Application of Fuzzy State Aggregation and Policy Hill Climbing to Multi-Agent Systems in Stochastic Environments

Dean C. Wardell

Application of Fuzzy State Aggregation and Policy Hill Climbing to Multi-Agent Systems in Stochastic Environments

THESIS

Dean C. Wardell, Captain, USAF
AFIT/GE/ENG/06-55

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/06-55

APPLICATION OF FUZZY STATE AGGREGATION AND POLICY HILL CLIMBING TO MULTI-AGENT SYSTEMS IN STOCHASTIC ENVIRONMENTS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Dean C. Wardell, MS

Captain, USAF

March 2006

AFIT/GE/ENG/06-55

APPLICATION OF FUZZY STATE AGGREGATION AND POLICY HILL CLIMBING TO MULTI-AGENT SYSTEMS IN STOCHASTIC ENVIRONMENTS

Dean C. Wardell, MS
Captain, USAF

Approved:

/signed/	
Gilbert L. Peterson (Chairman)	date
/signed/	
Scott R. Graham (Member)	date
/signed/	
Christopher B. Mayer (Member)	date

AFIT/GE/ENG/06-55

Abstract

Reinforcement learning is one of the more attractive machine learning technologies, due to its unsupervised learning structure and ability to continually learn even as the operating environment changes. Applying this learning to multiple cooperative software agents (a multi-agent system) not only allows each individual agent to learn from its own experience, but also opens up the opportunity for the individual agents to learn from the other agents in the system, thus accelerating the rate of learning. This research presents the novel use of fuzzy state aggregation, as the means of function approximation, combined with the fastest policy hill climbing methods of Win or Lose Fast (WoLF) and policy-dynamics based WoLF (PD-WoLF). The combination of fast policy hill climbing and fuzzy state aggregation function approximation is tested in two stochastic environments; Tileworld and the simulated robot soccer domain, RoboCup. The Tileworld results demonstrate that a single agent using the combination of FSA and PHC learns quicker and performs better than combined fuzzy state aggregation and Q-learning reinforcement learning alone. Results from the multi-agent RoboCup domain again illustrate that the policy hill climbing algorithms perform better than Q-learning alone in a multi-agent environment. The learning is further enhanced by allowing the agents to share their experience through a weighted strategy sharing.

Table of Contents

List of Figures

List of Tables

APPLICATION OF FUZZY STATE AGGREGATION AND POLICY HILL CLIMBING TO MULTI-AGENT SYSTEMS IN STOCHASTIC ENVIRONMENTS

I. Introduction

1.1 Overview

From swarms of unmanned aerial vehicles to teams of game playing robots, the ability for multiple agents to learn and function in chaotic environments has been a challenge for researchers and engineers. One of the challenging traits of stochastic environments is the ever-increasing state-space size. Even with the higher speeds and memory capacity of today's computers, the sheer volume of information a control system must process and track, consisting of maintaining consistent state information, and action and event outcomes, places severe limits on algorithm performance. Dealing with the enormous state-spaces requires developing a smaller representation of the environment or a rapid search/learning algorithm or both.

As we become more reliant upon technology to perform tasks deemed too dangerous or too complicated for humans, the problems addressed by that technology are only becoming more difficult. Large-scale planning and scheduling, robotic explosive disposal and Martian surface exploration are just some examples of technology designed to operate in rapidly changing environments where human interaction is either too expensive or simply impossible. These environments present a host of variables, many of which are unpredictable, but which the software controller must handle and include in the decision making process. It is not reasonable to expect systems to exhaustively search over millions of possible states to determine the best solution, much less the next best step.

1.2 Background

The first and most common approach to dealing with the large stochastic environment state-space problem makes use of a function approximation of the domain's policy. This approach is also referred to as state generalization.

State generalization architectures provide a means to limit the size of the state space and approximate the learned policy. This approach has been presented in several previous efforts [1, 2, 3]. Rather than attempting to maintain the entire state or state/action policy representation state, generalization provides an intermediate approximation function that reduces the policy state space. The key elements in this procedure are that the function approximation must be able to adequately approximate the true policy representation and also reduce the state dimensionality to something more manageable. The state generalization procedure used in this work is *state aggregation* which is a type of function approximation in which the states of a domain are combined into groups with some common value estimate [1]. When a state is updated, the entire group is updated. Two of the best known methods for state aggregation are tile coding [1] and artificial neural networks [2].

Tile coding approximation generates several overlapping grids or tilings of the state space, such that any given point in the state space will lie in exactly one tile in each tiling. Thus, the representation of a point is represented by the set of tiles that it lies in. A variation on tile coding is Berenji and Vengerov's [4, 5] use of fuzzy state aggregation (FSA) as a means of effectively limiting the state space in a Q-learning experiment. Fuzzy state aggregation uses the concept of fuzzy sets to represent the environment with a limited number of "fuzzy states".

Artificial neural network (ANN) learning can be specified as a function approximation problem where the goal is to learn an unknown function (or a good approximation of it) from a set of input–output pairs. A variety of constructive neural-network learning algorithms have been proposed for solving the general function approximation problem. These algorithms typically use a greedy strategy wherein each new neuron added to the network is trained to minimize the residual error as much as possible. Thus, the artificial neural network can be used to limit the state space size of large domains.

The second method for improving the results of reinforcement learning, such as Q-learning, is the use of a separate policy table which tracks the probability of selecting an action from a given state. The off-policy reinforcement learning algorithm, policy hill climbing (PHC), yields improved empirical results over on-policy methods [1]. Bowling and Veloso [6] showed continued improvement over the standard policy hill climbing by separating the PHC's update value into two values, one which updates when winning and one when losing, hence the name of Win or Lose Fast (WoLF) policy hill climbing algorithm. When winning, the updates are more conservative (and therefore slower) in anticipation that the opponent may change its losing policy. The updates when losing are more aggressive, and therefore faster. Banjeree and Peng extend WoLF, introducing a policy dynamics based version of WoLF (PDWoLF), which compares the change in policy from the previous time step with the change in policy from the current time step, further improving results [7].

In the case of multi-agent domains additional progress has been made by allowing the agents to share information and the benefit of their experience via weighted strategy sharing (WSS) [8, 9]. Based on the value of their learned information, each agent's learning data is weighted and combined to provide each agent the benefit of information gathered by other agents in the domain.

1.3 Research Focus

This research presents the novel application of fuzzy state aggregation combined with three different policy hill-climbing algorithms described above, comparing the speed and efficacy of their learning in the highly stochastic Tileworld [10].

Tileworld is grid domain in which the agent tries to select the best reward opportunity while avoiding penalty spots. These rewards and penalties randomly disappear and re-appear in the domain making the environment very stochastic. The Tileworld tests are followed by experiments in the simulated robot soccer domain RoboCup in which multiple soccer playing agents make up teams and play against each other in a fairly realistic soccer simulation.

Fuzzy state aggregation and policy hill climbing techniques have each been applied separately in these domains, but not in combination. Based on the improved performance each has shown in past research, using them in combination should result in further advances in performance.

These domains provide stochastic environments in which multiple interactive agents can learn and operate. The simulators are readily accessible and enough research has been conducted using both Tileworld and RoboCup that there is a well established baseline for measuring performance. The results of this research demonstrate the improved performance of combining a policy hill climbing method with fuzzy state aggregation in both the Tileworld model and the RoboCup domain.

The results of the Tileworld experiment demonstrate the efficacy of combining fuzzy state aggregation with each of three simple policy hill-climbing algorithms to accomplish learning in this stochastic environment. This novel combination of FSA and PHC is applied to the RoboCup domain where learning is further improved by applying a weighted strategy sharing method which allows all agents to benefit from the experience of their team mates.

1.4 Thesis Organization

Chapter 2 of this thesis provides an overview of the various types of robot control, the different approaches used in robot soccer and the different methods and algorithms used in reinforcement learning, state generalization and weighted strategy sharing. Chapter 3 covers the implementation of combinations of FSA with PHC learning. The Tileworld domain and its use as an environment for a single-agent experiment are described. The specifics of how the team of robots operates in RoboCup along with the specifics of their reinforcement learning method complete the chapter. The results of applying the three variants of policy hill climbing to the Tileworld domain open Chapter 4. The results of applying the PHC algorithm combined with FSA and WSS in the RoboCup domain comprise the rest of Chapter 4. Chapter 5 contains a discussion of the conclusions and recommended future research areas.

II. Related Work

The research in this thesis touches on multiple domains, namely multi-agent systems, reinforcement learning, fuzzy state aggregation and multi-agent learning. This chapter reviews the extensive research accomplished in these fields, specifically:

- Reinforcement learning in its many forms has been widely researched. Since it is a key part this research, a review of the work and progress made to date in Q-learning and policy hill climbing provides the context for later sections of this paper.
- Separate, but equally important, are the various means of performing state generalization. The fuzzy state aggregation used in this research is itself not new; however an understanding of the research done in the past makes the application in this document more clear.
- By understanding the theories behind weighted strategy sharing, applying that technique to this specific research becomes a logical next step in improving the learning of multiple agents.
- The work done in Robot soccer teams which provides a sound basis for the multi-agent experiment conducted in this research effort using the RoboCup domain.

2.1 Q-Learning

In the realm of reinforcement learning, Q-learning [11] is one of the simplest and most commonly used methods. Q-learning assigns values to the actions, *a*, the agent can possibly take from a given state, *s*. These state-action pairs are represented by the symbol *Q*(*s,a*), and are usually stored in a Q table. After the algorithm selects an action, the Q table element containing that state-action pair is updated. The amount of the update is based on the rewards received and the expected rewards as represented by the Q value of the new possible actions *a'* the agent can take from the new state, *s'*, according to the function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{2.1}$$

where α is the learning rate (or step size), set between 0 and 1, which controls convergence, and γ is the discount factor, set between 0 and 1, which affects the value placed on rewards *r* that can be earned later.

In Q-learning, the agent learns through continuous interaction with the environment, during which it exploits what it has learned so far. At each step the algorithm searches for the next best step and is, therefore, referred to as a "greedy" algorithm. To ensure the agent is not missing more valuable state-action pairs, it can also explore. In practice, this means that the current approximation *Q* is used to select an action most of the time. However, in a small fraction of cases an action is selected randomly from the available choices, so as to explore and evaluate unseen state/action pairs.

2.2 Fuzzy Logic and Q-Learning in Multi-agent Architectures.

Fuzzy logic has been used to represent continuous state spaces as discrete, thereby making it possible to implement Q-learning in continuous state spaces. The combination of FLCs with Q-learning has been proposed as Fuzzy Q-Learning (FQL) for many single robot applications [12, 13, 14].

Gültekin and Arslan [15] present a modular-fuzzy cooperative algorithm for multi-agent systems which takes advantage of a modular architecture, internal model of other agents, and fuzzy logic in multi-agent systems. In this algorithm, the internal model provides an estimate of the agent's own action and evaluates other agents' actions. To overcome the problem of dealing with an extremely large state space, fuzzy logic is used in mapping from the input sets representing the state space of each learning module to the output sets representing the action space. The authors used Q-learning to build a fuzzy rule base for each learning module, but without providing any convergence proof.

Kilic and Arslan [16] developed a "Minimax" fuzzy Q-learning for cooperative multi-agent systems. In this system, the learning agent must observe other agents and use fuzzy state and goal representations to update the fuzzy Q-values. This Minimax refers to *min* and *max* fuzzy operators and is completely unrelated to the Minimax Q-learning described by Littman [17]. As with the system described above, this Minimax fuzzy Q-learning is not guaranteed to converge on an optimal solution.

2.3 Fuzzy Logic in the RoboCup Domain

Looking specifically at the robot soccer domain, there are several examples of fuzzy logic applications. First, applications of Fuzzy Logic Controllers (FLC) are discussed, followed by those fuzzy behavior-based implementations to include learning components.

One of the earliest applications of fuzzy logic in the RoboCup domain was the Zeng99 team [18]. The designers presented a Hierarchical Fuzzy Intelligent Control system (HiFIC) applied to the control of soccer agent behavior, planning and cooperative control. The HiFIC controller is a derivative of the three-layered control model presented by Jens Rasmussen [19].

The HiFIC consist of three levels: a lower layer to regulate primitive reaction control, a middle layer to perform skill level behavior and the highest layer to make decision on strategic and tactical play planning. Implementing this system revealed an information transfer problem between the different layers, each of which has information on a different level of abstraction. Though it did not use any type of reinforcement learning, this effort did serve as a test-bed for the implementation of a HiFIC in the robot soccer domain

In [20] Aguirre et al present a RoboCup team using fuzzy logic (but no learning) to select behaviors for each agent. The goalie, forward (offensive) and rear (defensive) players each use a different FLC to select between behaviors based on various state conditions. As an example, the goalie's conditions are the location of the ball, the proximity of opponents to the goal and the game score.

Team Milan [21] uses a fuzzy cognitive model to integrate coordination, planning and reactive behaviors in a team of cooperating robots. The authors use the representation of concepts as fuzzy predicates, i.e., logical predicates that map real-valued variables into the truth interval [0..1].

A team Milan robot is governed by a set of behavior modules, implemented as a set of fuzzy rules, and supported by a fuzzy behavior management system. Fuzzy predicates in the antecedents of the rules are evaluated to weight the actions proposed by the consequents. A set of conditions is associated with each behavior. They are represented by fuzzy predicates whose truth value is taken as an evaluation of the applicability of the behavior in the current state. If this applicability value is above a given threshold, the rules of the behavior module are evaluated and the corresponding actions are proposed.

Each action is associated with a weight computed as a fuzzy composition of the applicability and the weight coming from the fuzzy rule evaluation. Generally, several behaviors are activated at a time, there are many proposed actions. These are weighted by the *WANT* conditions, fuzzy predicates, associated to each behavior, that state the opportunity to do the actions proposed by a behavioral module.

In [22] Sng et al present a fuzzy logic based strategy which employs an arbiter that assigns a robot to attack (shoot or pass) the ball. The fuzzy logic based strategy is implemented for a five-a-side robot soccer game. Role assignment is necessary to avoid collision between players going for the ball or no player being assign such a role to attack the ball. Making this assignment purely on distance from the ball is insufficient for the dynamic soccer environment. Since, the main objective of the game is to score goals; so if a player is in a better position to secure a scoring chance, it must be given the opportunity. Parameters used as inputs to the fuzzy arbiter for each robot are *distanceToBall*, *orientation*, *shootAngle* and *pathObstacle*. Fuzzy logic rule based reasoning is used to decide which robot should 'attack the ball'. The teams described above do not use any reinforcement learning.

2.4 Fuzzy Logic Controllers and Q-Learning in the RoboCup Domain

When applying Fuzzy Q-Learning to the robot soccer domain it is often in the context of learning one particular skill or behavior. Gu and Hu [23] (and with Specek [24]) present a fuzzy logic controller (FLC) for the implementation of a ball chasing behavior for Sony Aibo robot. The FLC is refined using an adaptive heuristic Critic (AHC) reinforcement learning. The actor part of AHC is a conventional FLC in which the parameters of input membership functions are learned by an immediate internal reinforcement signal. This internal reinforcement signal comes from a prediction of the evaluation value of a policy and the external reinforcement signal. The evaluation value of a policy is learned by temporal difference (TD) learning in the critic part that is also represented by a FLC. A genetic algorithm (GA) is employed for learning internal reinforcement of the actor part.

These same authors present a fuzzy classifier system (FCS) to teach the agent to chase the ball [25] and to perform formation keeping [26], using wheeled robots. The FCS approach uses a learning approach based on a Q-learning credit assignment strategy. This approach adopts an inverse measure of the rule accuracy as a rule's fitness value. Each rule maintains a *q* value, but it is not an estimate of accumulated payoffs. It is only used for calculating the rule's accuracy. This *q* value is updated by the Q-learning mechanism. An on-line fuzzy Q-learning algorithm is used for the credit assignment. The max operator in the standard Q-learning is not used since the rules that have maximum *q* values no longer represent rules with the best payoffs. Finally the action selection mechanism employs a "niching" Genetic Algorithm which selects actions (*MOVE FORWARD, LFFT FORWARD, RIGHT FORWARD, LEFT TURN*, *RIGHT TURN,* or *STOP)*

In [27] Nakashima et al propose a fuzzy Q-learning method in which an agent determines its action based on the inference result by a fuzzy rule based system. The authors apply the method to a soccer agent that tries to learn to intercept a passed ball. The state space is represented by internal information that the learning agent maintains such as the relative velocity and the relative position of the ball to itself. The authors divide the state space into several fuzzy subspaces and define each fuzzy subspace by specifying the fuzzy partition of each axis of the state space. The learning agent receives a reward if the distance between the ball and the agent becomes smaller or if the agent catches up with the ball.

Ammerlaan and Wright [28] address the question of whether systems based on fuzzy logic can effectively adapt themselves to dynamic situations. To answer this question, they design and implement an adaptive fuzzy logic agent for playing RoboCup soccer. The agent has a FLC for basic behaviors, but a neural network allows the agent to adapt to the changes in the environment.

Rapidly changing environments usually generate very large state spaces. While the learning methods described here do provide a means for the agent to improve its performance, they do not address the size of the state space in which the agent must operate. This is left to the function approximation methods used in state aggregation and generalization.

2.5 State Aggregation and Generalization

State aggregation is a type of generalizing function approximation which allows machine learning to learn in larger environments more quickly. State aggregation works by combining the states of a domain into groups with some common value estimate [1]. When a state is updated, the entire group is updated. The best known methods for state aggregation are tile coding (also known as sparse coarse coding) [1] artificial neural networks [2], and fuzzy state aggregation [4, 5]. Since this thesis primarily uses fuzzy sets, the following section describes fuzzy state aggregation in depth.

2.5.1 Fuzzy State Aggregation

Fuzzy state aggregation uses Zadeh's [29] concept of fuzzy sets to represent the environment with a limited number of "fuzzy states". Fuzzy sets are sets that allow elements to be partially in more than one set at a time. The degree to which an element is a member of a fuzzy set is measured on a scale between 0 and 1.

For example, consider the outside ambient temperature [30]. Classical set theory can only classify the temperature as hot or cold (*i.e.*, either 1 or 0). It cannot interpret the temperature between 20 °F and 100 °F. In other words, the characteristic function for the classical logic for the above example is given by

$$\mu_{Hot}(x) = \begin{cases} 1 \ \mathit{iff} \ \ x \geq 50^{o} F & \mathit{classified} \ \ \mathit{as} \ \mathit{hot} \\ 0 \ \mathit{iff} \ \ x < 50^{o} F & \mathit{classified} \ \ \mathit{as} \ \mathit{cold} \end{cases}$$

The boundary 50 °F is taken because classical logic cannot interpret intermediate values. On the other hand, fuzzy logic solves the above problem with a membership function as given by

$$\mu_{Hot}(x) = \begin{cases} 0 & \mathit{if} \ \ x < 20^{o} F \\ \dfrac{x-20}{80} & \mathit{if} \ \ 20^{o} F \leq x \leq 100^{o} F \\ 1 & \mathit{if} \ \ x > 100^{o} F \end{cases}$$

The above membership function is graphed in Figure 2.1, demonstrating that the degree of coldness is the complement of the degree of hotness.

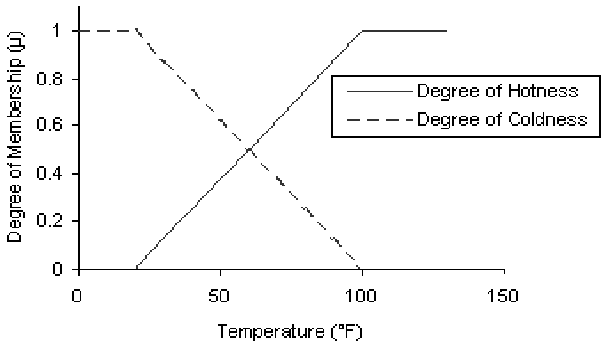

Figure 2.1: Membership Function for the Degree of Hotness and Degree of Coldness [30]

Fuzzy state aggregation is a variation of Singh's soft state aggregation [3], which uses probability values as a measure of the extent to which the current state falls into the various aggregate (cluster) states.

Like soft state aggregation, fuzzy state aggregation uses a fixed number (*K*) of aggregate states to represent the environment and thus minimize the number states the learning algorithm must deal with.

Rather than using probabilities, a crisp state (*s*) is represented by its degree of simultaneous membership in each of the *K* fuzzy states. The total number of fuzzy states is determined by the number of fuzzy sets (labels) used and the number of state variables.

2.5.2 Combining State Aggregation with Q-learning

In a domain with a large state-space, it is very memory inefficient to learn separate Q-values for each state-action pair. Therefore, it is not uncommon to see Q-learning used in conjunction with some form of state aggregation. When implementing Q-learning with such an architecture, the term *Q(s,a,r)* is used to approximate *Q(s,a.)* Here *r* is a vector of the learned parameters. The fundamental parameter updating rule for each time step *t* is [4]

$$r_t \leftarrow r_t + \alpha\delta_t \Delta r_t Q(s_t, a, r_t)\,. \tag{2.2}$$

Where α is the learning rate and δ_t is the Bellman error used for the look-up table in this corresponding learning rule

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha\delta_t\,. \tag{2.3}$$

In discounted Q-learning the Bellman error is calculated as follows

$$\delta_t = g(t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a)\,. \tag{2.4}$$

Where *g(t)* is the cost of taking the specific action and γ is the discount rate.

In this work fuzzy state aggregation is the specific function approximation architecture. Using this architecture, the Q-value of action *a* in state *s* is calculated using:

$$Q(s,a) = \sum_{k=1}^{K} q(k)\mu_k(s,a) \tag{2.5}$$

Where *q(k)* is the Q-value of the *k*th fuzzy state and $\mu_k(s,a)$ is the degree of membership of state *s* to *k* with respect to action *a*.

Replacing $\Delta r_t Q(s_t,a,r_t)$ from equation (2.2) with $\mu_k(s,a)$, the equation to update *q(k)* becomes:

$$\forall_{k \in K} \quad q(k) \leftarrow q(k) + \alpha\delta_t \mu_k(s,a) \tag{2.6}$$

Otherwise, the Q-learning algorithm remains unchanged. Also note that in performing this update that all of the fuzzy sets are updated based upon their degree of membership in the state representation not just the most similar fuzzy set. The value of $\mu_k(s,a)$ determines how much weight is applied to that particular update.

2.6 Policy Hill Climbing

Policy Hill Climbing (PHC) is an extension of Q-learning. The algorithm, performs hill-climbing (seeking the highest global reward) in the space of mixed policies. Q-values are maintained as an estimate of the optimal policy. In addition to the Q-table ($Q(s,a)$), the algorithm maintains the current mixed policy ($\pi(s,a)$). The PHC algorithm is shown in Table 1.

Table 1: Basic Policy Hill Climbing Algorithm

```
Input $\alpha \in (0,1], \quad \delta \in (0,1].$
Initialize $Q(s,a) \leftarrow 0$ and $\pi(s,a) \leftarrow \frac{1}{|A|}.$
Loop
    from state $s$, select action $a$ according to $\pi(s,a)$
    Observe $r$ and next state $s'$ and update $Q(s,a)$
    compute
        $\Delta_{sa} = \begin{cases} -\delta_{sa} & \text{if } a \neq \max_a Q(s,a') \\ \sum_{a' \neq a \in A} \delta_{sa} & \text{otherwise} \end{cases}$
        where $\delta_{sa} = \min\left(\pi(s,a), \frac{\delta}{|A|-1}\right)$
Update $\pi(s,a) \leftarrow \pi(s,a) + \Delta_{sa}$
```

In PHC, the policy is improved by increasing the probability that it selects the highest valued action according to a learning rate $\delta \in (0,1]$. If $\delta = 1$ the algorithm is equivalent to Q-learning, since with each step the policy moves to the greedy policy, always executing the highest valued next step rather than pursuing the greatest overall reward.

*2.6.1 Win or Lose Fast (WoLF-PHC***)**

The WoLF-PHC [6] algorithm is a policy hill climber that uses variable learning rates instead of a single learning rate (δ in Table 1). The algorithm requires two learning parameters δ_l and δ_w. where $\delta_l > \delta_w$ as shown in Table 2. The parameter that is used to update the policy depends on whether the agent thinks it is currently winning or losing. This determination is made by comparing whether the current expected value is greater than the current expected value of an average policy. If the current expected value is lower (i.e., the agent is "losing"), then the larger learning rate δ_l is used, otherwise δ_w is used. The purpose of using the variable learning rate is to increase the speed at which the algorithm reaches the optimum policy. The functions in Table 2 are used to calculate δ for the WoLF-PHC algorithm, and are the only changes to PHC in Table 1.

Table 2: Additional Functions for WoLF-PHC Algorithm

$$\bar{\pi}(s,a) \leftarrow \bar{\pi}(s,a) + \left(\frac{\pi(s,a) - \bar{\pi}(s,a)}{C(s)} \right)$$

$$\delta = \begin{cases} \delta_w & if \ \sum_a^A \pi(s,a)Q(s,a) > \sum_a^A \bar{\pi}(s,a)Q(s,a) \\ \delta_l & otherwise \end{cases}$$

2.6.3 Policy Dynamics Based Win or Lose Fast (PDWoLF)

Like WoLF, PDWoLF [7] uses the variable learning rate parameters δ_l and δ_w. Where WoLF checks itself against an average policy to determine if it is winning or losing, PD-WoLF uses the change in policy from the previous time step $\Delta^2(s,a)$ and the change in policy from the current time step $\Delta(s,a)$. If both changes are in the same direction (both positive or both negative) the agent believes it is losing and selects the larger learning rate δ_l as shown in Table 3.

Table 3: Additional Functions Used in PDWoLF-PHC Algorithm

$$\delta = \begin{cases} \delta_w & \text{if } \Delta(s,a) \times \Delta^2(s,a) < 0 \\ \delta_l & \text{otherwise} \end{cases}$$

$$\text{where} \quad \Delta^2 \leftarrow \Delta_{sa} - \Delta(s,a), \text{ and } \Delta(s,a) \leftarrow \Delta_{sa}$$

Cousin and Peterson [9] successfully demonstrate increased performance over WoLF by applying PD-WoLF to a multi-agent hunter-prey domain. They further improve the learning performance by implementing weighted strategy sharing along with the PD-WoLF PHC.

2.7 Weighted Strategy Sharing

First introduced by Ahmadabadi et al [8], Weighted Strategy Sharing (WSS) is a method in which a group of *n* homogeneous agents learn in a particular environment using two modes: individual learning and cooperative learning. Initially all agents learn individually, executing some number of learning trials based on Q-learning. After a specified number of individual learning trials, the agents switch to the cooperative learning mode.

In this mode each agent assigns weights to the other agents based on that other agent's expertness. The agent then takes the weighted average of the Q-tables and uses that as its own new Q-table. Agent *(i)* would calculate the new Q-table based on weighted values from the *n* other agents *(j)* as shown in equation 2.7.

$$Q_i^{new} \leftarrow \sum_{j=1}^{n} (W_{ij} * Q_j^{old}) \qquad (2.7)$$

This new Q-table is then used as the starting point for subsequent learning.

Thus far this chapter has reviewed the related research conducted in state generalization and reinforcement learning. While interesting, these theories and algorithms require application in some domain to be useful. The balance of this chapter reviews the related work performed robot controllers and specifically robots designed for playing soccer, which is the domain of choice for this thesis.

2.8 Robot Control Designs

Agents built for the RoboCup competition usually differ from other software agents. Instead of sophisticated communications systems and strict pre-programmed steps, RoboCup agents must have the ability to react to changing situations, make team oriented decisions in real-time and make difficult decisions on uncertain information. This has resulted in a wide variety of team designs, ranging from variations on the classic Belief-Desire-Intention (BDI) architecture to simple reactions and from explicit team coordination protocols to hard-coded player positions.

2.9 Individual Agent Architecture

The focus of this section is on the balance between deliberation and reaction in the single-agent architectures. In general, a deliberative agent is one in which plans and goals are explicitly represented, requiring that the designer generally create some sort of plan domain which the agent then uses in instantiating a full plan at runtime. In performing planning, a form of symbolic reasoning is usually used and the building of the plan is a long and slow process.

In contrast, reactive agents do not explicitly represent goals, instead merely reacting blindly to the current state of the environment. In creating reactive agents, the designer aims to create reactions such that the overall observed behavior of the agent seems intelligent. The reality is the reactive agent either will or will not respond to a given stimulus. A reactive agent is like the motion-activated light on many homes, when the circuitry is triggered, the light comes on. There is no planning, deliberating or deciding – only reaction. The casual observer may see this reaction and think there is some level of intelligence at work, when, in fact, there is none. A reactive soccer-playing agent may be coded to respond to the presence of the ball, opponents and teammates. Given the use of enough *if – else* statements, the agent may even function in a way that appears intelligent, but lacks any element of intelligence.

The agents implemented for robot soccer are not at the extreme ends of this reactive - deliberative spectrum. Rather, in true Aristotelian fashion, they are usually somewhere near the middle. The following section reviews some robot soccer architectures beginning with the more deliberative designs of UM-PRS and CMUnited. The middle ground is represented by the ROGI and MAPs teams and the more reactive designs of MICROB, GAMMA and Scerri round out the review.

2.9.1 UM-PRS

One of the more clearly deliberative architectures is the University of Michigan's Procedural Reasoning System (UM-PRS) [31], which uses explicitly coded multi-agent plans on top of a reactive individual control system.

UM-PRS is based on the PRS architecture. Each agent has a database (world model), a set of goals (aka plans or plays), knowledge areas (procedures for achieving goals) an intention structure (run-time stack of the system) and an interpreter which controls the system. The interpreter looks at the database, identifies the knowledge areas (actions) to apply and places them on the intention structure for execution in pursuit of a goal.

The authors do not specify the particular behaviors, but do indicate that when no plan/play is applicable for the situation the interpreter falls back to a simple reactive behavior like running for the ball. This allows the player's behavior to degrade gracefully when unexpected situations occur.

2.9.2 CMUnited-98

Like the UM-PRS design, Carnegie Mellon's CMUnited-98 robot team [32] is a mixture of deliberation and reaction. In this architecture, each agent maintains a concept of the world state, internal state (e.g. agent's role in a particular formation), internal behaviors (update internal state), external behaviors (actions sent to actuators) and the locker room agreements as shown in Figure 2.2.

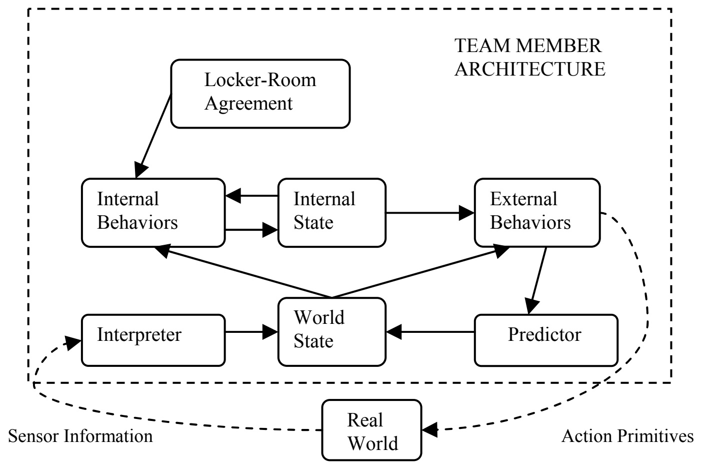

Figure 2.2: A functional input/output model of the CMUnited team member agent architecture [32].

Locker room agreements are established off-line and define the teamwork structure and communication protocols the team will use.

The individual behaviors (ball interception, dribbling, kicking, goal tending, defending and clearing) are in the basic level of the multi-layered hierarchy of behaviors illustrated in Figure 2.3.

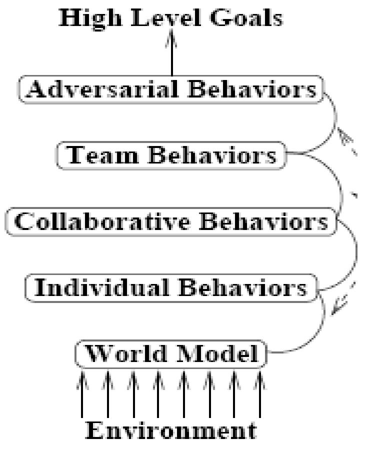

Figure 2.3: Hierarchy of Behaviors in the CMUnited-98 Architecture [32]

The *Individual Behaviors* are predictive, locally optimal skills. They use predicted world models and predicted effects to determine the optimal behavior. *Collaborative Behaviors* are those which involve other teammates, like passing the ball. The only behavior that is completely reactive (strategic positioning) is a *Team Behavior*. Using attraction and repulsion the agents position themselves autonomously, and the agent with the ball decides autonomously where to pass: no negotiation is involved, enabling the players to act as quickly as possible.

*2.9.3 **Ro**botico universitat de **Gi**rona (RoGi)*

Team RoGi [33] is built using agent-oriented programming (AOP) [34] vs. object-oriented programming. Agent-oriented programming is a fairly new programming paradigm that supports a societal view of computation. In AOP, objects known as agents interact to achieve individual goals. Agents exist in a structure as complex as a global internet or one as simple as a module of a common program. Agents can be autonomous entities, deciding their next step without the interference of a user, or they can be controllable, serving as a mediary between the user and another agent.

At the lowest level each agent in the RoGi architecture “instinctively” decides on a private action. This decision is based on the agent’s beliefs as calculated from its distance to the ball and distance to the goal. Each agent then informs its team mates of its intentions. In those situations where two or more agents have conflicting intentions (e.g. 2 agents intend to get the ball) one of the two will opt to change behaviors to avoid the conflict. This decision is based on the “certainty” value each has associated with their announced decision. The certainty value is obtained from a fuzzy inference calculation. This architecture relies heavily on communication between teammates to declare intentions and make decisions on a cooperative basis.

At the individual agent level, this architecture provides private actions: shoot the ball (SHOOT), get the ball (GET), move forward (FORW), and go backwards (BACK). It also supports communicative actions: send a decision to a specified soccer player (INFORM), and request an action to another soccer player (REQUEST).

2.9.4 MAPs

The Multi-Agent Planning system (MAPs) [35] consists of a high level planning algorithm that analyzes game play and sends instructions to the command interpreters. The robots are then expected to carry out the plan. The particular form of the instructions is not specified. This architecture uses potential fields as a means of weighting options the planner must choose between. The algorithm has 3 steps;

1. **Get new information** - Update the world model based on sensor data.
2. **Choose an Action** – Based on the influence of agents and obstacles, as represented in the potential field, select the next action. This action is chosen to minimize the interference between agents. The action choice is built from the perspective of the team’s goal.
3. **Find the Location for the Action** – using potential fields and the type of action selected, the individual agent will determine the best location for the action.

The soccer robot example builds potential fields out of the elements described below. Each element is designed to provide low values at attractive regions, and high values at unattractive regions. The elements are combined on a grid array that represents the physical environment such that coordinates are easily extracted. Some elements cover the whole area, while others influence only part.

1. **Base Field -** The base field mask is a biased towards the goal of the system. When looking for low valued coordinates, this has the effect of encouraging the agents to move towards areas of interest. The opposition goal is at the base of the ramp and this representation encodes the desire for the agents to carry out their directives towards it.
2. **Object Regions -** This is a field mask that represents an object's presence in the working field. This mask is relatively small in area and is placed wherever the objects are on the field. The masks represent the objects' locations, and regions around them considered to be their influence zone. This can be used for robot locations or obstacles.
3. **Robot's Position -** It may be desired to have the robots maintain responsibility in specific areas. In the soccer robot system, this is used to keep players in their specific field positions (e.g., left wing).
4. **Distance From Current Position -** This function is added to prevent the planner from selecting coordinates on the boundaries of the potential field. It creates a field-wide virtual "dish" encouraging the selected coordinates to be close to the current position of the object in question.
5. **High Value Continuation -** In some cases, the planner evaluates the field in a line-of-sight manner similar to the *clear path to object* function described below. This is carried out by adding the highest valued coordinate to all other coordinate values from the object to the boundary of the field which has the effect of building a 'shadow' of high values where it is undesirable for the agent to go.
6. **Clear Path To Object -** It is important for a robot navigating to positions in the environment to have a clear view of certain objects otherwise their location can't serve any purpose. This function is represented in the potential field by making all occluded coordinates high values. From the robot's perspective the areas blocked by an obstacle (another player) are undesirable.

Like Team ROGI, the MAPs architecture is implemented on a physical robot league which means the system has access to a global view of the environment and is allowed to use off-board processing. For a given situation the planner generates a potential field, starting with a basic field biased towards the goal. Other potential field elements are then add to this one resulting in a composition field which represents the soccer ground, the positions of opponents and teammates as well as clear paths to desirable locations. The potential field info is then passed to the agent.

2.9.5 MICROB

Perhaps one of the more interesting reactive architectures is the MICROB robotic soccer team which is an implementation of the Cassiopeia programming method [36]. The goal of this design is to demonstrate that a set of robots can achieve intelligent behavior without being provided with the general solution of the problem. The designers focused on the emergence of "self-organized collective phenomenon in societies of robots" [37], where each agent is provided with minimal individual capacities, in terms of communication, interaction or cooperation.

This design is made up of 4 basic behaviors; the authors have used them to compose the more complex sequences of behaviors [38].

1. Shoot the ball (in the direction of the goal)
2. Take up their position (waiting for a pass)
3. Block an opponent's way
4. Defend their goal (against the opponents' attacks)

The authors' don't point to any particular "learning". Rather, they recognize that some behaviors depend upon, or are influenced by, others. This led to the design of four different types of agents (roles). At any given time, one agent may be filling any of the four roles. The determination of which role the agent is filling is made by which roles the other agents are filling. To better visualize these complex interactions, the coupling and influence graphs in Figure 2.4 and Figure 2.5 illustrate these relationships.

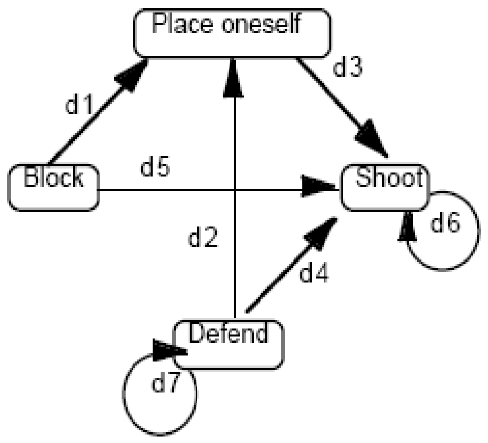

Figure 2.4: Coupling Graph of the MICROB Soccer Robot Game [38]

The following dependencies have been identified (an arrow from A to B means that B potentially depends on A):

d1. Blocking an opponent can help another robot to better place itself.
d2. Defending can help oneself or another robot to better place itself.
d3. It may be necessary to place oneself if another robot wants to shoot.
d4. Defending may allow agent to catch the ball of the opponent.
d5. Blocking can help oneself or another robot to shoot the ball.
d6. Shooting can help oneself or another robot to shoot (this is the pass).
d7. Defending depends on the other robots' defense strategy.

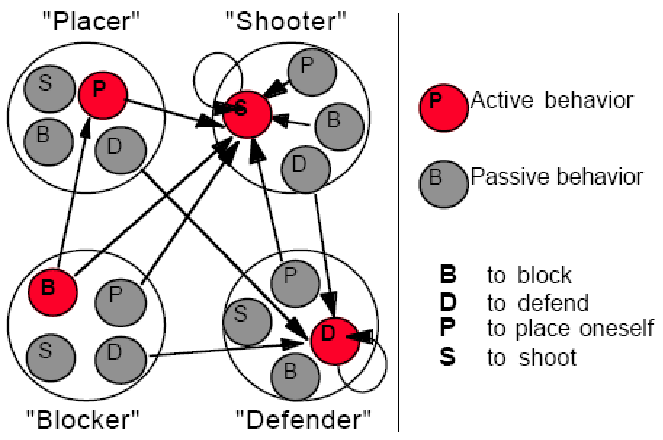

Figure 2.5: Influence Graph for the MICROB Soccer Robot Team [37]

The behaviors that allow the agents to dynamically organize themselves according to their mutual influences are based on the contract net mechanism described in the multi-agent portion of this survey.

2.9.6 GAMMA

Moving further to the reactive side of the spectrum, Team GAMMA [39] uses a variation on the subsumption architecture written in Gaea. Gaea is an "organic programming" language which allows the creation of entire teams of agents. Each agent uses an extension to the subsumption architecture to choose actions. The standard subsumption architecture has a number of layers that subsume or inhibit each other whenever they are activated by sensors connected to the world [39]. The extension used in Gaea, called dynamic subsumption architecture, is the combination of subsumption architecture and dynamic environment change. Since subsumption architecture assumes fixed layers of functions, it is either difficult or inefficient to implement multiple modes on top of it. It is straightforward in organic programming, using context reflection. An agent may respond to a given input differently according to the mode (e.g. offense vs. defense). The mode changes as a result of the agent's action and plan, and according to the changes in the situation (context).

In Gaea a single agent is programmed in a multi-agent manner. A player consists of the following processes (agents):

1. **Sensor Process**: receives sensor information sent from the Soccer Server, analyzes it, and puts the results into the common cell.
2. **Command Process**: sends control commands to the Soccer Server. It is designed to send one command every 100 milliseconds, because the RoboCup Soccer Server only accepts one command per 100 milliseconds. In other words, this process is a resource manager of sending control commands.
3. **Action Process**: controls low-level modes of player's action by manipulating the environment of the command process.
4. **Object Detection Process**: checks the path to the target for chasing or kicking, and changes the behavior by modifying the environment of the command process. if there are objects on the way,

5. **Communication Process**: controls high-level modes of player's action according to the message from the referee and teammates.

The top level of each process is a loop that repeatedly calls "cycle(top)". It is defined in the "basic" cell that is shared by all processes.

2.9.7 Scerri

Paul Scerri [40] presented a multi-layered approach to robotic soccer. Each layer consists of three controlling processes; *action selection, behavior instantiation* and *information extraction*. Since all layers work in the same manner, the only difference is which layer is active at the moment. Each layer deals with a level of abstraction greater than the one below it. Only the lowest level interfaces with the outside world while the layers above it have their effect by setting behaviors in the levels below. For example, a low-level “move-to-ball” behavior is given the ball’s precise location and sends the agent to it, while a high-level “defend” behavior knows only that the ball is in the defensive half of the field and will implement lower level behaviors in response to that input. In this architecture a behavior is created dynamically, by instantiation, from a generic behavior and a parameter component. Each behavior acts independently of the other behaviors at that layer. The author did not provide a comprehensive list of all the behaviors, but cite some in examples. The behavior *attack-down-wing* is listed as an example of a high level behavior, while *move-to-the-ball, move-to-the–wing* and *kick-goa*l are examples of low level behaviors.

Only the behavior chosen by the action selection process of a particular layer is executed by the system. This “best” behavior is chosen by its score, a sum of:

- Applicability Value – score for how applicable a behavior is given the current state
- Priority Value – score based on rank ordering of behaviors
- Persistence Value – used as a tie breaker to prevent oscillations between different behaviors.

All of these scores are set by the designer. The intent is for the designer to set up situations, with a relatively small number of simple behaviors, where a wide range of different and complex behaviors will emerge.

2.10 Multi-Agent Interaction

With an elementary understanding of the architecture implemented by each of these teams, and since RoboCup consists of multiple agents working together, it is useful to compare and contrast how each system deals with inter-agent activities (e.g. assigning roles and positions, calling plays, making plans, etc).

2.10.1 UM-PRS

The UM-PRS architecture treats roles as the first deliberative step above basic reactive behavior. The roles (such as goalie, defender, attacker etc) can be assigned by player number or by field position. The next step up is the use of formations. The authors acknowledge the use of formations, but do not discuss their specific implementation. With roles and formations established, the next step is to make and execute plans. Using relatively simple plans, the PRS architecture's interpreter can draw upon state knowledge and current player positions to select and execute appropriate plans (goals). Those agents not involved in a plan will degenerate to reactive behavior, rather than be paralyzed, until a plan is invoked. Since each agent carries its own "copy" of the architecture, the agents use limited communications to update each others' databases and intention stacks. This communication in conjunction with sensor input from the environment is how individual agents coordinate with each other.

2.10.2 CMUnited-98

The CMU simulator team is based on flexible formations made up of flexible roles. Both are independent of the agent filling the particular role. There are predetermined conditions under which a particular formation may be used. Specific pre-planned plays are determined in the locker room agreements. Rather than rely on on-field communications, the agents have the pre-set, multi-agent, multi-step plays (locker room agreements) in an internal play book and know to execute them when the preconditions are met.

2.10.3 RoGi

The agent-oriented design used by de la Rosa, et al, uses a step-by-step reasoning for inter agent coordination. After making an initial decision of an action to take for itself, the agent will communicate it to its team mates. The agents also request information from all other agents to learn the proposed action and certainty for each team mate. In this cooperative mode conflicting intentions are worked out. The final decision is then communicated to the coach-agent. This coach-agent has global vision of the environment and uses a fusion procedure to determine and direct which agent should take the proposed action. It also will direct an agent to take a required action if none of the players have proposed it on their own.

2.10.4 MAPs

MAPs uses a less rigid approach to planning. Referred to as “plan-as-communication”, the agents are given a high level version of a plan and are free to alter it to better fit the situation.
Conspicuous by its absence is any mention of team strategy or a coordinated plan. The choice of action and its location are determined by components of the environment as represented on the potential field. In other words, the MAPs architecture relies on the use of the common potential field to be the de facto means of coordinating multiple agent actions. The common potential field is a compilation of fields. The base field simply points towards the goal. This base field is modified with other fields depending on the game situation. As an example, an agent in possession of the ball use the base field combined with a field pointing toward other teammates and away from opponents to determine where, and how hard, to kick the ball.

2.10.5 MICROB [37]

In order to distribute the basic behaviors among the players within a team, agents enter into contracts with each other. Since only one agent can be the shooter at a given time, it must decide whether to allow another robot to shoot, and which robot is going to be the blocker. This kind of collaboration is implemented by a kind of contract-passing between the agents. Agents outside this "contract-net" automatically become the defenders. The authors did not provide details as to the values/utilities used in these contracts.

In an effort to reduce the communication between the agents, the authors have also implemented virtual agents that possess (and reason about) a very simple model of the other robots. Whenever an agent has to make a decision, it can enter into contracts with these virtual agents, as if they were contract-based robots, and make decisions based on that interaction.

2.10.6 GAMMA and Scerri

Both team GAMMA and Paul Scerri's multi-layered design use a subsumption architecture and rely on the "intelligent" decisions of individual agents to naturally result in a kind of coordinated multi-agent effort. In simple English, neither design implements a deliberate means of planning or coordinating the actions of multiple agents. Instead they appear to follow the ideas of pre-Nash economic theory, which states that each individual acts in the manner most advantageous to himself results in the realization of the best results for the collective group.

2.11 Learning & Decision Making

Many existing machine learning (ML) techniques are applicable in multi-agent scenarios simply by duplicating the single-agent technique in each agent. However *multi-agent learning* is more concerned with learning issues that arise because of the multi-agent aspect of a given domain. As described by Weiss, multi-agent learning is "learning that is done by several agents and that becomes possible only because several agents are present" [41].

Among the teams considered thus far, only the most deliberative (UM-PRS and CMUnited) use machine learning. Most provide a means of making behavioral decisions, even if no learning is involved.

2.11.1 UM-PRS – Feedback and Opponent Modeling

In UM-PRS, goals are broken down into sub goals, and contexts are used to find the most appropriate action that achieves that goal or sub goal.

In some situations there are two or more actions which are equally appropriate for pursuit of a goal or sub goal. In these situations, UM-PRS uses both satisfaction of contexts and priority values given to actions and procedures to decide what to do. These priority values can be modified at runtime to allow the agent to adapt by learning which action is preferable based on experience (by observing its outcome).

Feedback is key to successful learning in this, or any other domain. Once the system gathers feedback for an action, it is used to reorder the set of weights on alternative procedures which satisfy the same goal. The authors do not specify the exact type of feedback used in this system. The weights referred to may be probabilities or a simple Q-learning. The authors do point out an advantage of using UMPRS as a real-time architecture for RoboCup is that UM-PRS itself can be extended to automatically generate Bayesian networks for plan recognition [42]. This allows for both improved agent collaboration and the ability to model and predict the goals and actions of opponents.

2.11.2 CMUnited's Layered Learning

Once the world model is successfully created, the agents must use it to respond effectively to the environment. As described previously, internal behaviors update the internal state while external behaviors produce executable actuator commands. Spanning both internal and external behaviors, *layered learning* [43, 44] is CMU's bottom-up hierarchical approach to client behaviors that allows for machine learning at the various levels. The key points of the layered learning technique are as follows:

- A bottom up, hierarchical task decomposition is given.
- Machine learning exploits data to train and/or adapt. Learning occurs separately at each level.
- The output of learning in one layer feeds into the next layer.

The type of learning used at each level depends upon the task characteristics. The authors have used neural networks and decision trees to learn ball interception and passing respectively [43]. These offline approaches are appropriate for opponent independent tasks trainable outside of game situations. They also use online reinforcement learning approaches for behaviors that depend on the opponents [45].

2.11.3 ROGI – Fuzzy Inference

Though Team ROGI doesn't use explicit machine learning, every agent has perception and communication capabilities, as well as decision capabilities. Since all the micro-robots of this project have the same technical specifications, various reactive models (behavior roles, such as, defense, attack, and goal keeper) are programmed in.

In the first step of the reasoning procedure, every soccer player decides a private action instinctively. This decision depends on local environment configuration (Beliefs) defined by two parameters: distance player-ball (DPB), and distance player-goal (DPG). The decision also has a degree of certainty, derived from a fuzzy inference calculation. This calculation is based on the membership values of established fuzzy sets, no learning or adjusting of the membership values occurs. This is one of the few robot soccer designs that implements fuzzy logic. The fuzzy logic controller and fuzzy state aggregation used in this thesis are similar to the fuzzy inference calculator referenced here.

2.11.4 MICROB, GAMMA and Scerri

These more reactive architectures do not use machine learning. The decision making process for selecting an action or behavior to be invoked was previously described in their respective architecture descriptions.

2.11.5 Other Examples

In their survey of multi-agent architectures [46] Stone and Veloso make reference to some of the systems described here as well as citing examples of the types of learning used by other systems which could be applicable to the RoboCup domain, including local or global perspective, opponent modeling, affecting other agents, roles, and communication content.

Local or global perspective

On the surface it may seem that having a global perspective would always be preferable to only a local perspective. However, the global perspective also carries with it a greater volume of information which may actually impede performance. This type of improved performance by agents with less knowledge is sometimes referred to as “Ignorance is Bliss.”

Modeling of Other Agents’ States

Modeling of other agents can be used to allow better interaction between cooperative agents or to allow competitive agents to better anticipate their opponent’s next action.

Affecting others

When communication is impossible, or at least very expensive, agents cannot interact with each other directly. However, since they exist in the same environment, the agents affect each other indirectly in several ways. One agent may change something in the environment and that change is perceived by another agent which then acts differently because of the perceived change.

Roles

When agents are organized into a team, each agent plays a separate role within the team. This requires some form of role assignment. This assignment might be obvious if the agents are very specific and can each only do one thing. However in some domains, the agents are flexible enough to fill one of many roles.

Communication Content

One important consideration for communicating agents is *what* they should communicate. For planning purposes, some agents may need to communicate goals while others can advance their learning by communicating state information, much like the weighted strategy sharing used in this research.

From the broad research areas of multi-agent robot architectures, reinforcement learning and function approximation this chapter has reviewed particular soccer-playing robot systems, Q-learning, policy hill climbing, information sharing and state generalization. Further narrowing the focus of this thesis, the subsequent chapters address the specific implementations of fuzzy state aggregation combined with PHC and weighted strategy sharing in the RoboCup domain along with the resultant performance in learning.

III. Implementation

The use of state aggregation for function approximation with Q-learning is not a new or unusual concept [1, 3]. Berenji and Vengerov [4, 5] advanced this work in their application of Q-learning and fuzzy state aggregation. The Proof-of-Concept experiment is built upon their work, beginning with fuzzy state aggregation and a basic Q-learning algorithm and extending that to the application of PHC algorithms. With the state-space constrained to *K* total fuzzy states, three different variants of a Policy Hill Climbing algorithm; standard PHC, Win or Lose Fast (WoLF) PHC and Policy Dynamics (PD) WoLF-PHC are applied. The implementation of these algorithms uses two vectors representing the learned parameter data. The *q*-vector *q(k)* as described in section 2.5.2 and a policy vector *π(k)* explained in section 3.1.

The following section lays the theoretic groundwork for applying the policy hill climbing algorithms to fuzzy state aggregation. This is followed by a detailed description of the Tile World domain and the specific test methodology for single-agent learning. Section 3.5 discusses the particular application of PHC and FSA to the RoboCup simulation domain and the specifics of using weighted strategy sharing with the multi-agent architecture.

3.1 Applying PHC to FSA

The *q*-vector holds the expected reward over time which is iteratively updated using a common temporal-difference formula. The *π*-vector holds the probabilities used to select an action from a given state (the policy). The policy decision of which action to take next is then based on both the expected reward value (*q*) and the policy value (π):

$$\Pi(s,a) = \sum_{k=1}^{K} \pi(k) q(k) \mu_k(s,a) \tag{3.1}$$

The vectors *q(k)* and *π(k)* are initialized as shown below:

$$q(k) \leftarrow 20 \text{ and } \pi(k) \leftarrow \frac{\#\,of\ fuzzy\ labels}{|AK|} \tag{3.2}$$

where *A* is the number of possible actions in the domain. The reason for initializing *π(k)* this way may not be intuitively obvious. Since this particular implementation uses three fuzzy labels, the initial value of each element of $\mu_k(s,a)$ is 1/3 before learning begins. The elements of *π(k)* are initialized so that

$$\sum_{a=1}^{A}\sum_{k=1}^{K}\pi(k)\mu_k(s,a) = 1 \tag{3.3}$$

Normalizing it with a Boltzmann distribution to ensure equation 3.3 remains true, the π-vector is updated as follows:

$$\forall_{k\in K} \quad \pi(k) \leftarrow \pi(k) + \left[\frac{\Delta_{sa}}{K}\mu_k(s,a)\right] \tag{3.4}$$

with Δ_{sa} calculated as:

$$\Delta_{sa} = \begin{cases} -\delta_{sa} & if\ a \neq \max_a Q(s,a') \\ \sum_{a'\neq a\in A}\delta_{sa} & otherwise \end{cases} \tag{3.5}$$

where

$$\delta_{sa} = \min\left(\sum_{k=1}^{K}\pi(k)\mu_k(s,a), \frac{\delta}{|A|-1}\right) \tag{3.6}$$

In this application δ is set in the range (0,1].

Because the intent is to use Δ_{sa} to update the entire summation

$$\sum_{k=1}^{K}\pi(k)\mu_k(s,a) = \sum_{k=1}^{K}\pi(k)\mu_k(s,a) + \Delta_{sa} \tag{3.7}$$

for a given action (*a),* the division used in equation (3.4) is necessary to scale and weight Δ_{sa} correctly and prevent it from causing disproportionate growth in the elements of *π(k)*.

To clarify, consider a very simple example using ambient air temperature. Assume an agent is learning which of two possible directions to move to find the warmest state in its environment. The agent uses temperature as a state variable when considering which direction to move (i.e. which action to take). Assume also that the agent has the q-vector and *π*-vector depicted in figure 3.1, which shows a higher expected reward for moving to a *High* temperature fuzzy state than for moving to a *Low* temperature fuzzy state:

	High	**Low**
q()=	6.2	4.7

	High	**Low**
π()	0.52	0.43

Figure 3.1: Notional q-vector and *π*-vector for Temperature Agent

The agent considers its first possible action (a_1) and finds that action would put it into a state with a temperature of 45°F. The other possible action (a_2) would put the agent into a state with a temperature of 70°F. Using the fuzzy labels in figure 2.1, calculating $\mu_k(s,a)$ for each action results in

$$\mu_{High}(s,a_1) = 0.29 \qquad \mu_{Low}(s,a_1) = 0.71$$
$$\mu_{High}(s,a_2) = 0.63 \qquad \mu_{Low}(s,a_2) = 0.37$$

The decision of which action to take is then based on the outcome of equation 3.1 using this information:

$$\Pi(s,a_1) = (0.52)(6.2)(0.29) + (0.43)(4.7)(0.71) = 2.3699$$
$$\Pi(s,a_2) = (0.52)(6.2)(0.63) + (0.43)(4.7)(0.37) = 2.7789$$

Based on these calculations, the agent would select a_2. The q-vector is then updated using equation 2.6. For this example, let α=0.1 and δ_t=0.85. The q-vector updates are then

$$q(H) = 6.2 + (0.1)(0.85)(0.63) = 6.25355$$
$$q(L) = 4.7 + (0.1)(0.85)(0.27) = 4.72295$$

The *π*-vector is updated using equations 3.6, 3.5 and 3.4 as follows:

$$\delta_{sa} = \min\left((0.52)(0.63) + (0.43)(0.37), \frac{0.5}{2-1}\right) = 0.4867$$

Since the action taken was the best possible action, Δ_{sa} is updated as

$$\Delta_{sa} = (0.52)(0.29) + (0.43)(0.71) = 0.4561$$

And the policy vector itself is updated as

$$\pi(H) \leftarrow 0.52 + \left[\frac{0.4561}{2}(0.63) \right] = 0.6637$$
$$\pi(L) \leftarrow 0.43 + \left[\frac{0.4561}{2}(0.37) \right] = 0.5144$$

After applying a Boltzmann distribution to ensure equation to ensure equation 3.3 holds true, the policy vector values are

$$\pi(H) = 0.5692$$
$$\pi(L) = 0.4411$$

This same procedure applies regardless of the size of the policy and q vectors or the number of possible actions. In the case of WoLF or PD-WoLF, the procedure is the same with some additional calculations as described in sections 3.2 and 3.3.

3.2 Combining WoLF-PHC and Fuzzy State Aggregation

Unlike the standard PHC algorithm, the WoLF-PHC and PDWoLF-PHC both utilize a dynamic learning rate to increase the speed of convergence over the standard PHC. The heart of this method is the quick learning when losing, and cautious learning when winning. Ideally the learner will adapt quickly when it is doing worse than expected. When it is doing better than expected it should be more cautious since the opponent(s) may change their strategy. The key to this is finding a way to determine if the agent is winning or losing [6]. Comparing the agent's current policy with an estimated average policy is one way of doing this.

In this application the WoLF-PHC algorithm uses an additional vector to estimate the average policy value. The average policy vector is initialized like the π-vector:

$$\bar{\pi}(k) \leftarrow \frac{\#of\ \ fuzzy\ labels}{|AK|} \tag{3.8}$$

This vector is updated by

$$\forall_{k \in K} \quad \bar{\pi}(k) \leftarrow \bar{\pi}(k) + \left(\frac{\pi(k) - \bar{\pi}(k)}{C} \right) \tag{3.9}$$

where C is a counting function used to track how many times the elements representing a state have been updated. In this implementation all state elements for the selected action are updated simultaneously, so C is simply the number of times the algorithm has looped.

The delta selection for determining the learning rate in WoLF is then calculated as follows:

$$\delta = \begin{cases} \delta_w & if \sum_{k=1}^{K}\sum_{a=1}^{A} \pi(k)q(k)\mu_k(s,a) > \sum_{k=1}^{K}\sum_{a=1}^{A} \overline{\pi}(k)q(k)\mu_k(s,a) \\ \delta_l & otherwise \end{cases} \tag{3.10}$$

where $\delta_l > \delta_w$ and both fall within the range (0,1]. This value for δ is used to calculate Δ_{sa} as described in equation (3.4) and is derived from the δ calculation of WoLF in equation (3.10).

3.3 Applying PDWoLF to the FSA

The PDWoLF-PHC also uses additional values to change the learning rate. Rather than using an average policy table (or vector), PDWoLF uses the change in policy from the previous time step $\Delta(s,a)$ with the change in policy from the current time step Δ_{sa}. These are initialized as

$$\Delta(s,a) \leftarrow 0 \text{ and } \Delta^2(s,a) \leftarrow 0 \tag{3.11}$$

Where Δ and Δ^2 are changing rates within the policy and are estimates of the slopes of the decision space. These are respectively updated for the selected action as

$$\begin{aligned} \Delta^2(s,a) &\leftarrow \left(\sum_{k=1}^{K} \frac{\Delta_{sa}}{K} \mu_k(s,a) \right) - \Delta(s,a) \\ \Delta(s,a) &\leftarrow \left(\sum_{k=1}^{K} \frac{\Delta_{sa}}{K} \mu_k(s,a) \right). \end{aligned} \tag{3.12}$$

The delta selection then becomes

$$\delta = \begin{cases} \delta_w & if\ \Delta(s,a)\,\Delta^2(s,a) < 0 \\ \delta_l & otherwise \end{cases}. \tag{3.13}$$

3.4 The Tile World Experimental Domain

The original Tile World introduced by Pollack and Ringuette [10] is a chessboard-like grid on which there are agents, tiles, obstacles, and holes. The agent can move up, down, left, or right, one cell at a time. A tile is a unit square which slides when pushed by the agent, an entire row of tiles can be pushed by the agent. An obstacle is a grid cell or group of cells which are immovable. A hole is a cell or group of grid cells, each of which can be “filled in" by a tile when the tile is moved on top of the hole cell; the tile disappear and the size of the hole cell diminishes If a hole becomes completely filled, the agent gets points for filling it, and the hole disappears. The agent knows ahead of time how valuable the hole is; its overall goal is to get as many points as possible by filling in holes.

A Tileworld simulation takes place dynamically: it begins in a state which is randomly generated by the simulator according to a set of parameters, and changes continually over time. Objects (holes, tiles, and obstacles) appear and disappear at rates determined by parameters set by the experimenter, while at the same time the agent moves around and pushes tiles into holes.

The version of Tileworld implemented in this research is designed as a test-bed for machine learning methods that are then transitioned to the very stochastic world of robot soccer. The intent is to approximate the conditions under which a soccer player decides what to do with the ball when he has possession of it; to whom should he pass it? Should he dribble it, and if so in which direction? Should he shoot for a goal? These are the questions a human soccer player must answer instantaneously as must the soccer playing agent in RoboCup. The simplified Tileworld domain for these experiments is based on the modified Tileworld domain used by Berenji and Vengerov [4, 5]. The simplified Tileworld consists of agents, reward spikes, and deformations. The agent must select which reward to pursue while avoiding the penalty deformations that move about the board. Like the original Tileworld, the reward spikes and penalty deformations have random, but finite life spans, but the agent gathers the reward by reaching the hole rather than pushing a tile into it. The agent also has the option of moving in one of eight directions as opposed to four in the original Tileworld. Figure 3.1 provides a conceptual picture of the modified Tileworld domain.

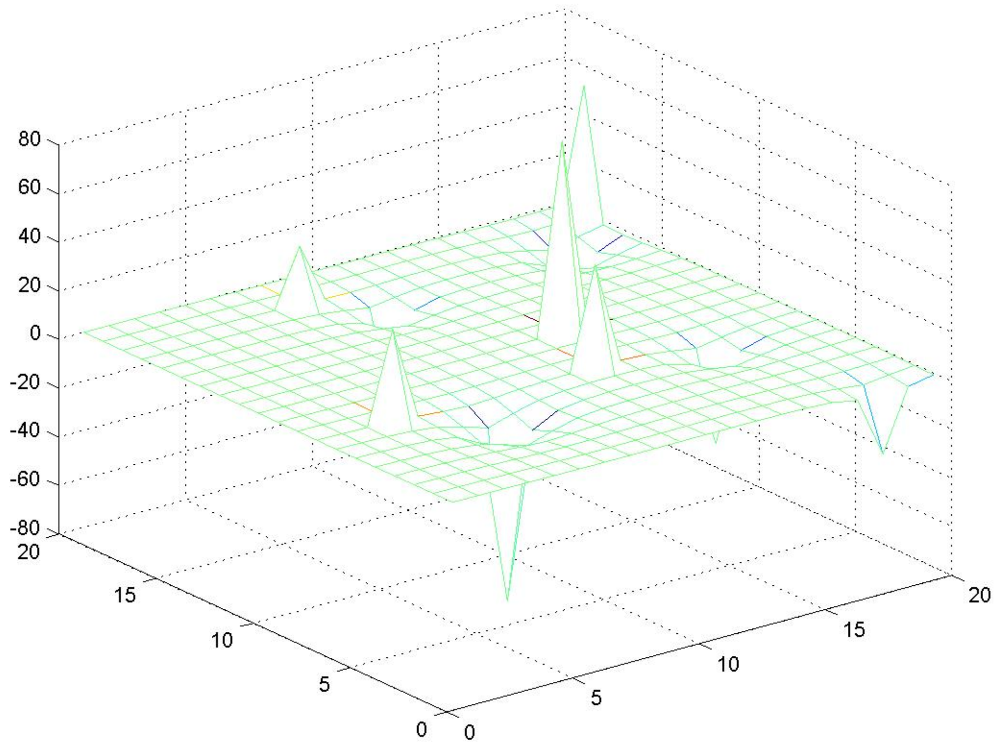

Figure 3.2: Conceptual Image of the Simplified Tileworld Domain

The simplified Tileworld implementation consists of a 20 x 20 grid world containing five reward opportunities and five deformations. The reward opportunities each have random value of 20 to 100 points and a random life span of 5 to 15 time steps. Anytime the agent reaches a reward or the reward expires, it disappears from the domain and another one is generated elsewhere on the board. Agent can move 1 step each time step.

Each deformation has a random penalty value of -5 to -20 points and, unlike the rewards, these deformations occasionally drift. At each time step each deformation has a 10% chance of moving one square in a random direction. Each deformation is also the center of a potential field that radiates out based on the following equation:

$$P = \frac{v}{(d+1)^2}. \tag{3.14}$$

Where v is the value of the deformation and d is the distance from the deformation. The cost of each square in the domain is the sum of the effects of each potential field at that point. The environment is fully observable in that the agent knows the location and value of each reward at all times as well as the location and effect of the deformations.

Each state in the domain is represented by four state variables:

1. Distance to the reward
2. Value of the reward
3. Estimated life expectancy of the reward
4. Roughness of path to the reward.

The distance to the reward is calculated simply using the Pythagorean Theorem. The value of each reward is randomly determined at the time it is generated. The estimated life expectancy of a reward (L) is calculated by

$$L=m-t(r) \tag{3.15}$$

where m is the mean life span (m=10 in this example) and $t(r)$ is the number of time steps that reward r has existed. The roughness of the path to the reward is calculated by constructing a rectangle with the agent and the reward at opposite corners. The roughness is the average cost of all the squares in that rectangle.

At each time step the agent must decide which of the reward opportunities to pursue. This decision is based on the state variables described above.

Once the decision is made, the agent moves one square towards that opportunity, the policy is updated and the process repeats.

Because the agent can move in any of eight directions (orthogonally or diagonally) there are always three contiguous squares that the agent can choose from to move towards the selected reward. At each time step the agent simply uses the square with the lowest cost.

With each step, the agent garners a negative reward equivalent to the cost of the square it moves to. The agent only receives a positive reward upon reaching a reward opportunity before it expires.

The value of each of the state variables is described by the three fuzzy labels (Small, Medium and Large). The shapes and specific values of these fuzzy labels are shown in figure 3.3.

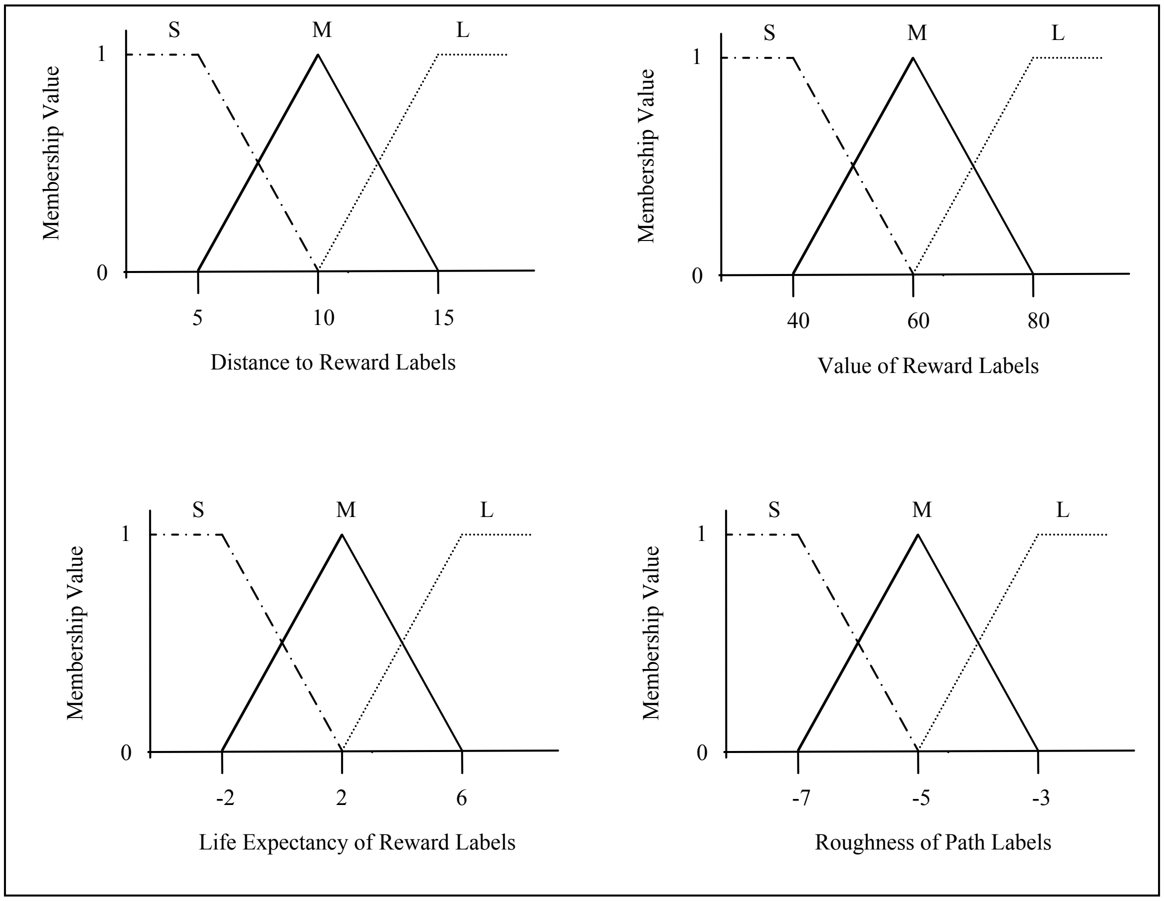

Figure 3.3: Fuzzy Labels Used by the Agent

For each of the state variables, the fuzzy labels are assigned so that they evenly divide the range of possible values for the variable. The degree to which the agent is in one of the fuzzy states is the mean of the degrees to which all the state variables belong to the corresponding labels in the fuzzy state. In this experiment there are four state variables and three fuzzy labels resulting in 81 (3^4) total fuzzy states. For comparison purposes, without fuzzy state aggregation, this same domain would have 210 possible distance values, 80 possible reward values, 15 different life expectancy values and at least 1000 different roughness values resulting in 2.52×10^9 possible states. By limiting the state variable values to only integer values (which is not the case in our experiment) this number could be reduced to just over 320,000 states.

At the beginning of each experiment the Q-values are all set to 20. This number is selected because it is comparable to the maximum Q-values found at the end of the experiment and starting with this value results in some natural exploration in the earliest stages of learning. Because the entire *q(k)* vector and *π(k)* vector are updated at each time step, learning occurs very quickly and no dedicated exploration is required.

3.5 Proposed Design Plan for a RoboCup Team Architecture

The RoboCup domain is a simulated soccer game played by two teams of 11 players each. To the extent possible, all of the rules and strategies of a real soccer game are applicable to RoboCup games. Figure 3.4 shows a screenshot of a RoboCup soccer game in progress.

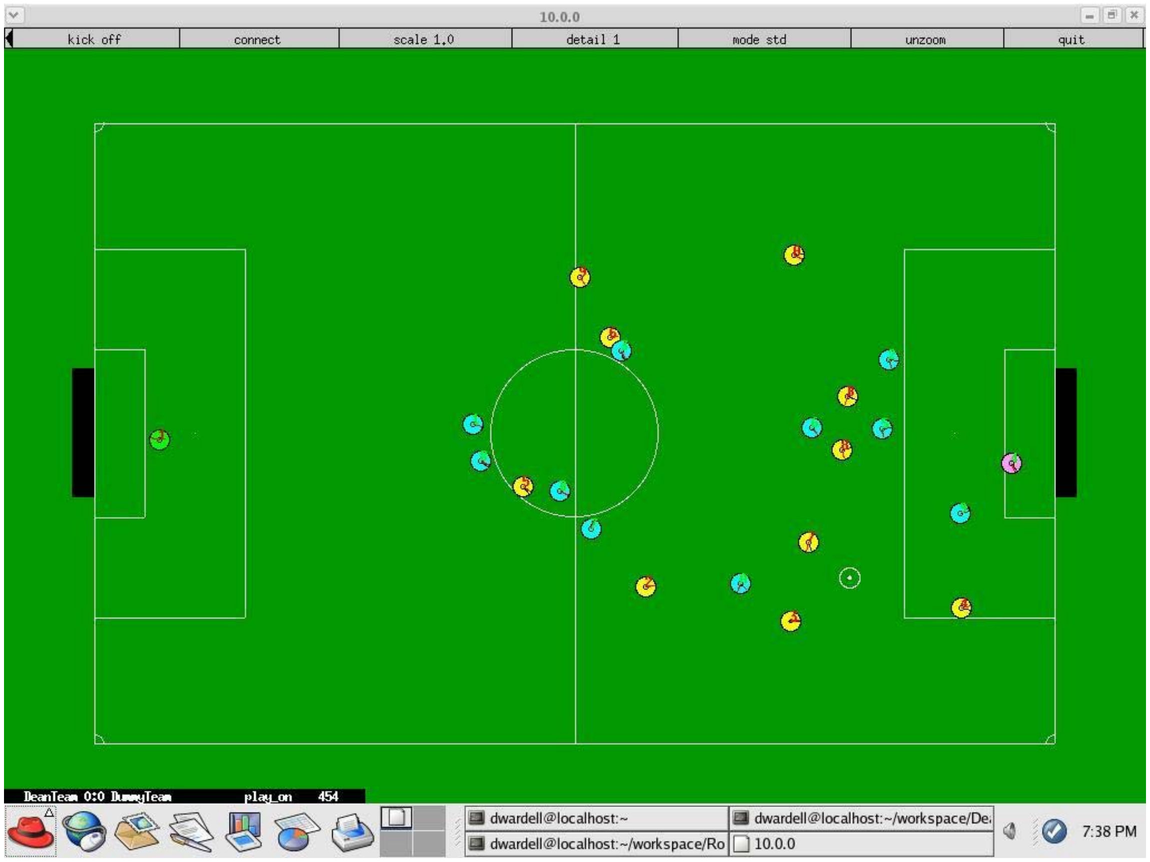

Figure 3.4: Screenshot of RoboCup Game in Play.

3.5.1 Multi-Agent Architecture

The overarching design for the multi-agent system is a three-level design, composed of a planner, a deliberator and an execution layer with no outside inputs to the control program during play (no Coach).

At the highest level, the planner uses current state information to project out the best course of action to achieve the goal (such as scoring a goal). Since this is such a dynamic environment, and since the program must be fully distributed across the agents, the deliberative creation of plays must be efficient and comparatively simple, but more than just "find the next best step". This may also include the use of pre-determined plays and locker room agreements as on-field communications is minimal.

In any case it is necessary for the agents to assume roles in the different formations and plays. This is accomplished using a fuzzy logic based function that each agent calculates for itself and then coordinate with other agents. As a notional example, in order for three agents to coordinate attacking the ball (without running into each other) each calculates the value of their own position using their distance and orientation to the ball and goal as inputs to the fuzzy logic unit. Depending on which is most efficient either each agent makes the same calculation for the other two, or they simply communicate their results with each other. In either case the agent with the highest position value leads the attack on the ball and the others assume supporting roles in accordance with the play.

As this is a decentralized control design, each individual agent (with exception of the goalie) independently runs the same program as described below.

3.5.2 Single Agent Architecture

This section describes the code actually implemented as part of the overall multi-agent architecture described in section 3.5.1.

The interactions with the simulator/server and the fundamental agent actions (e.g. kick, dribble, move, collision avoidance) are based on the source code provided by Peter Stone of CMU. It is a portion of the program used by CMU's 2002 RoboCup team and is freely distributed for research/educational purposes as United2002-source.

The principle philosophy behind the single-agent architecture is to develop simple behaviors which when combined results in the emergence of more complex behaviors. As the more complex behaviors are better refined, attention is then focused on executing pre-planned plays as described in Section X. The architecture is modular in design, as shown in figure 3.5, and consists of:

- A deliberator which, at this point, simply selects between the possible behaviors.
- A behavior module which consists of a Fuzzy Logic Controller (FLC) for controlling the default positioning behavior and calls the fuzzy PHC programs that each calculate its best estimated reward based on the current state.
- A reinforcement learning module which updates the Q-value and policy vectors for the individual agent during the course of play.
- An offline Weighted Strategy Sharing routine which runs after each game updates the Q-value and policy vectors for each agent based on the values provided from all other agents. The data is saved and made available for the beginning of the next game
- The fundamental actions module constituted by the CMU software described previously in this section.

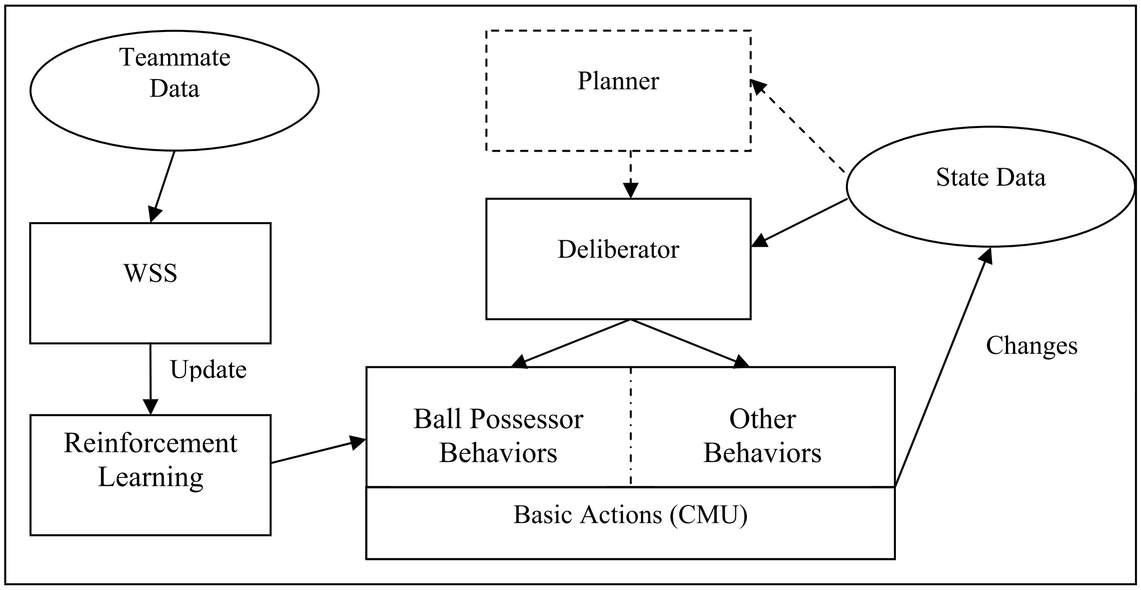

Figure 3.5: Architecture for RoboCup Agents

The RoboCup simulator allows each agent/team member to submit a command at each 0.1 sec time increment. This means in any given second of play, each team could be submitting up to 110 commands to the simulator. The challenge then is not only to select the best action for each agent, but to do so as quickly as possible. As part of the “realism” of the game, each agent has a limited range of vision and a player may not be able to identify the location of the ball or a particular player if the player’s view is obstructed or the object in question is too far away. This is useful because decisions are based only on those pieces of information which the agent has ready access. For example, when deciding which teammate to pass the ball to, the agent should not even consider those teammates outside his visual range or to whom the agent does not have an unobstructed view. This allows the agent to limit the number of calculations made at each decision point.

At each iteration of the program the agent finds or estimates the position of the ball in Cartesian coordinates on the field. Based on the agent's relative position to the ball and which team is in possession of the ball, the agent either acts on the ball or moves to a position determined by the positioning fuzzy logic controller (FLC). For this positioning, the FLC uses as inputs the X position of the ball on the field and the agent's X position on the field as shown in figure 3.6.

Ball Position

Agent Position

	Near our Goal	**Mid-Field**	**Near their Goal**
Near our Goal	More Defensive	Less Defensive	Neutral Posture
Mid-Field	Less Defensive	Neutral Posture	Less Offensive
Near their Goal	Neutral Posture	Less Offensive	More Offensive

Figure 3.6: Position FLC Output

Each agent has a zone of responsibility on the field. The position determined by the FLC is calculated and executed in reference to the center point of the agent's zone of responsibility. As an example, one agent has responsibility for the upper part of the field closest to its own goal. When the ball is on the other end of the field and there are no opponents in this zone, the position FLC directs the agent to a point on the forward part of its zone of responsibility. As the ball moves towards the agent's side of the field, the agent falls back towards the center of its zone. If an opponent moves into the agent's zone the agent moves to cover that opponent. As the ball approaches the agent's goal the agent moves back to a more defensive position closer to its own goal. By using a Fuzzy Logic Controller in this application, the agent's response to the ball and opponent players is smoother and more easily adjusted.

If the agent determines it is in a position to go after the ball (closest team player to the ball) the positioning decision described previously is set aside to pursue the ball. Once the ball is close enough for the agent to kick it, the agent must make other decisions; shoot for a goal, pass the ball or dribble it. This decision is based on which of these actions has the highest expected reward for the current state of the game. The expected reward for each action is calculated in the same manner as in the proof-of-concept Tile World experiment discussed earlier.

Passing the Ball - To calculate the expected reward for passing the ball, the agent with the ball considers each teammate (except the goalie) that is visible and to which the agent has a clear line of sight. For each teammate, the state variables the agent uses are

- How many opponents are around the teammate
- How far away is the teammate
- How much closer to, or further from, the opponent's goal is that teammate

The number of Opponents around the teammate is calculated by projecting a 45^o cone from the agent to the teammate and counting the number of opponent players located inside that cone as illustrated in figure 3.7.

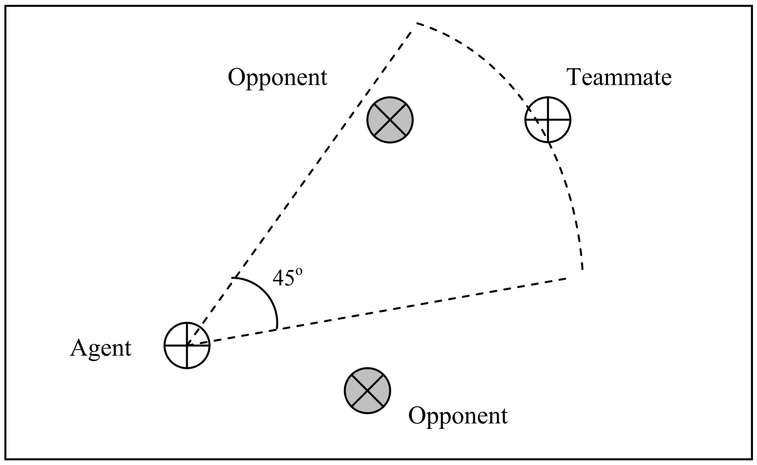

Figure 3.7: Cone used to count opponents near a teammate

The distance to the teammate is a simple calculation using the Pythagorean Theorem and the relative closeness to the opponent goal is the difference between the teammate's X position and the agent's X position.

As in the tile world experiment, fuzzy state aggregation constrains the number of states in the domain. There are three fuzzy labels (sets) for each state variable resulting in 3^3 or 27 total fuzzy states used for passing the ball. Since the RoboCup domain is continuous, the actual number of states is incalculable. The fuzzy labels used for each state variable are shown in figure 3.8.

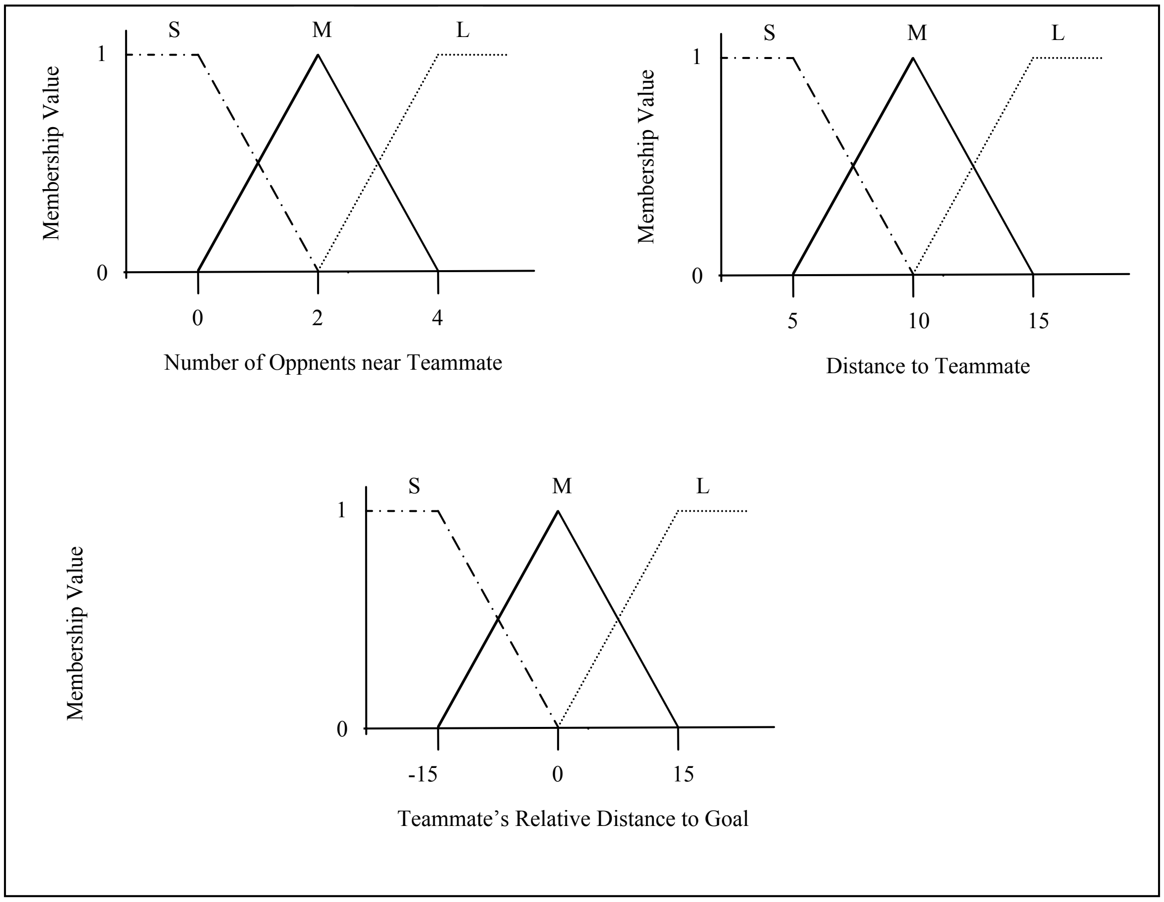

Figure 3.8: Fuzzy Labels used for Passing

There is a small reward for completing a forward pass (up to 20 points); while the cost of completing a backwards pass is 5 to -15 points. The penalty for having a pass intercepted is -30 points. This reward structure encourages forward passes, mildly discourages backward passes and strongly discourages passing to a teammate in close proximity to opponents, as they likely lead to the ball being intercepted.

The agent first calculates the expected value for passing the ball to each teammate based on the reward values stored in the q-vector and probability values stored in the current policy vector (see equation

3.1). The teammate with the highest expected reward, based on that calculation is selected to receive the pass.

Dribble the Ball - In calculating the expected reward of dribbling the ball, the agent considers dribbling in each of eight different directions. The eight directions are 45^o apart in a complete circle around the agent, beginning with the -Y direction. For each possible direction the agent uses two state variables

- Number of opponents in that direction
- Degrees away from a direct path to the goal

The number of opponents in the direction is calculated by projecting a 45^o cone to a point 10 meters away in that direction and counting the opponents within that cone. The second state variable value is simply the difference between the direction in question and the angle to the goal as shown in figure 3.9.

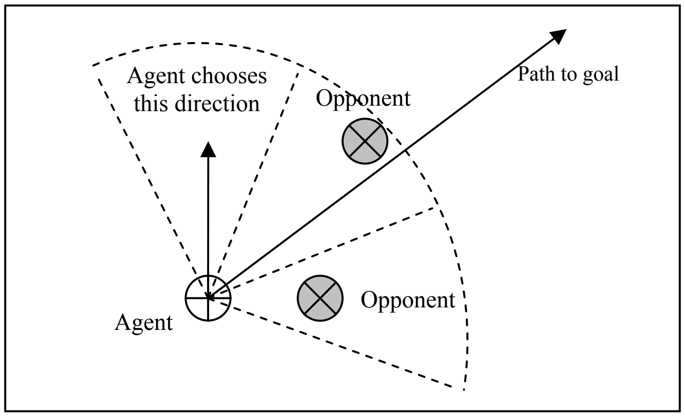

Figure 3.9: Agent decides direction to dribble

Using fuzzy state aggregation with three fuzzy labels and these two state variables the domain consists of nine (3^2) fuzzy states for dribbling the ball.

The learning feedback for the decision process consists of the agent receiving a reward proportional to the progress towards the goal resulting from that decision. Dribbling directly towards the goal is worth 4.5 points down to 0 for dribbling at an angle perpendicular to the goal. Dribbling away from the goal results in a proportionally negative cost (0 to -4.5 points) and losing possession of the ball costs the agent 30 points. This reinforcement method encourages dribbling towards the goal, discourages dribbling away from it and strongly discourages dribbling towards nearby opponents. The fuzzy labels used for dribbling are shown in figure 3.10.

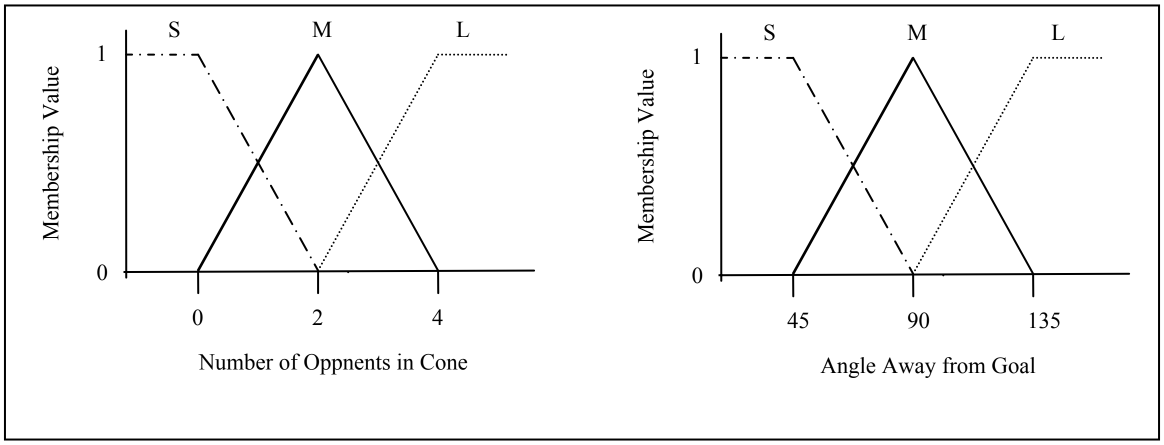

Figure 3.10: Fuzzy Labels used for Dribbling

The agent calculates the expected value for dribbling the ball in each direction based on the reward values stored in the q-vector and probability values stored in the current policy vector using equation 3.1. The direction with the highest expected reward, based on that calculation is the direction the agent selects.

Shots on Goal - To calculate the expected reward of shooting the ball, the agent considers shooting at each of seven different points in the goal. The seven points are all along the back of goal, starting at dead center (55, 0) and working out 2 meters at a time. For each possible target point the agent uses three state variables

- Number of opponents along the path to the target point
- Distance between the target point and the opposing goalie
- Distance to that point

The number of opponents near the path to the target point is calculated by projecting a 15^{o} cone from the agent to the target point and counting the number of opponent players located inside that cone, as shown in figure 3.11. The distance between the goalie and the target point is simply the difference in Y value of the goalie's location compared to the Y value of the target point. The distance to the target point is a simple calculation using the Pythagorean Theorem.

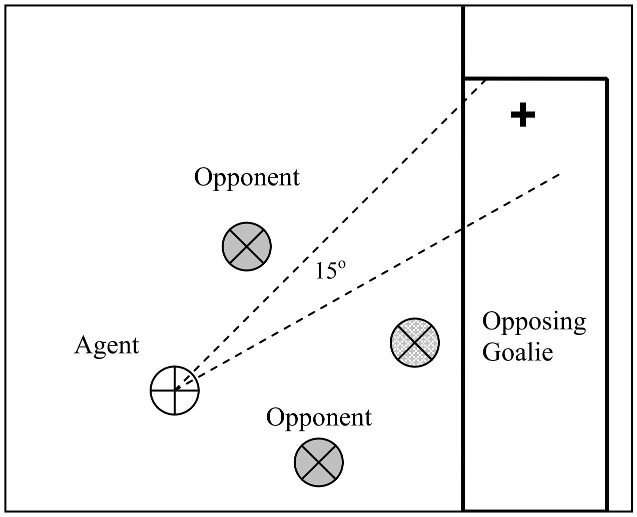

Figure 3.11: Agent Considers Point (55, -6) for Goal Shot

Using fuzzy state aggregation with three fuzzy labels and these two state variables the domain is constrained to 27 (3^3) fuzzy states for shots on goal. The fuzzy labels used for shooting are shown in figure 3.12.

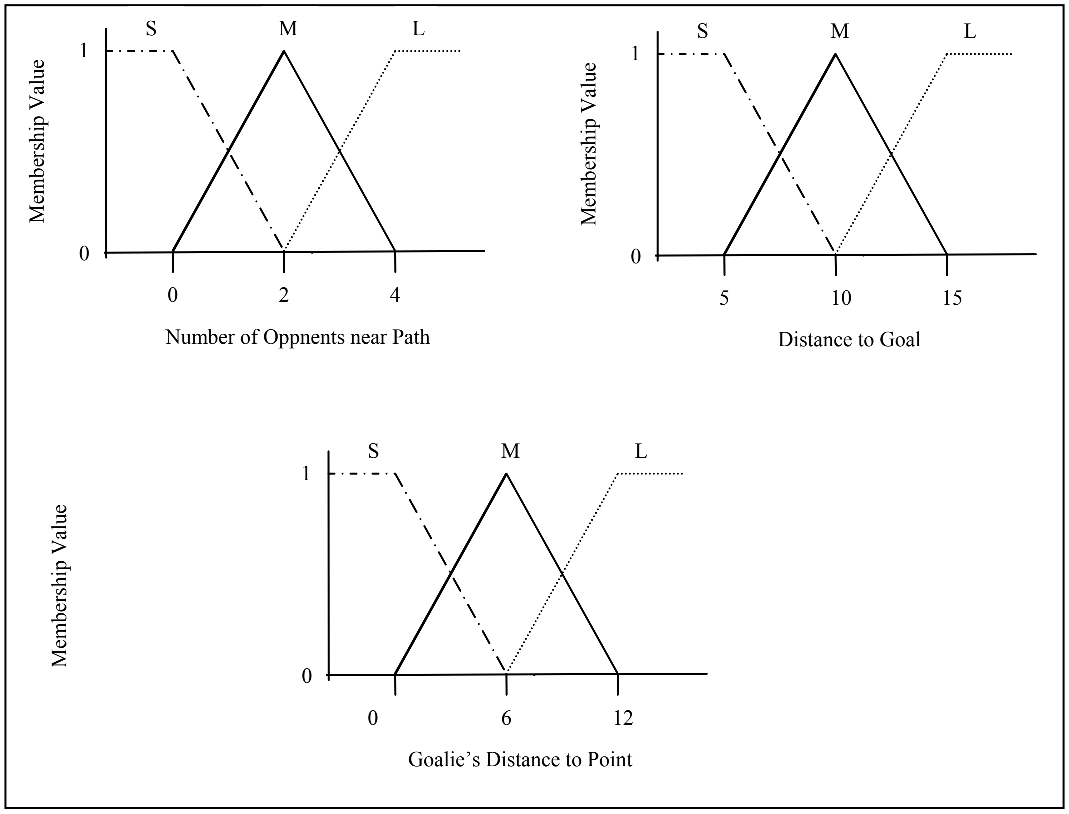

Figure 3.12: Fuzzy Labels used for Shooting

To reinforce the decision, the agent receives a high reward for scoring a goal (75 points) and a penalty of -30 points if the opposing team takes possession of the ball. This reinforcement method encourages shooting towards the points in the goal furthest away from the goalie and strongly discourages shooting close to nearby opponents. It also encourages shooting from closer in than further out.

The agent calculates the expected value for shooting the ball to each of the possible target point based on the reward values stored in the q-vector and probability values stored in the current policy vector (equation 3.1). The target point with the highest expected reward, based on that calculation is the point at which the agent shoots the ball.

Every time the agent has possession of the ball and must decide which action to take, the agent compares the highest expected reward for each of the three possible actions and selects the one with the highest overall expected reward. The decision and control flow is shown in Figure 3.13.

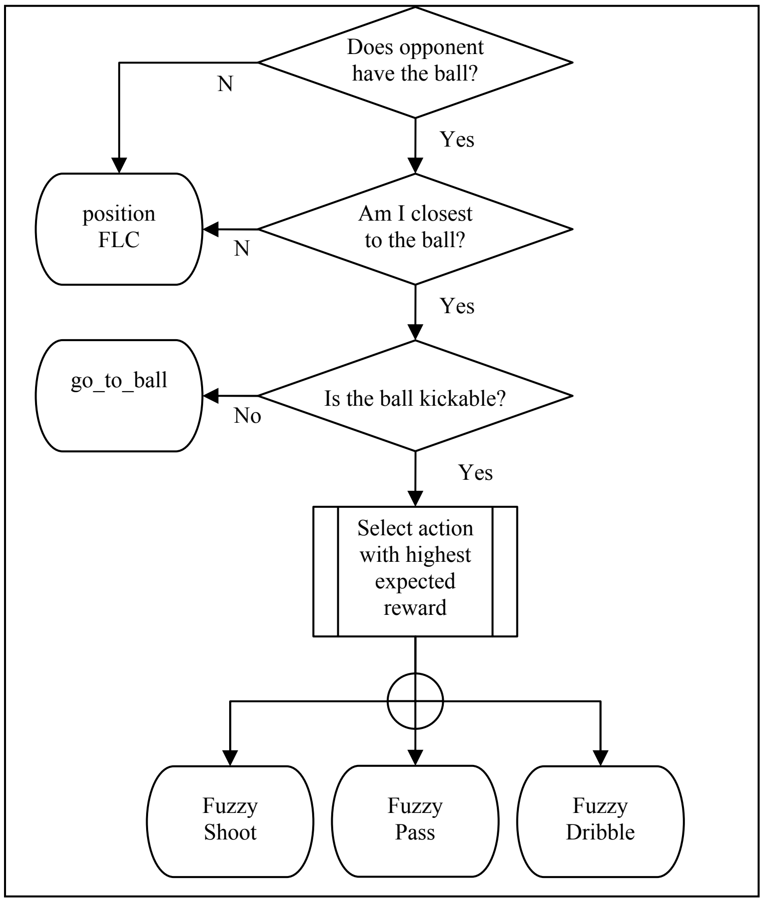
Does opponent have the ball?
N
Yes
position FLC
Am I closest to the ball?
N
Yes
go_to_ball
Is the ball kickable?
No
Yes
Select action with highest expected reward
Fuzzy Shoot
Fuzzy Pass
Fuzzy Dribble

Figure 3.13: Decision and Control Flow for Agent Architecture

3.6 Weighted Strategy Sharing

Because each agent runs this algorithm independently of its teammates, each agent learns a different policy based on its individual experience. Weighted Strategy Sharing (WSS) allows the agents to benefit from their teammates' experiences develop more refined policies. The communication available between agents during game play is limited in range, constrained in its content size and not completely reliable. For all of these reasons, and the fact that RoboCup rules prohibit any inter-agent communication outside the simulator, the WSS occurs off-line at the end of each game.

At the conclusion of the game, each agent stores their respective Q vector and total reward for each of the three activities using reinforcement learning (passing, dribbling and shooting). The WSS algorithm accesses these files and creates a single updated vector using the "Learning From All" [8] weight assignment mechanism in which agent *j* is weighted by agent *i* using the equation:

$$W_{ij} = \frac{e_j}{\sum_{k=1}^{n} e_k} \tag{3.16}$$

Where *n* is the total number of agents and e_k is the amount of the expertness of agent *k*. The individual expertness value for each agent is calculated using the absolute value of the algebraic sum of the reinforcement signals (r_i).

$$e_i = \left|\sum r_i\right| \tag{3.17}$$

The updated Q vector is then calculated as follows:

$$Q_i^{new} \leftarrow \sum_{j=1}^{n} (W_{ij} * Q_j^{old}) \tag{3.18}$$

This new single Q-vector is then used as the initial vector for the next game.

3.7 Summary

This chapter provides the detailed implementations of fuzzy state aggregation and ties that in with Q-learning. The next section expanded the Q-learning to include three different versions of policy hill climbing; standard PHC, WoLF and PD-WoLF. These methods are implemented in a variation of Tileworld which was described in detail. The experiment progressed to a multi-agent implementation in the

RoboCup simulator where weighted strategy sharing is also included. The results of these experiments are described in chapter 4.

IV. Experimental Results

This chapter describes the results of the experiments presented in Chapter 3. Beginning with a proof-of-concept Tileworld experiment, performance is measured by the average learning reinforcement points earned at each time step. The learning performance of a Q-learning agent is compared to that of a policy hill climbing (PHC) agent, both of which are using fuzzy state aggregation. This is followed by a side-by-side comparison of the three different policy hill climbing variants.

The second part of the chapter presents the results of similar experiments conducted in the RoboCup domain. The Q-learning algorithm using FSA is compared to the PHC with FSA algorithm. The teams using these algorithms play against a reactive "Dummy" team and against a 2004 RoboCup tournament quarter-finalist team, TokyoTech 2004. Additionally, the learning performance of the PHC with FSA team against the Dummy team is evaluated both with and without weighted strategy sharing,

4.1 Tileworld Experiment Results

Initial experiments consisted of running multiple games of 200 time-steps each. The q and π-vectors were reinitialized at the beginning of each game and the same number of games was run for each algorithm. In order to obtain useful results, it is necessary to perform a large number of games (1000-2000) and average the results. The stochastic nature of the domain prevents finding meaningful results from a small number of games.

The parameter settings (see equation 2.1) are $\alpha=0.1$, $\gamma=0.3$, and $\delta=0.5$. Experience shows that the ratio of $\gamma/\alpha=3$ works well in most situations. Increasing the value of α only results in a larger magnitude of Q values, but with no corresponding increase in performance. Figure 4.1 shows an evaluation of the learning ability of an on-policy Q-learning algorithm using fuzzy state aggregation compared to the basic off-policy hill climbing algorithm using fuzzy state aggregation.

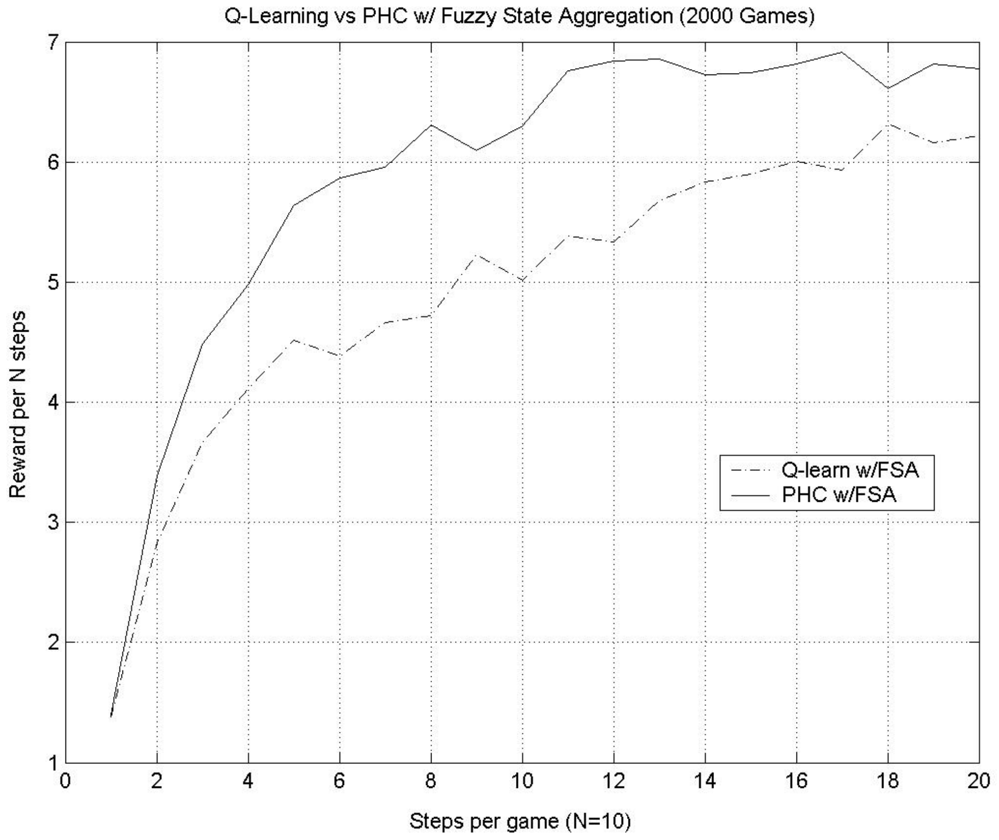

Figure 4.1: Results of PHC with FSA vs. Q-Learning with FSA

The plots in Figure 4.1 show the average number of reinforcement points gathered every 10 time steps in a 200 time step game. Experience shows no significant learning occurs in this domain after 200 steps. These results are averaged over 2000 games. Not surprisingly, the policy hill climbing algorithm performs better (learns a better policy) than the Q-learning algorithm even though both are using the exact same fuzzy state aggregation method.

Figure 4.2 shows a side-by-side comparison of the evaluated learning ability of the PHC, WoLF, and PD-WoLF algorithms. These three algorithms are compared to determine if the variable learning rate in WoLF and PD-WoLF improves performance. For these tests, the run time of each game is shortened to just 200 time steps from the initial testing as there is not a significant amount of learning after this point. The following parameter settings are used for these algorithms: $\alpha=0.1$, $\gamma=0.3$, $\delta=0.5$, $\delta_w=0.2$, and $\delta_l=0.8$. These values were chosen for δ_l and δ_w based on Bowling and Veloso's [6] finding that $\delta_l/\delta_w=4$ is a good ratio for using WoLF in stochastic environments. As in the previous experiment, all three algorithms used the same FSA method.

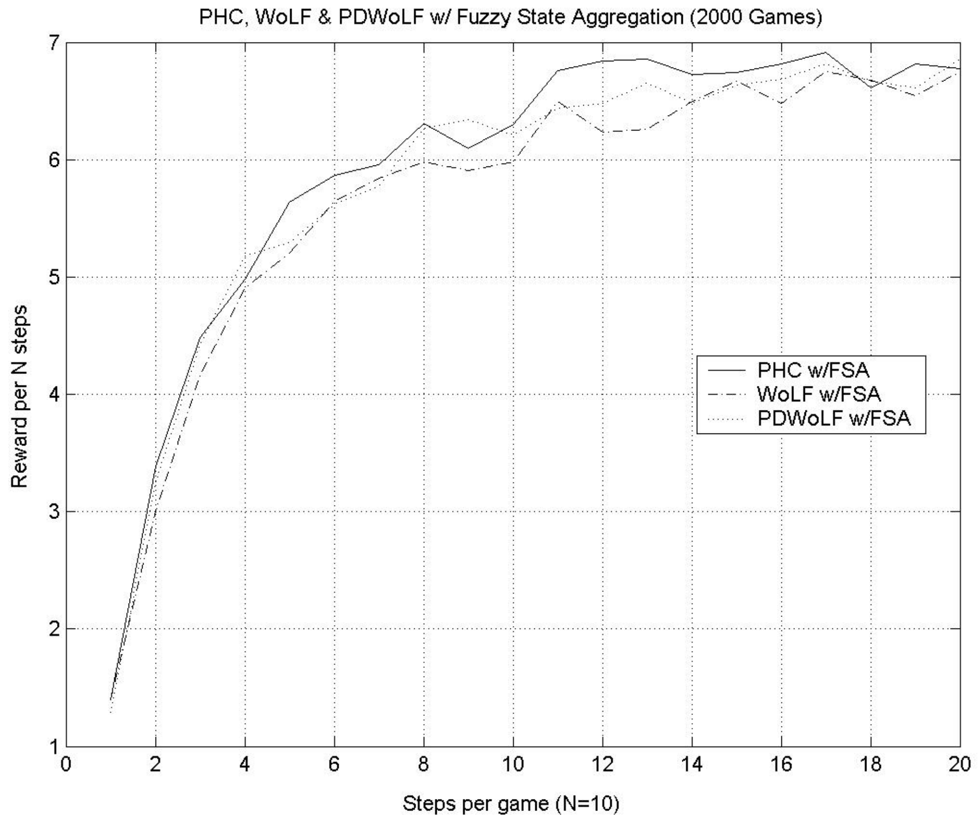

Figure 4.2: Results of PHC, WoLF and PDWoLF

The experiments run indicate that all three PHC algorithms consistently provide similar results, even after varying the values of the parameter settings. The reason for this is that generalizing the states using a fuzzy state approximation vector (which reduces the Q and π table dimensionality) smoothes the landscape of the policy table to an extent that the use of the variable learning rate has little or no effect. The variable learning rate used with WoLF and PD-WoLF requires a more chaotic policy landscape to produce improved results over the standard PHC.

Figure 4.3 shows the relatively smooth surface of the search area generated both by the Q values and the policy values under this implementation of fuzzy state aggregation.

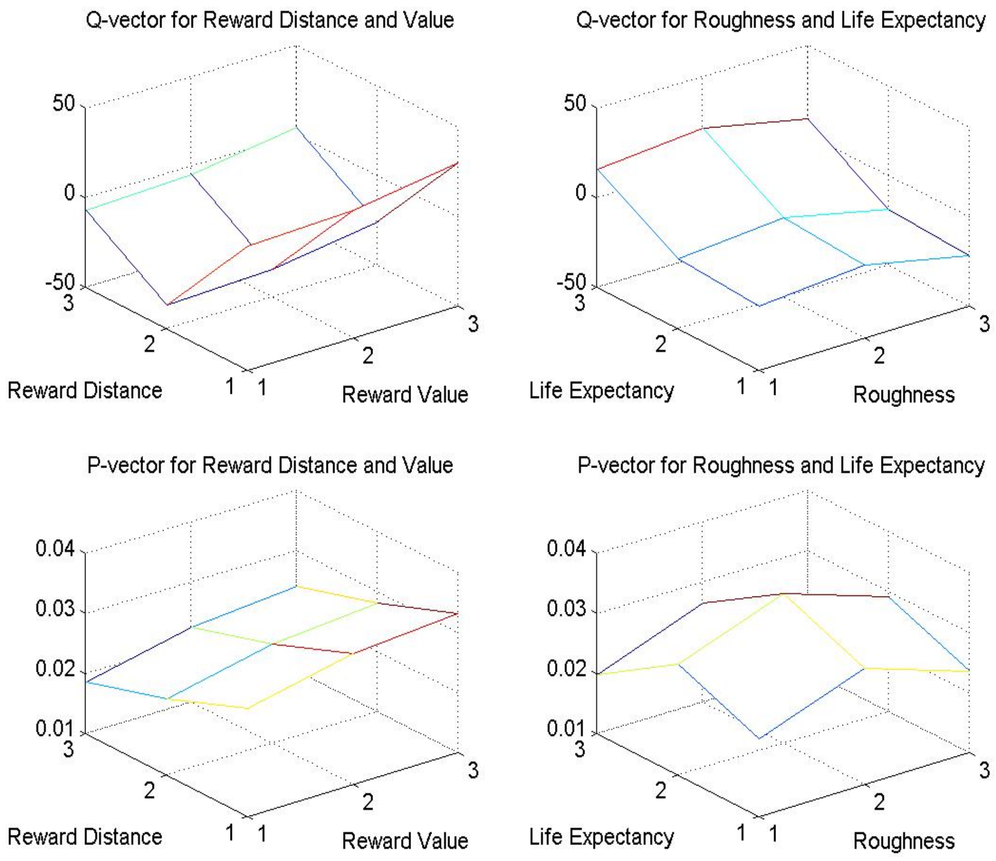

Figure 4.3: Q-vector and Policy Vector Plotted against the three Fuzzy Labels

In the interest of determining if this was indeed the case, the state space of the Tileworld domain is increased by using five fuzzy labels rather than three. This expands the number of fuzzy states to 625 (5^4). A close look at the surface using five fuzzy sets, shown in figure 4.4, indicates that while the search surface is slightly more contoured, the FSA is still smoothing the surface to the point that neither WoLF nor PD-WoLF has an advantage over the standard PHC algorithm.

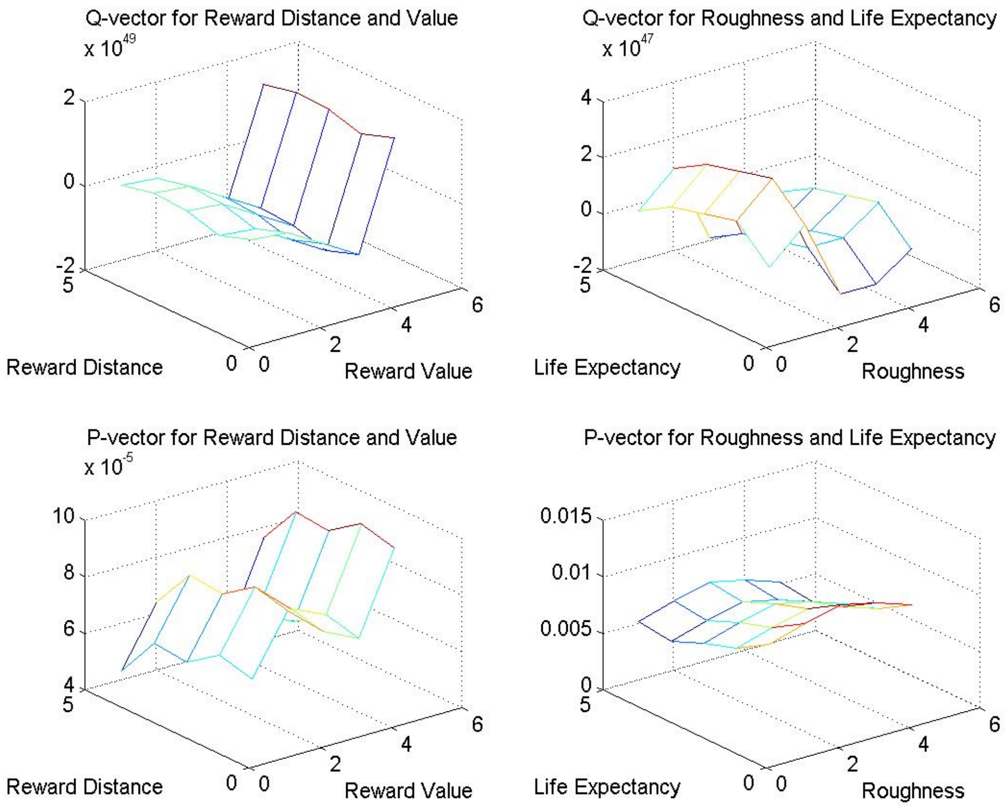

Figure 4.4: Q-vector and Policy Vector Plotted against the Five Fuzzy Labels

Based on the results of this Tileworld experiment, there is sufficient evidence to indicate the PHC with FSA will also perform well in the multi-agent domain of RoboCup. The WoLF and PD-WoLF are not tested in the RoboCup experiment, since the fuzzy state aggregation so smoothes the search surface that the more powerful WoLF and PD-WoLF algorithms are no more effective than the PHC algorithm. The problem would likely be exacerbated by the fact that there are fewer fuzzy states used in each of the learned activities in RoboCup than are used in Tileworld.

4.2 RoboCup Experiment Results

To obtain a quick snapshot of how well the algorithm works in the RoboCup soccer domain, the team using FSA and a PHC play against a “Dummy” team. The Dummy team is simply an earlier version of the experimental team which uses conventional if–else statements in selecting the action taken.

From the perspective of the learning agent, opponent players are treated as part of the environment. The Dummy team generally performs poorly, but provides a good stochastic environment in which to measure the performance of the experimental team. As in the Tileworld experiments, the parameter settings are; $\alpha=0.1$, $\gamma=0.3$, $\delta=0.5$. Figure 4.4 shows the evaluation of the learning ability of the experimental team using just Q-learning with FSA, and using PHC with FSA against the Dummy team.

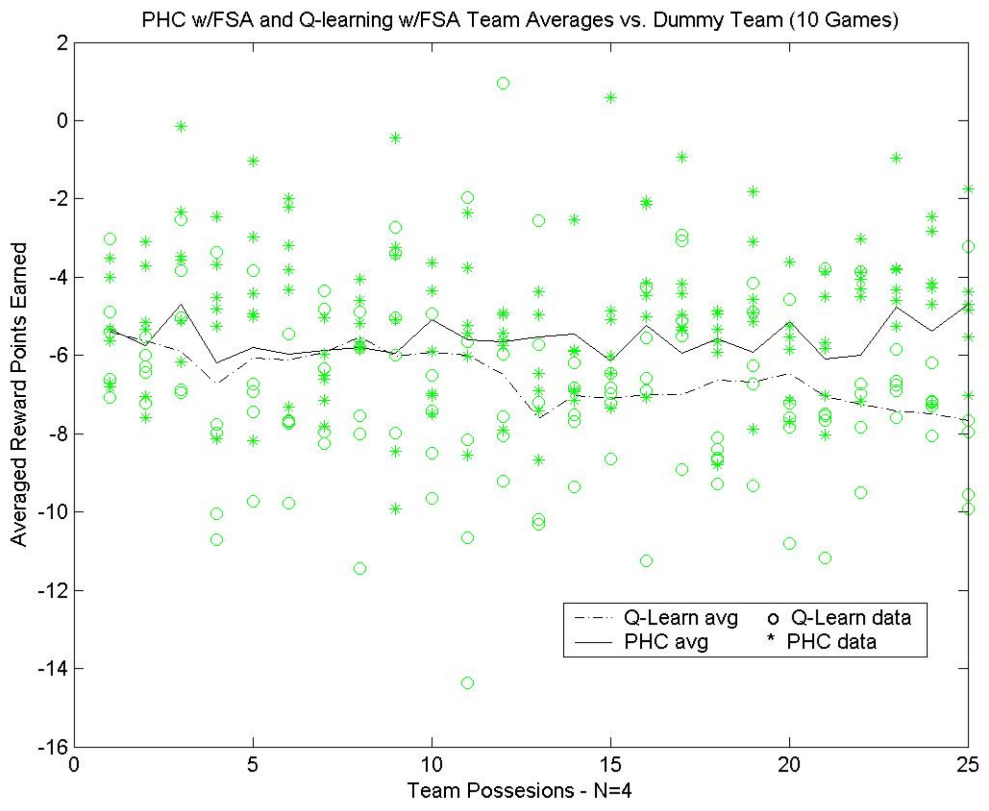

Figure 4.5: Q-learning with FSA and PHC with FSA Performance vs. the Dummy Team

As in the Tileworld experiment, these results are the average reinforcement points collected by the team each time the agents had control of the ball. Each curve is a collective average over five runs of 10 games each. At the beginning of each run of 10 games the Q-vector and policy vector were reset to their initialization values. In between each game within the run, each agent simply keeps their own q-vector from the end of the previous game. The average lines plotted through the data points are based on the average value of the team's reward points at that time. To ease visual comparison between the lines, they were smoothed using a bi-Laplace curve smoothing algorithm (7 iterations).

Consistent with results from the Tileworld experiment, the PHC algorithm demonstrates a better learning ability over the Q-learning algorithm, despite the fact that both algorithms use the exact same FSA method.

For comparison purposes the experimental team also plays against a very accomplished team, TokyoTech.

The TokyoTech team plays well and provides a more challenging environment for the experimental team to learn in. Figure 4.5 shows the reinforcement learning points garnered by the PHC team when playing against the TokyoTech team as compared to playing against the Dummy team.

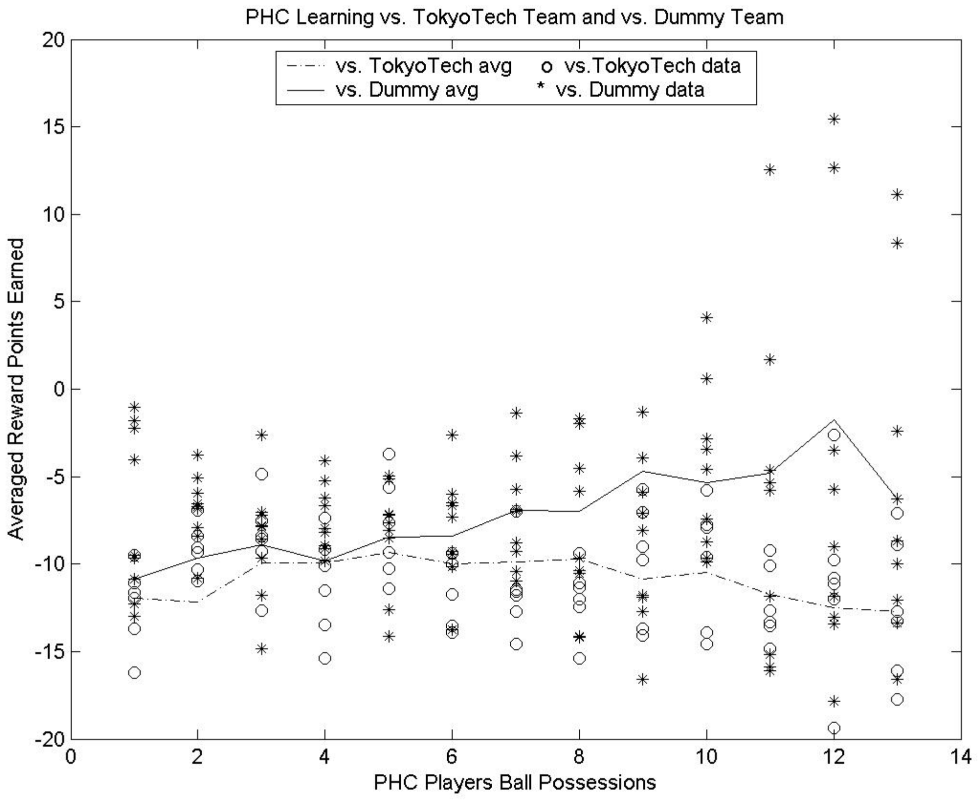

Figure 4.6: PHC with FSA Team Learning vs. TokyoTech Team and vs. Dummy Team

It is not surprising that the PHC team accrues fewer reward points during a game when facing a proficient team than it does against the Dummy team. Against a team that plays well, the agents have the ball less often and frequently find the higher reward options are unavailable to them (dribbling or passing backwards may be the only way to keep the ball). Additionally, the TokyoTech goalie is so good that the

PHC team never scores a goal against them. This means the positive reinforcement for scoring a goal never factors into the reinforcement for learning how to shoot at the goal.

It should be noted that the goalie for Dummy team, Q-learning team, and PHC team are all identical. It runs separate code from the learning agents and simply responds reactively to the presence of the ball. The goalie for these experimental teams moves very slowly compared to the TokyoTech goalie and frequently appears to ignore the ball when it comes close. As a result, game scores between the TokyoTech team and the PHC team are usually in the area of 10 to 0 or 12 to 0. Against the Dummy team TokyoTech routinely scores 20 unanswered goals in a 10 minute game. Adding improvements to the goalie would definitely result in additional goals saved, as currently 80% of the shots taken on the goal score a point.

4.2.1 Dissimilar Team Training

The question of learning against one team then playing against another deserves investigation. In this section there is an investigation of the learning performance of the experimental team (PHC w/FSA) playing against the Dummy team after learning against the TokyoTech team. Those results are compared to the results of the experimental team learning and playing against the Dummy team only.

That section is followed by an experiment with how the PHC w/FSA team learns against TokyoTech after learning against the Dummy team. These results are also compared to the results of the experimental team learning and playing against the Dummy team only.

For these experiments the data was gathered by averaging five runs of four games each. In the dissimilar team training cases, the first two games of the four-game runs were played against one team (e.g. the Dummy team), while the last two games of each four-game run were played against the other team (e.g. TokyoTech).

The plots in Figure 4.7 show the evaluated learning ability of the PHC w/FSA team learning against TokyoTech and playing against the Dummy team.

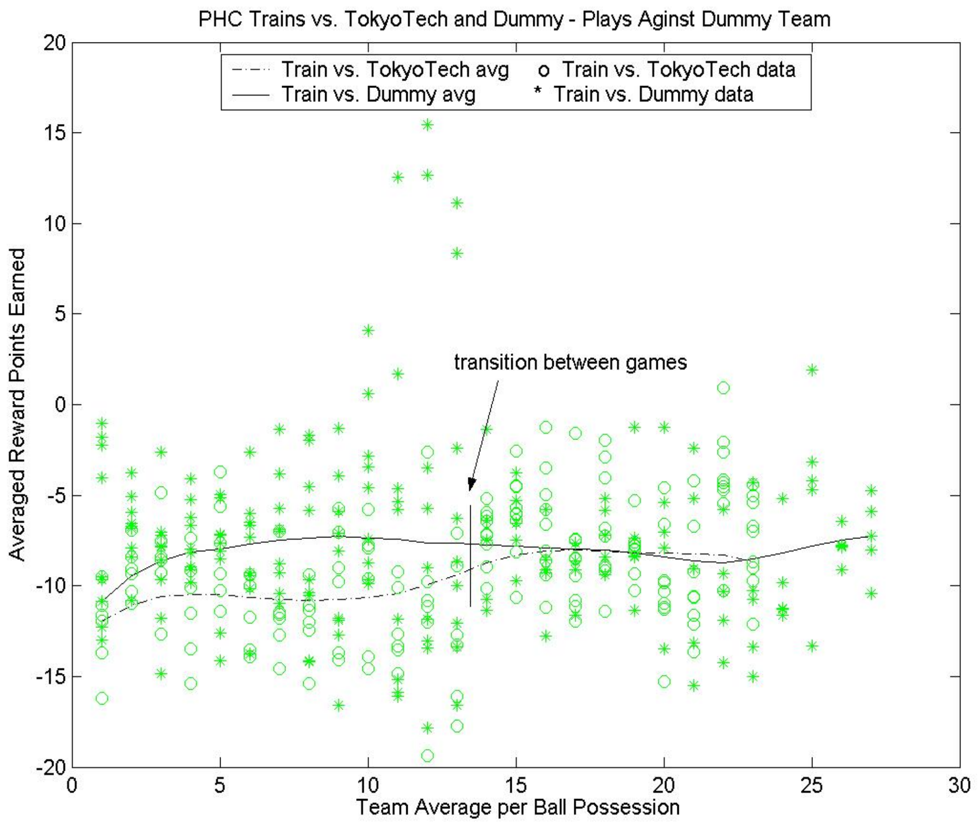

Figure 4.7: PHC with PHC Team Learns vs. TokyoTech and Dummy Team – Plays vs. Dummy Team.

Despite learning more slowly against the better TokyoTech team, the experimental team still performs well against the Dummy team. Any impact of learning against the harder team is compensated for early into first game with the Dummy team.

Figure 4.8 shows the evaluation of the experimental team's learning against the Dummy team then playing against the better TokyoTech team. This is compared to learning and playing against the Dummy team only.

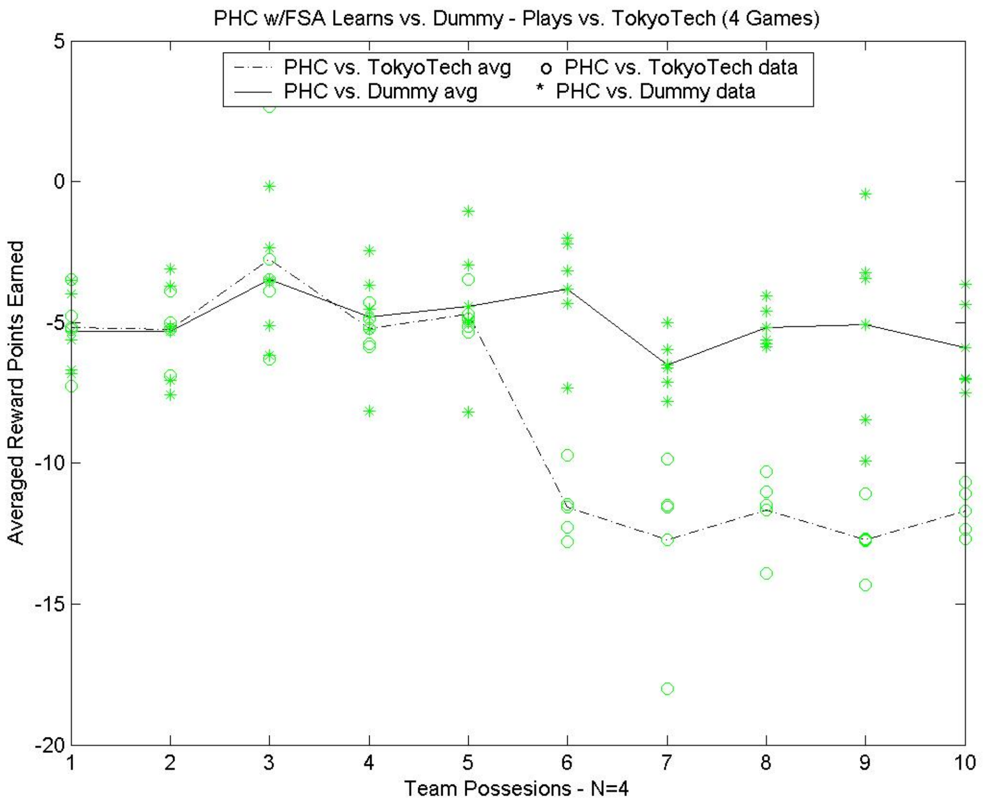

Figure 4.8: PHC Team learns against Dummy team – Plays against Dummy team and TokyoTech.

The results here are not unexpected. The PHC team accrues significantly fewer reinforcement learning points when it is playing against the TokyoTech team.

All of the experiments thus far have simply allowed the agents of the experimental team to maintain and use their own policy and q vector from game to game within a run of games. The final experiments investigate the impact allowing the agents to share information using weighted strategy sharing.

4.2.2 Results using Weighted Strategy Sharing

While the rate of learning in a single game is of interest, there is also value in demonstrating the effect of maintaining the Q values learned previously for use in later games. Rather than starting each new game "tabula rasa", the agents each store the Q-vector from their last game and use that as the initial Q-vector for the next game.

To further increase the rate at which the agents learn, weighted strategy sharing (WSS) is implemented. This allows each agent to benefit from the experiences of its team mates. Ideally, WSS would be implemented so the agents could share information while they played. The limited communications channels available to the players in the RoboCup preclude this. Instead, WSS is implemented at the end of the game to provide each agent with a new Q-vector for the next game based on the inputs of the other team members.

The results in Figure 4.9 are a comparison of the evaluated learning ability of the experimental team with and without weighted strategy sharing.

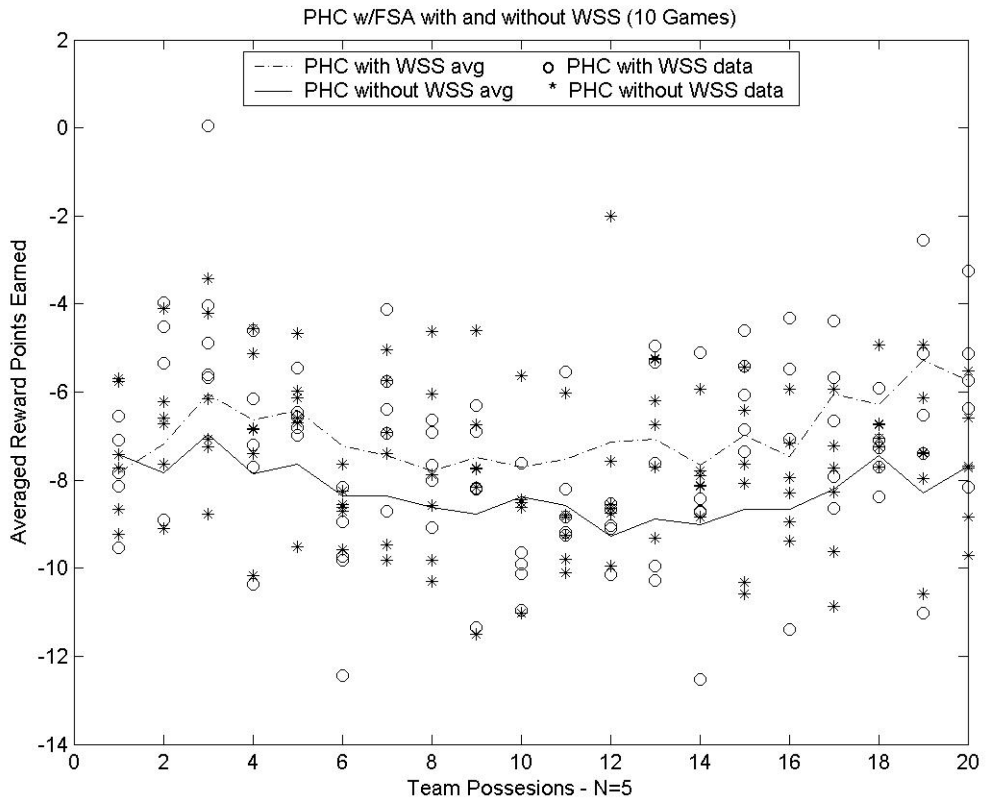

Figure 4.9: PHC with FSA Games With and Without WSS.

As anticipated, the experimental team shows an increased learning ability using WSS over the course of these games.

Clearly, the use of weighted strategy sharing increases the rate of learning over that of agents independently learning at their own rate.

4.3 Summary

The results presented here illustrate the benefit of off-policy learning (PHC) over the standard Q-learning. The comparison between the various policy hill climbers implemented in Tileworld makes it clear that the more powerful WoLF and PD-WoLF algorithms only produce improved results over the standard PHC if the search surface is sufficiently contoured, a characteristic minimized by the application of fuzzy state aggregation to the domain.

The results of implementing FSA and PHC for multiple agents in RoboCup demonstrate the portability of this method to other stochastic environments. The added benefit of weighted strategy sharing is also readily visible.

V. Conclusion and Future Work

Reinforcement learning in a stochastic environment is at best complicated, even for a single learning agent. Applying RL to a multi-agent system in such an environment simply compounds the difficulty. In addition to the rapidly changing elements indigenous to the environment, the other agents themselves become part of that environment and must be considered by each individual agent.

Standard reinforcement learning techniques such as Q-learning can be effective in such an environment unless the state space becomes too large. Even with a limited state space, on-policy learning methods (such as Q-learning) usually don't perform as well as off-policy learning methods such as policy hill climbers.

This chapter presents the research conclusions, significance, and recommended areas for future research.

5.1 The Research

This research presents an unusual approach to a difficult problem. By applying Fuzzy State Aggregation (FSA) to the environment, the state space is significantly constrained while still providing a good representation of the environment to support learning. This work demonstrates the improvement of combining FSA with each of three different PHC algorithms over standard Q-Learning. Both in terms of speed to convergence and the convergence value itself. The resulting increase in performance clearly shows the benefit of applying the off-policy hill climbing algorithm to the FSA in this highly stochastic environment. Unlike the results of using the WoLF-PHC and PDWoLF-PHC algorithms in a crisp environment, these two algorithms showed no improved performance over the common PHC algorithm.

The application of this same combination to the RoboCup soccer simulator also shows results consistent with those identified in the Tile world experiment. By constraining the state space and applying the reinforcement learning to the three offensive behaviors (shooting, passing and dribbling) we demonstrated the effectiveness of this method in a highly stochastic adversarial game setting. As a policy hill climber, this reinforcement learning method is not guaranteed to find the globally optimal solution. It does, however, consistently find a good solution.

While it is an interesting and entertaining pursuit, playing better robot soccer is not the only application for this type of RL. A swarm of micro UAVs or UGVs could benefit greatly from this type of on-the-fly decision making in response to rapidly changing environmental factors from avoiding threats to re-routing communications to compensate for a team mate that has broken down or moved out of range.

5.2 Future Work

Areas of future expansion includes applying the combination of fast policy hill climbing with fuzzy state aggregation to more complex domains in an effort to determine if the performance potential of the different algorithms maps to the fuzzy set aggregation function approximation method. A Tileworld with seven or nine fuzzy sets may provide a sufficiently contoured surface and result in the WoLF and PD-WoLF algorithms learning more quickly than standard PHC.

These is also potential benefit in learning the optimal fuzzy label values for each state variable as a means of further improving performance. Rather than hard-coding the center points for the triangular fuzzy sets, the agents can be programmed to learn and adjust those values to help bring performance closer to optimal.

In the soccer domain there is a great difference between learning well and playing well. In order to develop a competitive team based on this research, further work must be done on ball handling and extensive work is required to develop an effective goalie. Designing the agents to learn from, and anticipate, opponent behavior would likely be an area of interest for machine learning researchers. Finally, an over arching planner is required which will allow the agents to cooperate and create long term ‘plays’ and strategies required for an effective winning team.

Bibliography

[1] R.S. Sutton and A.G. Barto. *Reinforcement Learning: an Introduction,* Cambridge, Mass, MIT Press, 1998.

[2] S. Lawrence, A.C. Tsoi, and A.D. Back. "Function Approximation with Neural Networks and Local Methods: Bias, Variance and Smoothness", In: P. Bartlett, A. Burkitt and R. Williamson (eds), *Australian Conference on Neural Networks*, *16-21*, Australian National University, Australian National University, 1996.

[3] S.P. Singh, T. Jaakkola, and M.I. Jordan. "Reinforcement Learning with Soft State Aggregation," *In Advances in Neural Information Processing 7*, 361-368, MIT Press, 1994.

[4] H.R. Berenji and D. Vengerov. "Cooperation and Coordination Between Fuzzy Reinforcement Learning Agents in Continuous State Partially Observable Markov Decision Processes," *Proceedings of 1999 IEEE international Fuzzy Systems Conference*, 621-627, Seoul, Korea, 1999.

[5] H.R. Berenji and D. Vengerov. "Advantages of Cooperation Between Reinforcement Learning Agents in Difficult Stochastic Problems," *Proceedings of the 9th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '00)*, 871-876, San Antonio, Tx , 2000.

[6] M. Bowling and M. Veloso. "Multiagent Learning Using a Variable Learning Rate," *Artificial Intelligence, 136, no. 2*, 215–250, 2002.

[7] B. Banjeree and J. Peng. "Adaptive Policy Gradient in Multiagent Learning," *AAMAS '03 International joint conference on Autonomous Agent and Multi- Agent Systems*, Melbourne, Australia, 2003.

[8] M.N. Ahmadabadi, M. Asadpur, S.H. Khodaabakhsh and E.Nakano. "Expertness Measuring in Cooperative Learning," *Preceedings of IEEE/RSJ International Conference of Intelligent and Robot Systems (IROS)*, 2000.

[9] K. Cousin & G.L. Peterson. "Cooperative Reinforcement Learning Using an Expert-Measuring Weighted Strategy with WoLF," *Proceeding 481, Artificial Intelligence and Soft Computing* - 2005

[10] M.E. Pollack and M. Ringuette. "Introducing the Tileworld: Experimentally Evaluating Agent Architectures," *Eighth National Conference on Artificial Intelligence***,** Menlo Park, CA, 1990.

[11] C.J.C.H. Watkins. *Learning From Delayed Rewards*, Ph.D. dissertation, Cambridge University, Cambridge, United Kingdom, 1989.

[12] H.R. Berenji and D.A. Vengerov, "Cooperation and Coordination Between Fuzzy Reinforcement Learning Agents in Continuous State Partially Observable Markov Decision Processes," *Proceedings of the 8th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'99)*, 2000.

[13] H. R. Berenji and D. A. Vengerov, "Advantages of cooperation between reinforcement learning agents in difficult stochastic problems," in *Proceedings of the 9th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '00)*, 2000.

[14] H. R. Berenji and D.A. Vengerov, *Generalized Markov Decision Processes: Dynamic-Programming and Reinforcement-Learning Algorithms*, Technical Report IIS-00-10, Intelligent Inference Systems Corp., Sunnyvale, CA, October 2000.

[15] I. Gültekin and A. Arslan, "Modular-Fuzzy Cooperative Algorithm for Multi-Agent Systems," *Advances in Information Systems, Second International Conference, ADVIS 2002,* Izmir, Turkey, October 23-25, 2002*,*

[16] A. Kilic and A. Arslan. "Minimax Fuzzy Q-learning in Cooperative Multi-Agent Systems," *Advances in Information Systems, Second International Conference, ADVIS 2002,* Izmir, Turkey, October 23-25, 2002.

[17] M. L. Littman. "Markov Games as a Framework for Multiagent Learning," *Proceedings of the Eleventh International Conference on Machine Learning*, 157–163, San Francisco, California, 1994.

[18] T. Kawarabayashi, J. Nishino, T. Morishita, T. Kubo, H. Shimora, M. Mashimo, K. Hiroshima & H. Ogura. "Zeng99 in RoboCup-99," *RoboCup-99 Team Description: Simulation League*, 1999

[19] J. Rasmussen. "Skills, Rules, and Knowledge: Signals, Signs and Symbols, and Other Distinctions in Human Performance Models," *IEEE Transactions on Systems, Man and Cybernetics*. 13, No. 3, 257-266, 1983.

[20] E. Agirre, J. C. Gámez & A. González. "A Multi-agent System Based on Fuzzy Logic Applied to RoboCup's Environment," *Mathware & Soft Computing 8,* 153-178, 2001

[21] A. Bonarini, F. Marchese, M. Matteucci, M. Restelli, and D. Sorrenti. "Planning and Reaction in Milan Robocup Team," *Proceedings of the International RoboCup Symposium 2003*

[22] H.L. Sng, G. S. Gupta and C.H. Messom. "Strategy for Collaboration in Robot Soccer," *The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA '02)* 347, 2002.

[23] D. Gu and H. Hu. "Reinforcement Learning of Fuzzy Logic Controllers for Quadruped Walking Robots," *Proceedings of the 15th IFAC World Congress*, Barcelona, Spain, July 21-26, 2002.

[24] D. Gu, H. Hu & L. Spacek. "Learning Fuzzy Logic Controller for Reactive Robot Behaviours," *International Conference on Advanced Intelligent Mechatronics* (AIM 2003), Kobe, Japan, July 20-24, 2003.

[25] D. Gu and H. Hu. "Accuracy Based Fuzzy Q-Learning for Robot Behaviours," *Proceedings of IEEE International Conference on Fuzzy Systems*, Budapest, Hungary, 25-29 July 2004

[26] D. Gu and H. Hu. "Fuzzy Multi-Agent Cooperative Q-learning," *Proceedings of IEEE International Conference on Information Acquisition*, 193-197, Hong Kong, China, 27 June - 3 July, 2005.

[27] T. Nakashima, M. Udo, and H. Ishibuchi. "Acquiring the Positioning Skill in a Soccer Game using a Fuzzy Q-Learning," *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1488-1491, July 16-20, Kobe, Japan, 2003.

[28] J. Ammerlaan & D. Wright. "Adaptive Cooperative Fuzzy Logic Controller," *Computer Science 2004 Proceedings of the ACS Conferences in Research and Practice in Information Technology (CRPIT) 26*, 2004.

[29] L.A. Zadeh. "Fuzzy Sets," *Journal of Information and Control 8*, 338-353, 1965.

[30] S.N. Pant and K. E. Holbert. *Fuzzy Logic in Decision Making and Signal Processing*, http://ceaspub.eas.asu.edu/PowerZone/FuzzyLogic/chapter%202/frame2.htm, March 2004.

[31] T.F. Bersano-Begey, P. G. Kenny and E.H. Durfee. “Agent Teamwork, Adaptive Learning and Adversarial Planning in RoboCup using a PRS Architecture,” *RoboCup-97: The First Robot World Cup Soccer Games and Conferences*. Springer Verlag, Heidelberg, Germany, 1998.

[32] P. Stone, M. Veloso and P. Riley. “The CMUnited-98: RoboCup-98 Simulator World Champion Team,” *In RoboCup-97: Robot Soccer World Cup I*, 389–398. Hiroaki Kitano, editor, Springer Verlag, Berlin, 1998.

[33] J.L. de la Rosa, A. Oller, J. Vehi, and J. Puyol. ”Soccer Team Based on Agent-Oriented Programming,” *Robotics and Autonomous Systems, 21(2)*:161–176, October 1997.

[34] D. Parks. *Agent Oriented Programming Languages: a Practical Evaluation*, www.cs.berkeley.edu/~davidp/cs263/, December, 1997.

[35] A. Tews and G. Wyeth. “Multi-Robot Coordination in the Robot Soccer Environment,” *Proceedings of the Australian Conference on Robotics and Automation (ACRA '99)*, 90-95, Brisbane, Australia, March 30 - April 1, 1999.

[36] A. Drogoul and A. Collinot. “Applying an Agent-Oriented Methodology to the Design of Artificial Organizations: A Case Study in Robotic Soccer,” *Autonomous Agents and Multi-Agent Systems*, 1(1):113–129, 1998.

[37] A. Drogoul, *MICROB,* http://www-poleia.lip6.fr/~drogoul/projects/microb/microb.en.html

[38] A. Collinot, A. Drogoul and P. Benhamou. “Agent Oriented Design of a Soccer Robot Team,” *Proceedings of ICMAS’96.*

[39] I. Noda. “Team GAMMA: Agent Programming on Gaea,” *RoboCup-97: Robot Soccer World Cup I*, 500–507, Hiroaki Kitano, editor, Springer Verlag, Berlin, 1998.

[40] P. Scerri. “A Multi-Layered Behavior Based System for Controlling RoboCup Agents,”*RoboCup-97: Robot Soccer World Cup I*, Hiroaki Kitano, editor, 467–474. Springer Verlag, Berlin, 1998.

[41] G. Weiß. “Distributed Reinforcement Learning,” *Robotics and Autonomous Systems*, 15:135–142, 1995.

[42] M.J. Huber & E.H. Durfee. „Deciding When to Commit to Action During Observation-Based Coordination,” *Proceedings of the First International Conference on Multi-Agent Systems*, 163–170, Menlo Park, California, June 1995.

[43] P. Stone & M. Veloso. “A Layered Approach to Learning Client Behaviors in the RoboCup Soccer Server,” *Applied Artificial Intelligence*, 12:165–188, 1998.

[44] P. Stone. *Layered Learning in Multiagent Systems*. Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, 1998. Technical Report CMU-CS-98187.

[45] P. Stone & M. Veloso. “Team Partitioned, Opaque Transition Reinforcement Learning,” *Proceedings of the Second International Conference on Autonomous Agents*, 206–212, 1999.

[46] P. Stone & M. Veloso. “Multiagent Systems: A Survey from a Machine Learning Perspective,” *Autonomous Robotics 8, no. 3*, July, 2000.

Vita

Captain Wardell enlisted in the Air Force in September 1985. After basic training, he remained at Lackland Air Force Base in San Antonio, Texas, to study Russian at the Defense Language Institute (DLI). Upon completion of that 47-week course, he moved to Goodfellow Air Force Base in San Angelo, Texas for the Cryptologic Linguist Course.

In June 1987, he reported to his first assignment at Detachment 4, 691st Electronic Security Wing in Stuttgart, Germany, where he served as a tactical intelligence collection specialist. There he worked with members of 207th Military Intelligence Brigade, US Army VII Corps, supporting the *Guardrail* platform. When Iraqi forces invaded Kuwait, Captain Wardell and nine other members of his detachment deployed with VII Corps to Al Qaysumah, Saudi Arabia, for Operations Desert Shield, Desert Storm and Desert Calm.

He returned to Goodfellow Air Force Base in June 1991, to teach the Cryptologic Linguist Course for incoming Russian Linguists. While serving as an instructor there, he was accepted into the Airman's Education and Commissioning Program (AECP).

In January 1994, Captain Wardell was assigned to Wright State University, Dayton, Ohio to earn a Bachelor of Science Degree in Electrical Engineering. While attending school, he was inducted into the engineering honor society *Tau Beta Pi*. Captain Wardell graduated *Magna Cum Laude* and received his commission through the Officer Training School at Maxwell Air Force Base, Alabama.

After OTS he was assigned as a Target Identification Research Engineer for the Sensors Directorate, Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio in December 1996. There he concentrated his work in the area of automatic target recognition of moving ground targets. While there, he also served as the Assistant Executive Officer for the Sensors Directorate of the Research Lab.

In December 1998, Captain Wardell was assigned to Technical Engineering, 341st Space Wing, Malmstrom Air Force Base, Montana. There he served as a Missile System Engineer and as the Officer in Charge of Technical Engineering. While there he also completed a Masters of Science degree in International Relations.

Captain Wardell returned to Wright-Patterson Air Force Base, Ohio in November 2001 to serve as a Senior Intelligence Support Engineer in the Advanced Programs Division of the National Air and Space Intelligence Center. It was during this assignment that he was selected for AFIT.

He started classes in June 2004 for a Masters of Science in Electrical Engineering (Software Engineering). After graduation Captain Wardell will be assigned to the Air Force Technical Application Center (AFTAC) at Patrick Air Force Base, Florida.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 23-03-2006	**2. REPORT TYPE** **Master's Thesis**	**3. DATES COVERED** *(From – To)* Jun 2004– Mar 2006

4. TITLE AND SUBTITLE

Application of Fuzzy State Aggregation and Policy Hill Climbing to Multi-Agent Systems in Stochastic Environments

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Wardell, Dean C., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/06-55

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/SNAT	AFRL/SNRN
Attn: Robert Williams, PhD.	Attn: Mikel M. Miller, PhD.
Bldg 620	Bldg 620
2241 Avionics Circle	2241 Avionics Circle
WPAFB, OH 45433-7333	WPAFB, OH 45433-7333
DSN: 785-6427 ext. 4389	DSN: 785-6127 ext. 4274
E-mail: robert.williams@wpafb.af.mil	E-mail: mikel.miller@wpafb.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Reinforcement learning is one of the more attractive machine learning technologies, due to its unsupervised learning structure and ability to continually learn even as the operating environment changes. Applying this learning to multiple cooperative software agents (a multi-agent system) not only allows each individual agent to learn from its own experience, but also opens up the opportunity for the individual agents to learn from the other agents in the system, thus accelerating the rate of learning.
This research presents the novel use of fuzzy state aggregation, as the means of function approximation, combined with the policy hill climbing methods of Win or Lose Fast (WoLF) and policy-dynamics based WoLF (PD-WoLF). The combination of fast policy hill climbing and fuzzy state aggregation function approximation is tested in two stochastic environments; Tileworld and the robot soccer domain, RoboCup. The Tileworld results demonstrate that a single agent using the combination of FSA and PHC learns quicker and performs better than combined fuzzy state aggregation and Q-learning lone. Results from the RoboCup domain again illustrate that the policy hill climbing algorithms perform better than Q-learning alone in a multi-agent environment. The learning is further enhanced by allowing the agents to share their experience through a weighted strategy sharing.

15. SUBJECT TERMS
Multi-Agent Architecture, Fuzzy Logic, Machine Learning, Reinforcement Learning, Policy Hill Climbing, State Aggregation, Win or Lose Fast (WoLF), Stochastic Environment, Weighted Strategy Sharing

16. SECURITY CLASSIFICATION OF: UNCLASSIFIED			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Gilbert L. Peterson AFIT/ENG
REPORT U	**ABSTRACT** U	**c. THIS PAGE** U	UU	79	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4281; e-mail: Gilbert.Peterson@afit.edu