single corner water maze task described in this section. The maximum number of moves allowed per episode is 51. The paths followed by the robot for the episodes highlighted in red are shown in Figure A.4.

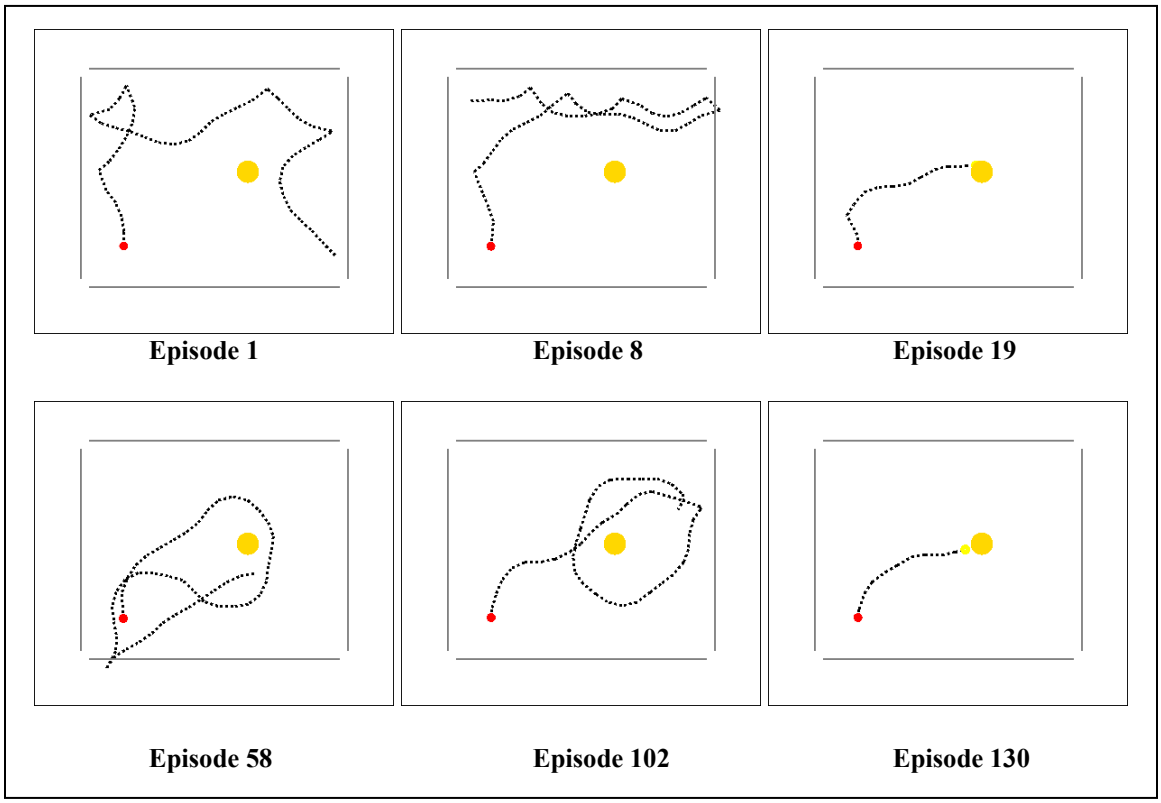

Figure A.4: Paths of robot during training episodes highlighted in red in Figure A.3. The inner box shows the approximate distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training episodes. This information is not used by the system.

A.3 Additional Single Corner Task - Sequence 3

This single corner task training sequence consisted of 169 episodes and was conducted with the following parameters: 8x8 SOM size, exploration rate of 0.05, learning rate of 0.01, temporal credit assignment value of 0.7, initial mean weight of 2.0, and reward function as described in section 3.6.4. Figure A.5 shows a plot of moves per episode during the training sequence. Figure A.6 shows example paths of the robot from the training sequence.

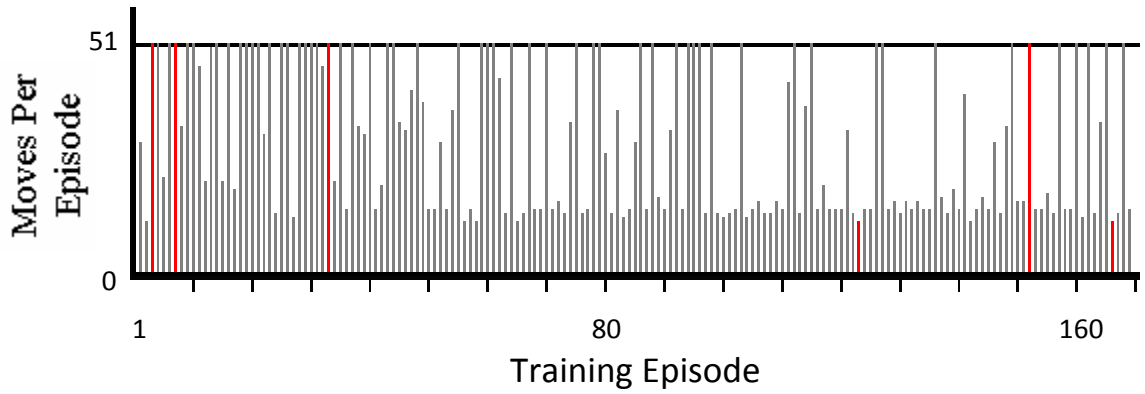

Figure A.5: Plot of moves per episode during training for the single corner water maze task described in this section. The maximum number of moves allowed per episode is 51. The paths followed by the robot for the episodes highlighted in red are shown in Figure A.6.

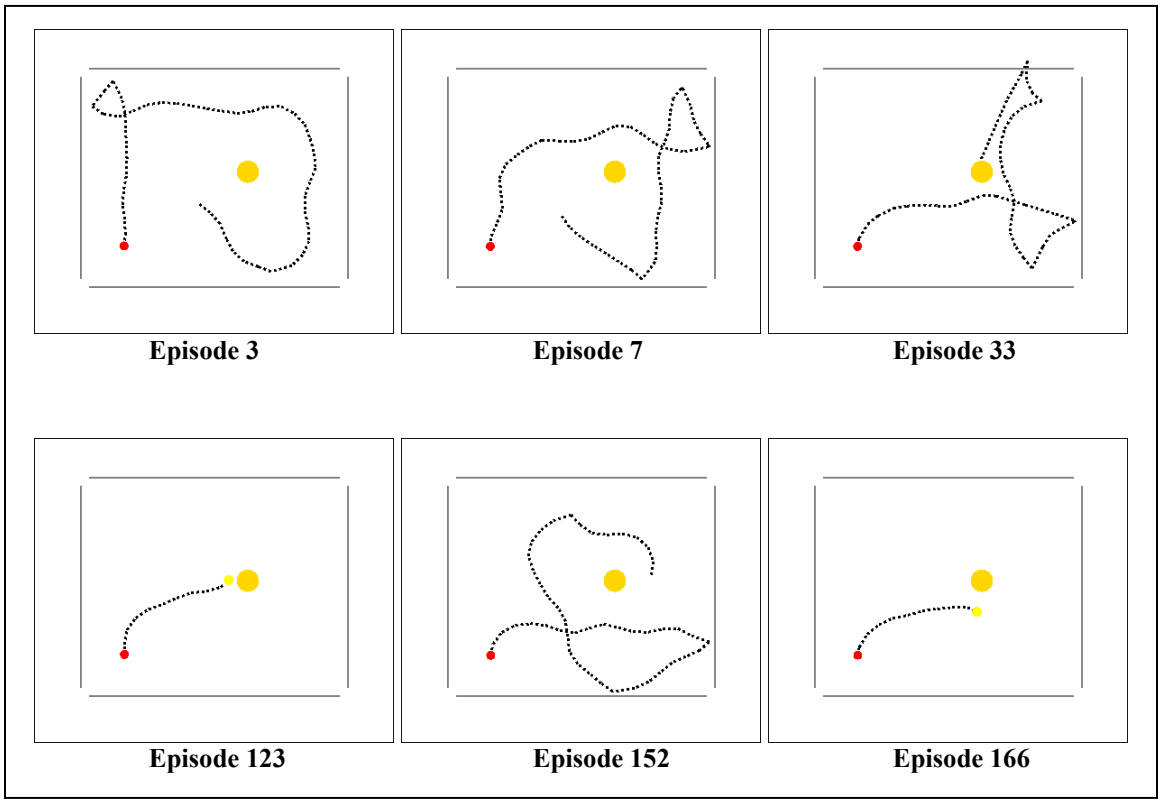

Figure A.6: Paths of robot during training episodes highlighted in red in Figure A.5. The inner box shows the approximate distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training episodes. This information is not used by the system.

Appendix B

Additional Four Corner Task Results from Physical Environment

B.1 Additional Four Corner Task - Sequence 1

This four corner task training sequence consisted of 100 episodes and was conducted with the following parameters: 8x8 SOM size, exploration rate of 0.1, learning rate of 0.01, temporal credit assignment value of 0.83, initial mean weight of 2.5, and reward function as described in section 3.6.4. Figure B.1 shows a plot of the four episode moving average of moves per episode during the training sequence. Figure B.2 shows example paths of the robot from the training sequence.

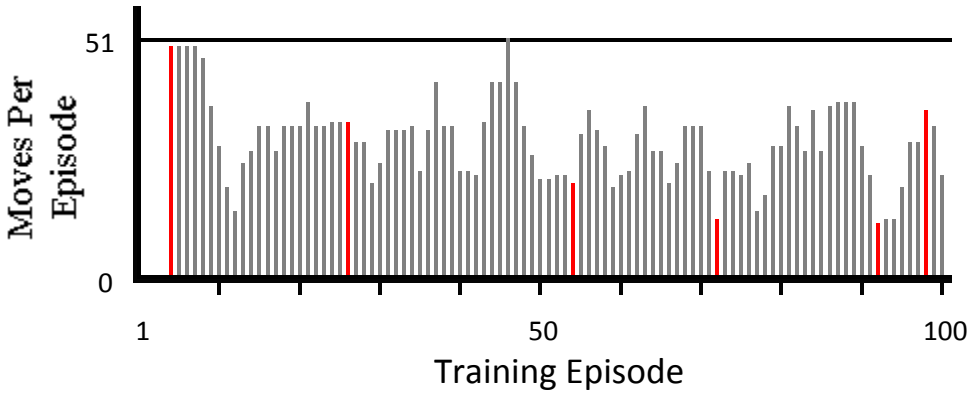

Figure B.1: Plot of the four episode moving average of moves per episode during training for the four corner water maze task described in this section. The moving average is of the current episode and three previous episodes, and is used to show the change in the number of moves from the four combined starting locations. The maximum number of moves allowed per episode is 51. The paths followed by the robot for the episodes highlighted in red are shown in Figure B.2.

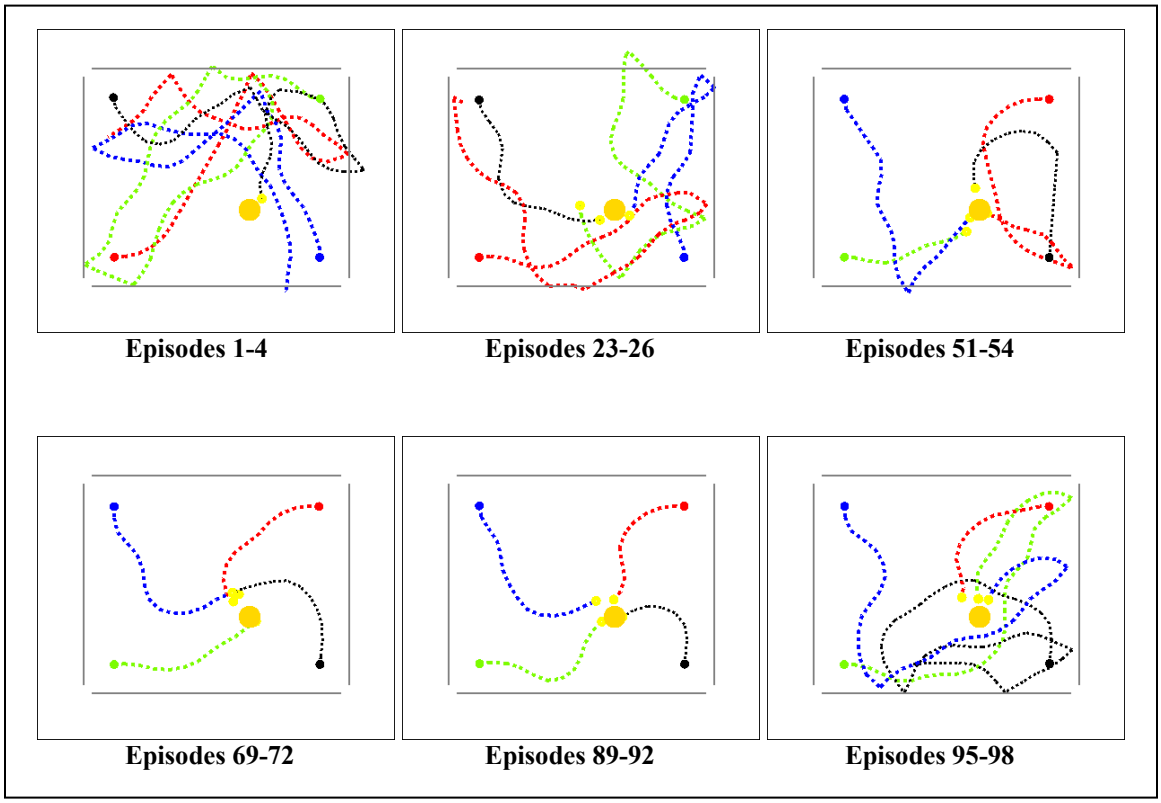

Figure B.2: Paths of robot during training episodes highlighted in red in Figure B.1. The inner box shows the approximate distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training episodes. This information is not used by the system.

B.2 Additional Four Corner Task - Sequence 2

This four corner task training sequence consisted of 100 episodes and was conducted with the following parameters: 8x8 SOM size, exploration rate of 0.1, learning rate of 0.01, temporal credit assignment value of 0.83, initial mean weight of 2.5, and reward function as described in section 3.6.4. Figure B.3 shows a plot of the four episode moving average of moves per episode during the training sequence. Figure B.4 shows example paths of the robot from the training sequence.

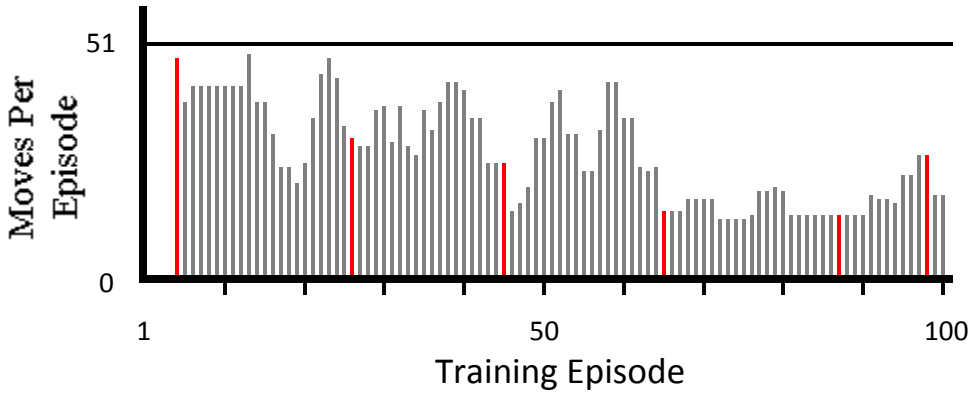

Figure B.3: Plot of the four episode moving average of moves per episode during training for the four corner water maze task described in this section. The moving average is of the current episode and three previous episodes, and is used to show the change in the number of moves from the four combined starting locations. The maximum number of moves allowed per episode is 51. The paths followed by the robot for the episodes highlighted in red are shown in Figure B.4.

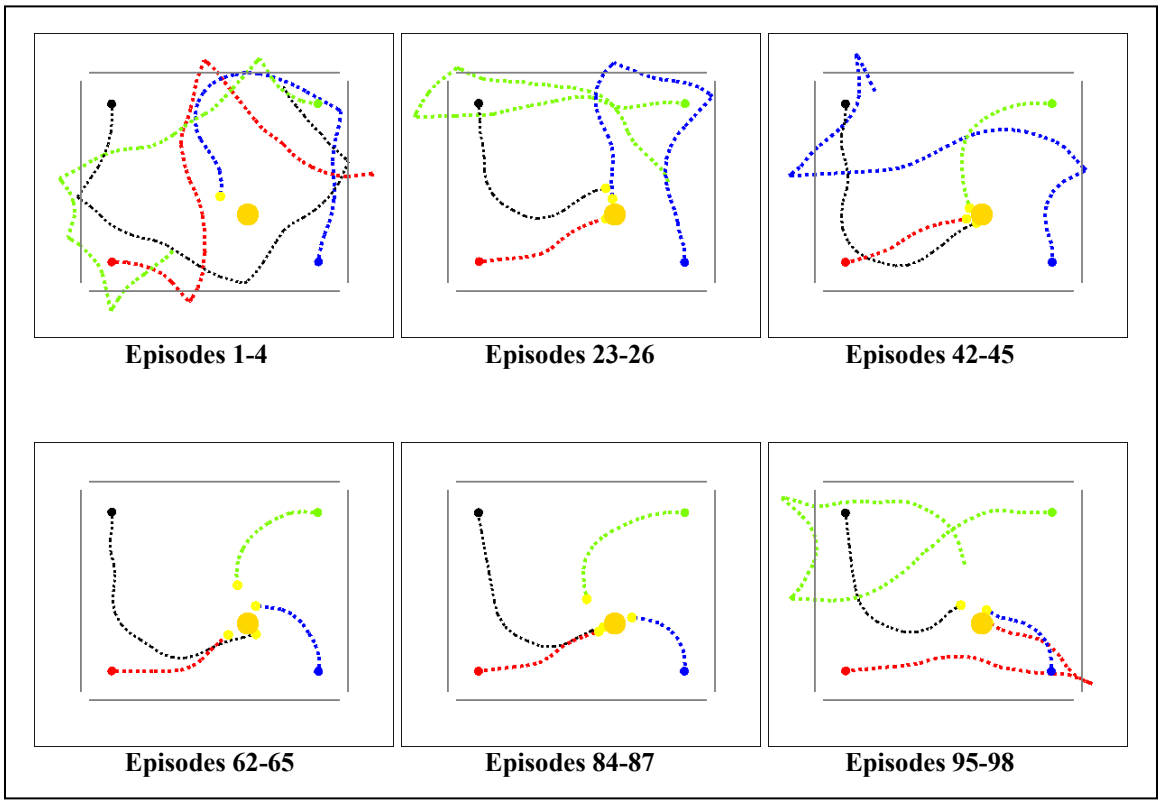

Figure B.4: Paths of robot during training episodes highlighted in red in Figure B.3. The inner box shows the approximate distance at which obstacle avoidance should be activated. The robot's path is logged using odometry during training episodes. This information is not used by the system.