REINFORCEMENT LEARNING USING POTENTIAL FIELD FOR ROLE ASSIGNMENT IN A MULTI-ROBOT TWO-TEAM GAME

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖZGÜL FİDAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
THE DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof . Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof . Dr. İsmet Erkmen

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Prof . Dr. Aydan Erkmen	Prof. Dr. İsmet Erkmen
Co-Supervisor	Supervisor

Examining Committee Members

Prof. Dr. Kemal Leblebicioğlu	(METU, EE)	_____
Prof. Dr. İsmet Erkmen	(METU, EE)	_____
Prof. Dr. Aydan Erkmen	(METU, EE)	_____
Assoc. Prof. Dr. Aydın Alatan	(METU, EE)	_____
Ass. Prof. Dr. İlhan Konukseven	(METU, ME)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Özgül Fidan

Signature:

ABSTRACT

REINFORCEMENT LEARNING USING POTENTIAL FIELD FOR ROLE ASSIGNMENT IN A MULTI-ROBOT TWO-TEAM GAME

FİDAN, Özgül

MSc., Department of Electrical and Electronic Engineering

Supervisor: Prof. Dr. İsmet ERKMEN

Co-Supervisor: Prof. Dr. Aydan ERKMEN

December 2004, 75 pages

In this work, reinforcement learning algorithms are studied with the help of potential field methods, using robosoccer simulators as test beds.

Reinforcement Learning (RL) is a framework for general problem solving where an agent can learn through experience. The soccer game is selected as the problem domain a way of experimenting multi-agent team behaviors because of its popularity and complexity.

Keywords : multi agent systems, reinforcement learning, robosoccer, potential field

ÖZ

ÇOKLU-ROBOTLU, İKİ-TAKIMLI BİR OYUNDA ROL BELİRLEMEYİ POTANSİYEL ALANLAR KULLANARAK YAPAN PEKİŞTİRMELİ ÖĞRENME

FİDAN, Özgül

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. İsmet ERKMEN

Ortak Tez Yöneticisi : Prof.Dr. Aydan ERKMEN

ARALIK 2004, 75 SAYFA

Bu çalışmada, robot futbolu simulatörleri test yatakları olarak kullanarak, potansiyel alan metodunun yardımıyla pekiştirmeli öğrenme algoritmaları araştırılmıştır.

Pekiştirmeli öğrenme (RL), elemanları tecrübe ile öğrendikleri genel problem çözümü için bir çerçevedir. Çoklu eleman davranışlarını incelemek için popülerliği ve karmaşıklığı sebebiyle robot futbolu seçildi

Anahtar Kelimeler : çoklu elemanlı sistemler, pekiştirmeli öğrenme, robot futbolu, potansiyel alanlar

To My Parents

ACKNOWLEDGEMENTS

I would like to express my gratitude to Prof. Dr. İsmet Erkmen and Prof. Dr. Aydan Erkmen for their valuable supervision, understanding.

Words are inadequate to express my thanks to Kemal Kaplan for his wisdom, patience, concern and technical supports in this study.

I am grateful to my family, especially to my sister, for their endless trust.

My special thanks go to Ali Osman Boyacı and to all other friends who gave me love and hope throughout this study.

I would like to thank to Tansel Alpsel for his support and limitless understanding.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
MAS	Multi-Agent Systems
RL	Reinforcement Learning
DAI	Distributed Artificial Intelligence
ASM	Action Selection Mechanism
MLP	Multilayer Perception
VRML	Virtual Reality Modelling Language
MDP	Markov Decision Process
SAP	State Action Pair
TD	Temporal Difference Learning
TD(λ)	Temporal Difference with Eligibility Traces
QL	Q Learning
CMAC	Cerebellar Model Articulation Controller

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

We are interested in games played by two teams, consisted of agents, in dynamic environments. Games have their own rules but mostly each has an aim to defeat the other team, by gaining more scores. The environment consists of opponents, teammates, a ball, the team goals, and physical or rule-based limitations. Opponents, teammates and ball are dynamic components and cause the environment to change continuously. So in a such competitive world the cooperation and coordination of teams are of great importantce. To achieve accurate coordination the teams should gather adequate information from the environment. The agent capabilities will effect the team's performance, but also building strategies and planning are important issues for the success of the team. We aim at constructing a system that adapts its strategy according to the changing game conditions. Our system should learn how to play against different teams based on its experience, and its strategy should improve according to the team he played against.

1.2 MOTIVATION

Many AI researchers are today striving to build agents for complex, dynamic multi-agent domains. Such domains include virtual theatre (Hayes-Roth, Brownston, & Gen 1995),realistic virtual training environments (e.g., for emergency drill(Pimental

& Teixeira 1994) or combat (Tambe et al. 1995)), virtual interactive fiction(Bates, Loyall, & Reilly 1992) and RoboCup robotic and virtual soccer(Kitano et al. 1995).

There is no generally accepted definition of *agents* in Artificial Intelligence (AI) [Stone,2000], an agent as a robot which has goals, actions and domain knowledge, situated in a environment. The ways it acts are called its *behaviors*.

An agent can be defined in the field of computer science, as a program, or piece of program, that functions as an active entity in a computerized environment together with other processes or programs.

Due to the distribution of control and reasoning,the multiagent systems (MAS) approach in Artificial Intelligence (AI) is well-suited for solving problems in complex domains, and this distribution enables agents to react to dynamic external events as they collaborate to attain their long-term team goals.

When a group of agents in a MAS share a common long-term goal, they can be said to form a *team* [Stone, 2000]. Other agents in the environment that have goals opposed to the team's long-term goal are called the team's *adversaries*.

Therefore in a research on collaborative team behavior of autonomous robots, the following subtopics are involved:

1. Multiagent Machine Learning (ML) – the processes and approaches of the team to learn their behaviors. In other words, to train the team to use the team strategies and also develop new team strategies.

2. World Model (WM) Construction – Generation of the World Model which contains the ball location, teammate location and orientation, opponent location and the goal location on the soccer field. This information is shared among the team. This is not an easy task as the game is real-time and all objects except the goal are changing very fast.

3. Cooperation – Share information, role assignment and coordination of the team. The team makes group decisions on team strategy and team members positions

themselves on suitable location to perform the task based-on the team strategy. The team's strategy needs to be adaptive to the behavior of the opponent and to follow plans made on or before the game.[8]

Cooperation involves multiple robots working towards a common goal. It is difficult to implement meaningful robot operations in these environments due to the constantly changing environment features making action-selection and action-location highly transient.

[Vail & Veloso, 2003] used the shared potential fields to solve the role assignment and coordination problem. The potential fields were based on the positions of the other robots on the team and the ball. The robots positioned themselves on the field by following the gradient to a minimum of the potential field. In principle, potential functions can be applied to any multi-robot domain. Robots perform distributed task allocation by calculating their suitability for a task and broadcast this suitability as a *bid* to their teammates. The robot with the highest bid wins the task. If the winning robot becomes unavailable some reasons, the robot with second highest bid wins the task. They test the approach on non-adversarial games by taking away the opponents and compare the average scoring time of a single robot, three robots with no coordination and three robots with coordination. Three robots with coordination scores most quickly and three robots without coordination perform the worst. They also applied this approach in winning championship in RoboCup 2002, Sony Legged League.

First, since complex environments with continuous states and actions have very large search spaces, at the single agent level, the solution methodology must address the problem of how to reduce these large search spaces. At the team level, it must address the problem of how to enable autonomous agents to collaborate efficiently and coherently.

The question then becomes: How can we build a multiagent system that can execute

high-level strategies in complex, dynamic, and uncertain domains? We may consider three possible answers to this question. First, we can build a system that successively

executes plans chosen from a static library. Second, we can build a system that can learn its strategies from scratch. Third, we can describe the strategies symbolically at an implementation-independent level and have the system learn how to implement the necessary implementation-level details under varying conditions to be effective in situated scenarios.

Implementation of a team of agents is a very complex task because the design includes not only the actions of agents, but also the coordination of agents. Reinforcement learning (RL) is a framework for general problem solving where an agent can learn through experience.

Most work in the multiagent learning literature has treated the challenge of building team-level strategies as a Reinforcement Learning (RL) problem. RL generates a strategy that emerges from an incrementally modified sequential decision memory over many training iterations. In complex domains, bottom-up learning techniques require practical convergence to provide stable policies, and, by their nature, they do not bound the search problem beyond rewarding every decision according to its perceived value since they intend to discover policies. Moreover, they suffer from the exponential growth of the search space as a factor of the size of the input vector. Therefore, scaling bottom-up learning approaches to large search spaces is a very difficult problem. On the other hand, we hypothesize that top-down approaches can constrain the search space, resulting in a more effective method for learning in multiagent systems.

A plan is initially a high-level specification for the implementation of a strategy, and it is decomposed into an ordered list of steps each of which may require the collaboration of multiple agents to implement its goal. A plan step, in turn, defines a specific role for each collaborating agent. A role describes the necessary conditions for executing and terminating the responsibilities of each given agent from that agent's local perspective of the world. Since the sequence of actions required for each situation can vary, before any learning takes place, a plan step does not contain any implementation-specific details about what actions each collaborating agent

needs to take to perform its own task in that plan step. Therefore, at the outset, a plan is only a high-level specification of a strategy whose implementation-level details need to be acquired from experience in a situated environment. To acquire these details, our approach uses learning.

Unlike systems that learn policies directly from experience in a bottom-up fashion, our system does not learn plans or policies. By training with different teams , the system acquires action knowledge that enables it to adapt its aim to specific situations.

1.3 STATE OF THE ART

We describe a framework for controlling and coordinating a group of robots for cooperative manipulation tasks. The framework enables a centralized approach to planning and control. We construct a system which adapts its coordination system according to the changing environment. We based our learning approach on the idea of learning by that is, learning from practice, and we established the benefits of our solution experimentally.

Our aim is to explore the possibility of learning multi-agent team coordination through reinforcement learning using potential fields as the state variables taken from the environment. Our system learns to assign correct roles to the appropriate agents according to the information taken from environment.

As a testbed to demonstrate our methodology, we used the Teambots simulated robotic soccer environment. The soccer game provides a very rich multi-agent environment that is both dynamic, complex, and uncertain.

Agents in the Teambots simulated soccer environment are capable of only basic actions which need to be sequenced in order to provide meaningful behavior.

Teambots is an open source simulator, written in Java. It provides an environment in which we can test our team with opponents which have different capabilities.

To implement our system and to demonstrate the learning capability of our methodology, we manually implemented high-level individual skills based on the basic behaviors

provided by the simulator. This hierarchical approach is akin to the layered learning in [Stone and Veloso, 1999; Stone, 1998], which used machine learning techniques.

While constructing the state vector, potential field information are used as magnitude and direction of vectors for each agent, and they are parametrically quantized. Different from other studies,[20] we take the resultant potential vector that affects on each agent to decrease the state vector dimension. To achieve this, we create our potential field equations for each object in the environment, that give enough information for our sytem coordination. In addition to the perceptual state variables, the zones that our agents occupy and the direction of the ball in that time interval are used as state variables.

Our thesis proposes a methodology for constructing a learning by doing solution to complex multiagent problems in dynamic and unpredictable physical environments.

We used Sarsa(λ) as the learning algorithm, and tile coding property of CMAC as function approximator.

With three opponent teams which have different capabilities, we train our team by making 1000 matches. The first team is GoToBall Team which has the only ability to follow the ball.The second team is BrianTeam which is stronger than GoToBall Team and provides the basic soccer team capabilities. The third team is AIKOHomoG which is the strongest of three, which has an high level coordination system and whose agents are capable of abilities for good playing soccer.

CHAPTER 2

LITERATURE SURVEY

The subjects, reinforcement learning, potential fields and multi agent systems, are very popular in computer science. The robocup organizations provides us to find many articles written about this subjects.

2.1 ROBOT SOCCER SYSTEMS

2.1.1 INTRODUCTION

The development of a multi-agent system amounts to searching for a method that will implement an intelligent system composed of multiple agents, cooperating with each other, with independent motion control. Multi-agent robotic systems are more flexible and fault tolerant as several simple robot agents are easier to handle and cheaper to build compared to a single powerful robot for different tasks.

The soccer game is different from other multi-agent systems, in that the robots of one team have to cooperate, while facing competition from the opponent team. The multi-agent control algorithm, in such an active environment, must comprise of low level kinematics and dynamics, and high level strategies to avoid obstacles and to compete with opponent robots. Such an environment needs fast processing algorithms and suitable robot structure.

From the standpoint of multi-agent systems, the soccer game is a good example of the problems in real world, which can be moderately abstracted.

Multiple robot systems show better fault tolerance and are easier, cheaper and flexible. The collective behavior of different agents deals with cooperative and/or competitive behaviors. The robots have to cooperate with others to achieve specific objectives. task allocation and decomposition are associated with collective behavior. It also requires a form of communication for cooperation among different robots. Learning can comprise of knowledge and skill acquisitions. Improving the realtionship with environment through interaction, helps agents to adapt.

The real-time decision making problem and action selection mechanism (ASM) for each agent given its role such as striker and goal-keeper in a robot soccer game can be achieved using a multilayer perception (MLP) to learn human judgment for the action selection.

In order for a robot team to actually perform a soccer game, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion.

2.1.2 TEAMBOTS

Teambots is a JAVA based simulator for multi-robot teams developed by Tucker Balch. It has different applications as well as robot-soccer simulator. The control applications of agents are also implemented in JAVA, however the simulator uses a description file for modelling the world. The simulator calls two functions from the strategy class, which extends abstractrobot.ControlSystemSS class. The first function Configure is called at the beginning of simulation. The other function, TakeStep, is called repeatedly by the simulation kernel to allow the programmer to read sensors and select actions. Unlike the previous simulators, there is no central controller. The environment is completely distributed.

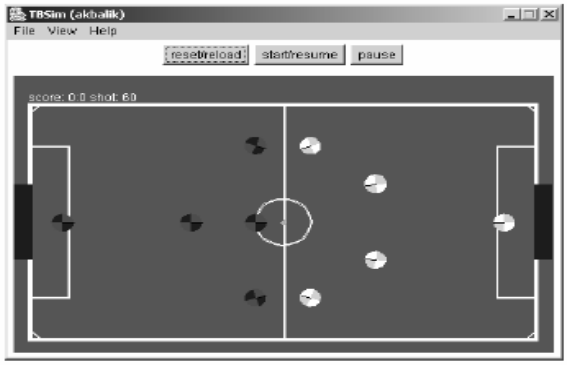


Figure 2-1 Teambots Simulator

2.2 MULTI-AGENT SYSTEMS

One of the most challenging goals in artificial intelligence (AI) is the development of artificial intelligent autonomous agents with human-level performance. The past few years has witnessed a tremendous interest in research and discussions on intelligent agents. With the ever increasing number of robots in an industrial environment, scientists and technologies are often faced with issues on cooperation and coordination among different robots and their self governance in a common work-space. This has led to developments in multi-robot cooperative autonomous systems. With an aim to study issues such as group architecture, resource conflict, origin of cooperation, learning and geometric problems, groups of robots are constructed. [1]

2.2.1 AGENT AND AUTONOMOUS AGENTS

We define the term agent as just about anything that can perceive its environment through sensors and can act through actuators. Humans, animals are good examples for agents and also robots and certain communicating software programs (“softbots”) are accepted as agents. In the field of computer science the term agent refers to a program, or piece of program, that functions as an active entity in a computerized environment together with other processes or programs. [2]

An agent is called autonomous if it operates completely autonomously, that is, if it decides itself how to relate its sensory data to motor commands in such a way that its goals are attended to successfully.

For an agent to be autonomous it has to reason about its environment before acting upon it. Figure 2-1 shows a diagram of an autonomous robot and its environment.

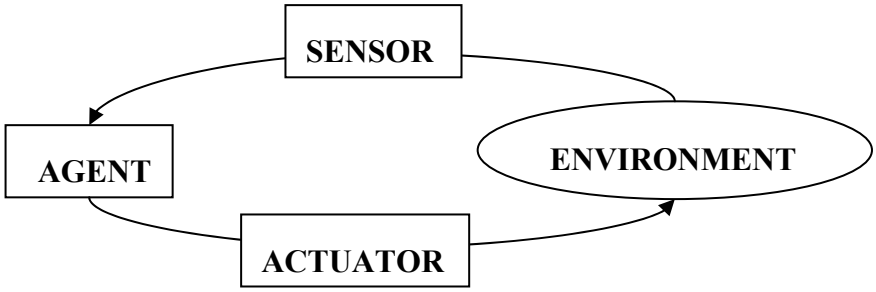


Figure 2-2 An Agent Interacting With Its Environment

2.2.2 LEARNING IN MULTI-AGENT SYSTEMS

A system can learn problem decomposition by acquiring instances from a human operator and by generalizing on them. Learning from experience can help an agent to reduce the need for negotiations or make it more directed, avoiding harmful interactions. An agent can learn other agents' intentions and beliefs as well as the characterization of the task environment. The modelling of another agent's goals and beliefs will enhance its ability to reason about other agents and improve its ability to coordinate its own activities. It helps to reconsile the conflicting intentions within and/or between agents. Adaptive agents are those which change their behaviors in uncertain and dynamical environments. To cope with such situations, an agent must be self-adaptive, self learning, and should have the capabilities to face real situations through cooperative behaviors with other agents, as well as, be capable of perceiving environmental changes. The knowledge and intelligence can not be programmed as such in the beginning, however it can be acquired from real environments and

through reactive behaviors with other agents [1]. Reinforcement learning [3, 4] and classifier systems [5] have laid the basis for learning in multi-agent systems.

2.2.3 MULTI-AGENT SYSTEM STRUCTURES

The agents in a multi-agent scenario, may have homogeneous or heterogeneous structures. In a homogeneous MAS, all of the agents have identical structure (goals, domain knowledge, and set of actions). They may differ by way of their sensor input and effector output. In a heterogeneous situation, agents can have different goals,domain knowledge and actions. The agents in such a system may be friendly (benevolent) or may be inhibiting each other (competitive).

The two general types of designs seen among the multi-agent system structures are the hierarchical and behavior structures. They differ on the type of information they process and in their interconnections [6]. In a hierarchical structure, the control issue is divided along functional lines into progressive levels of abstraction of data. It uses computational functions for system decomposition. Processes handling data deal with different information content and hence such a system sequences data from a perception process, to a decision-making process, through a series of action processes to the actuators. In the case of the behavior structure, the control problem is broken into behaviors without any central intelligent agent present. Through interaction between the competing constituents, emergent behaviors result. Each behavior is nothing but a compound module carrying out the main control computatinal function. These systems require action arbitration mechanisms, as differeent actions arise from different behaviors.

Compared with the previous research focus on the management of information among agents, the recent developments centers around the behavior management in an intelligent system. Hybrid control structures [7], a combination of hierarchical and behavior structures, can take the strengths from either of the structures and can get away with the drawbacks associated with both.

2.2.4 MULTI-AGENT COOPERATION

Cooperation among agents can be explicit or implicit. In case of the explicit information exchange, the agents perform actions to benefit other agents. However, in the implicit case, agents carry on with their own goal-seeking process and these actions will be beneficial to others. Information exchange through communication is an effective way for interaction among the agents. Through communication an agent can get a global view of the problem at hand, helping to take appropriate local decisions with a global view point.

Communication among agents opens a multitude of issues in the MAS scenario. Pasing wrong information to misguide another agent, decision to stop communication by a single or a group of agents, the degree of sharing information, etc., are issues to be examined in greater detail.

2.3 POTENTIAL FIELDS

When you think of potential fields, picture in your mind either a charged particle navigating through a magnetic field or a marble rolling down the hill. The basic idea is that behavior exhibited by the particle/marble will depend on the combination of the shape of the field/hill. Unlike fields/hills where the topology is externally specified by environmental conditions, the topology of the potential fields that a robot experiences are determinded by the designer. More specifically, the designer (a) creates multiple behaviors, each assigned a particular task or function, (b) represents each of these behaviors as a potential field, and (c) combines all of the behaviors to produce the robot's motion by combining the potential fields. [Appendix]

The potential field approach has been first proposed by Khatib[9] as an on-line collision avoidance approach. According to this, the robot moves in a field of forces, it senses its environment during motion execution and should be attracted toward its goal (attractive potential field) while being repulsed by obstacles (repulsive potential fields).[10]

They have a low computational overhead in comparison to higher level approaches such as path planning, they require simple, local knowledge about the environment; and because they do not require simple, local knowledge about the environment; and, because they do not require simple, local knowledge about the environment; and, because they do not require computationally expensive repair, such as replanning, when the environment changes, they are robust in dynamic situations. On the other hand, potential fields have a tendency to guide robots to local rather than global minima. However, in highly dynamic environments such as soccer, this is not a major problem as the world quickly changes and jogs the robot from the local minimum. [11]

In addition to static obstacle avoidance, potential fields may also be used for multi-agent formations and coordination. In [12],[13], Balch *et al* describe how robots can form and maintain formations using only local information to calculate potential fields. They name their approach "social potentials" because the potential functions are calculated using the distances between teammates. In [14][15], potential fields are used to position robots for particular roles.The potentials encode heuristic information about the environment. This information takes the form of attractive potentials that guide robots to desirable areas of the field. [11]

2.3.1 VECTOR FIELDS

The vector field model aims to associate with each point a vector indicating which direction the robot should head while it passes through that point. The collection of vectors at each point in a two dimensional field is called a potential field because it represents the syntetic energy potentials that the robot will follow.

A goal seeking schema (Figure 2) sets up an attractive field of vectors oriented towards the goal. On the other hand an obstacle-avoidance schema (Figure 3) associates with each obstacle a repulsive field. A row of obstacles, such as a wall, are combined to create a vector field, which would tend to move an object towards either end of the wall. For each of these types of fields,the strength of the vectors decreased

with the distance from the object creating the field. The schemas are then combined by summing their associated vector fields. [16]

Some types of potential fields are shown below:

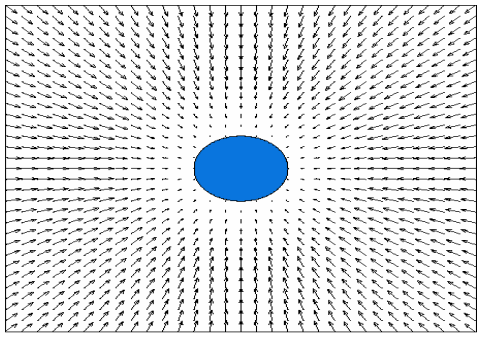


Figure 2-3 An Attractive Potential Field, SeekGoal Behavior

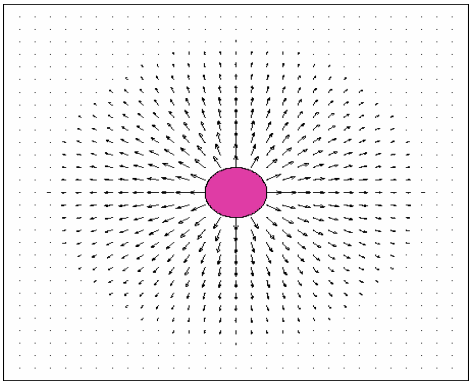


Figure 2-4 A Reject Potential Field, AvoidObstacle Behavior

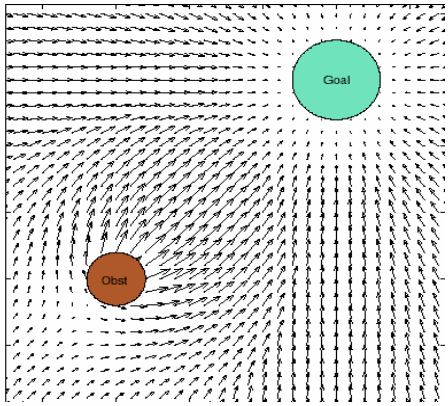


Figure 2-5 Potential Field Generated by An Attractor and An Obstacle

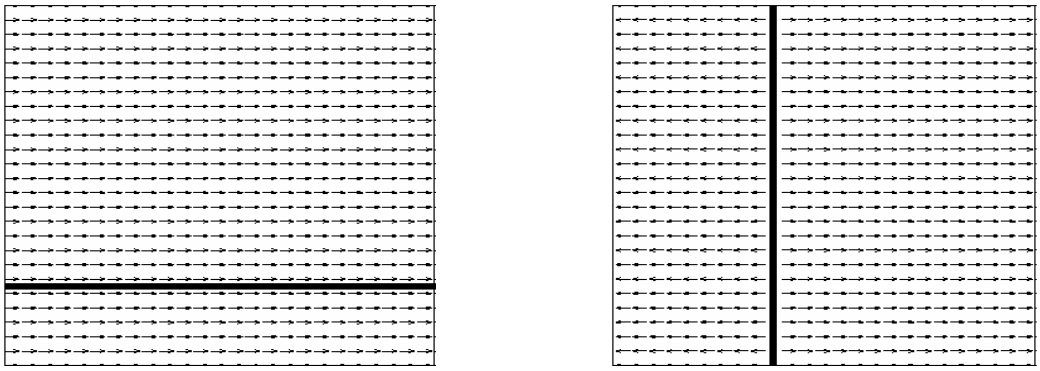


Figure 2-6 (a)Uniform, (b) Perpendicular Potential Field

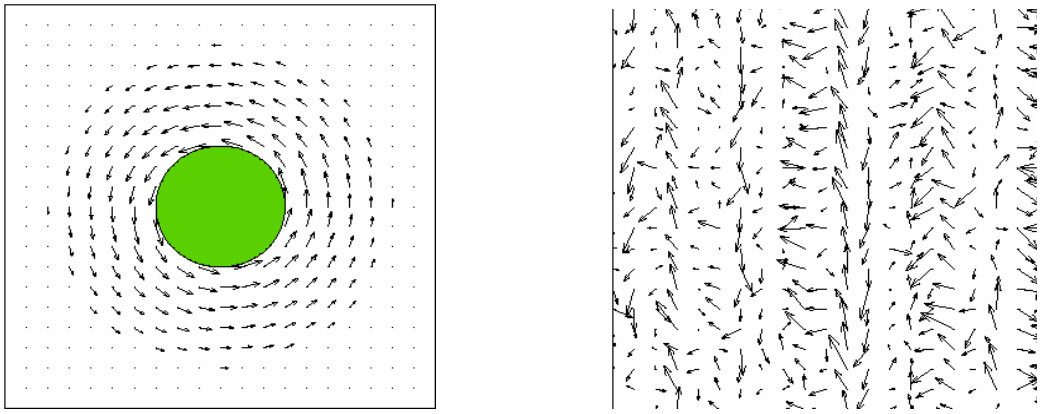


Figure 2-7 (a)Tangential, (b) Random Potential Field

Two types of methods for generation of potential fields are shown in Appendix1.

2.4 REINFORCEMENT LEARNING

2.4.1 REINFORCEMENT LEARNING

Reinforcement learning (RL) is a generic name given to a family of techniques in which an agent tries to learn a task by directly interacting with the environment. The method has its roots in the study of animal behavior under the influence of external stimuli. The agent's duty is to find a way, mapping states to actions. Figure 2-3 shows the agent's learning cycle during the interaction with the environment.

Reinforcement learning is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment.

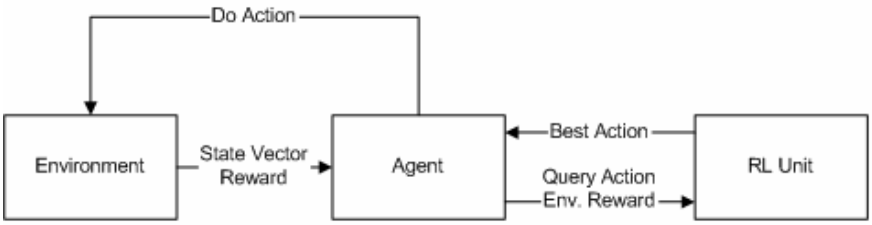


Figure 2-8 Reinforcement Learning Schema

There are two main strategies for solving reinforcement-learning problemS. The first is to search in the space of behaviors in order to find one that performs well in the environment. This approach has been taken by work in genetic algorithms and genetic programming, as well as some more novel search techniques (Schmidhuber, 1996). The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world.

Reinforcement learning differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action the agent is told the immediate reward and the subsequent state, but is not told which action would have been in its best long-term interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally. Supervised learning can be achieved by numeric methods such as NN or by deliberate planning where the supervisor supplies relevant information about the environment to the agent. Most of these algorithms need representation or model of the world namely the predefined knowledge of the world. Modeling of the world is not possible or feasible for most of the complex real world probleme. RL methods are generally independent of the environment that the agent experiences. RL methodology is based on agent's interactions with the environment. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning. Online performance of a RL algorithm is very important with respect to other supervised learning techniques. Generally for real world and simulated environments the agent should perform its actions in limited amount of time.

For real world tasks, world models are not available a priori, they have to be developed first. For most complex problems a complete model is not even needed, since it is unlikely that the agent (policy) will traverse all states – the agent's subjective world is only an approximation of the objective world. RL provides us a framework for training an agent by exploring an environment and learning from the outcomes of such trials [17]. In RL problems, an agent receives input from the environment, selects and executes an action, and receives reward which tells how good its last action was. The goal of the agent for each state is to select the action which leads to the largest future discounted cumulative rewards which is the definition of the learning task itself [18]. To solve this problem RL defines two parameters as state value and Q-value. Q-value is the average estimated gain by taking a specific action in a specific state. The state value is the best action's Q-value. This recursive definition is the underlying idea behind the RL methods. Calculation of these values may vary among definitions and implementations of algorithms but the fundamental idea is the same.

In the last two decades, RL has been extensively studied in artificial intelligence. The field of single-agent RL is nowadays mature, with well-understood theoretical results and many practical techniques (Suttonand Barto, 1998).

On the contrary, the field of multiagent reinforcement learning in which many agents are simultaneously learning by interacting with the environment and with each other, is less mature. The main reason is that many theoretical results for single-agent RL do not directly apply in the case of multiple agents. There are also computational issues like the difficulty of dealing with exponentially large state/action spaces, and the intractability of several distributed decision making algorithms (Bernstein et al., 2000).

Recent efforts involve linking multiagent RL with game-theoretic models of learning, with promising results (Claus and Boutilier, 1998; Wang and Sandholm, 2003).

2.4.2 REINFORCEMENT-LEARNING MODEL

In the standard reinforcement-learning model, an agent is connected to its environment via perception and action. On each step of interaction the agent receives as input, i, some indication of the current state, s, of the environment; the agent chooses an action, a, to generate as output. The action changes the state of the environment, and the value of this state transition is communicated to the agent through a scalar reinforcement signal, r. The agent's behavior, B, should choose actions that thend to increase the long-run sum of values of the reinforcement signal.

Formally, the model consists of

- a discrete set of environment states, S;
- a discrete set of agent actions, A; and
- a set of scalar reinforcement signals; typically {0,1}, or the real numbers.

The figure also includes an input function I, which determines how the agent views the environment state.

An intuitive way to understand the relation between the agent and its environment is with the following example dialogue.

Environment: You are in state 65. You have 4 possible actions.
Agent: I'll take action 2.

Environment: You received a reinforcement of 7 units. You are now in state 15.

Agent: I'll take action 1.

Environment: You received a reinforcement of -4 units. You are now in state 65. You have 4 possible actions.

Agent: I'll take action 2.

Environment: You received a reinforcement of 5 units. You are now in state 44. You have 5 possible actions.

. .

. .

. .

The agent's job is to find a policy π, mapping states to actions, that maximizes some long-run measure of reinforcement. In general, we expect that the environment will be non-deterministic; that is, that taking the same action in the same state on two different occasions may result in different next states and/or different values.

Some aspects of reinforcement learning are closely related to search and planning issues in artificial intelligence. AI search algorithms generate a satisfactory trajectory through a graph of states. Planning operates in similar manner, but typically within a construct with more complexity than agraph, in which satates are represented by compositions of logical expressions instead of atomic symbols. These AI algorithms are less general than the reinforcement-learning methods, in that they require a predefined model of state transitions, and with a few exceptions assume determinism. On the other hand, reinforcement learning, at least in the kind of discrete cases for which theory has been developed, assumes that the entire state space can be enumerated and stored in memory – an assumption to which conventional search algorithms are not tied. [19]

2.4.3 EXPLORATION POLICIES

An important issue in multiagent RL is how an agent chooses his exploration policy.

One major difference between reinforcement learning and supervised learning is that a reinforcement-learner must explicitly explore its environment. If all observed rewards are exactly equal, a simple method is to select a joint action in state s according to a Boltzmann distribution over joint actions using the current $Q^{(i)}(s, a)$ (which will be the same for all agents). Each agent can sample a joint action from this distribution by using the same random number generator (and same seed). This ensures that all agents will sample the same exploration action a. Then each agent can select his action a_i as the component i of the selected a.

Q-learning with an exploration policy like the above and common knowledge assumptions about parameters like the random number generator and seed, implies in effect that each agent runs Q-learning over joint actions identically and in parallel.

This guarantees the convergence of the algorithm, under conditions similar to those in single-agent Q-learning. Equivalently, if a transition model is available, value iteration can also be performed by each agent identically. In this way, the whole multiagent system is effectively treated as a `big' single agent, and the learning algorithm is simply reproduced by each agent.

2.4.4 MARKOV DECISION PROCESSES

Problems with delayed reinforcement are well modeled as Markov decision processes MDPs). An MDP consists of

- a set of states S,
- a set of actions A,
- a reward function $R : S \times A \rightarrow R$, and
- a state transition function $T : S \times A \rightarrow \Pi(S)$, where a member of Π (S) is a probability distribution over the set S (i.e. it maps states to probabilities). We write T(s, a, s') for the probability of making a transition from state s to state s' using action a.

The state transition function probabilistically specifies the next state of the environment as a function of its current state and the agent's action. The reward function specifies expected instantaneous reward as a function of the current state and action. The model is Markov if the state transitions are independent of any previous environment states or agent actions.[19]

In this section we address the single-agent case. In an MDP we assume that in each state s_t at time t the agent receives from the environment an immediate reward or reinforcement $R(s_t) \in IR$. The task of the agent is to maximize its total discounted future reward $R(s_t) + \gamma R(s_{t+1}) + \gamma^2 R(s_{t+2}) + \_ \_ \_$, where $\gamma \in [0, 1]$ is a discount rate that ensures that even with infinite sequences the sum is finite. Clearly, the discounted future reward will depend on the particular policy of the agent, because different policies result in different paths in the state space.

Given the above, the optimal utility of a state s for a particular agent can be defined as the maximum discounted future reward this agent can receive in state s by following some policy:

$$U^*(s) = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right] \qquad \text{(Eqn 2-1)}$$

where the expectation operator E[.] averages over rewards and stochastic transitions. Similarly, we can define an optimal action value Q*(s, a) as the maximum discounted future reward the agent can receive after taking action a in state s. A policy π*(s) that maximizes the above expression is optimal policy. We should note that there can be many optimal policies in a given task, but they all share a unique U*(s) and Q*(s, a).

$$U^*(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U^*(s') \qquad \text{(Eqn. 2-2)}$$

This is called the **Bellman equation**, and the solutions of this set of equations (one for each state) define the optimal utility of each state. A similar recursive definition holds for action values:

$$Q^*(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} U^*(s',a') \qquad \text{(Eqn. 2-3)}$$

2.4.4.1 Value Iteration

A simple and effcient method for computing optimal utilities in an MDP when the transition model is available is value iteration. We start with random utility values

U(s) for each state and then iteratively apply (Eqn 2.2) turned into an assignment operation:

$$U(s) := R(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a)U(s') \qquad \text{(Eqn. 2-4)}$$

It is repeated until convergence which is measured in relative increase in *U(s)* between two successive update steps. Value iteration converges to the optimal *U*(s)* for each state.

2.4.4.2 Q-Learning

One of the disadvantages of value iteration is that it assumes knowledge of the transition model P(s´|s, a). However, in many applications the transition model is unavailable, and we would like to have a learning method that does not require a model. Q-learning is such a *model-free* method in which an agent repeatedly interacts with the environment and tries to estimate the optimal Q*(s, a) by trial-and-error. In particular, the agent starts with random estimates Q(s, a) for each state-action pair, and then begins exploring the environment. During exploration it receives tuples in the form (s, R, a, s´) where s is the current state, R is the current reward,a is an action taken in state s, and s´ is the resulting state after executing a. From each tuple, the agent updates its action value estimates as :

$$Q(s,a) := (1-\lambda)Q(s,a) + \lambda\left[R + \gamma \max_{a'} Q(s',a')\right] \qquad \text{(Eqn. 2-5)}$$

where λ ∈ (0, 1) is a learning rate that controls convergence. Note that the maximization in (Eqn 2.5) is over all actions a´ from the resulting state s´.

If all state-action pairs are visited infinitely often and λ decreases slowly with time, Q-learning can been shown to converge to the optimal Q*(s, a). Moreover, this holds irrespective of the particular exploration policy by which the agent selects its actions

above. A common choice is the so-called ϵ-greedy policy by which in state s the agent selects a random action with probability ϵ, and action a = arg $\max_{a'} Q(s; a')$ with probability 1- ϵ, where $\epsilon < 1$ is a small number. Alternatively, the agent can choose exploration action a in state s according to a Boltzmann distribution

$$p(a \mid s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'} \exp(Q(s,a')/\tau)} \qquad \text{(Eqn. 2-6)}$$

where τ controls the smoothness of the distribution (and thus the randomness of the choice), and is decreasing with time.

2.5 CEREBELLAR MODEL ARTICULATION CONTROLLER

Cereballar Model Articulation Controller (CMAC) was first described by Albus in 1975 as a simple model of the cortex of the cerebellum. [2] It is a biologically insprired learning method like neural networks and is generally used as a function approximator and state generalizer in RL problems.

[3] present a novel combination of CMACs and world models. CMACs use filters mapping sensor based inputs to a set of activated cells. Each filter partitions the input space into sub-sections in a prewired way such that each (possibly multi-dimensional) subsection is represented by exactly one discrete cell of the filter. For game playing, a filter may represent different but similar positions and the activated cell may represent the presence of a particular position. In a RL context each cell has a Q-value for each action. The Q-values of currently active cells are averaged to compute the overall Q-values required for action selection.

In principle filters may yield arbitrary divisions of the input space, such as hypercubes. To avoid the curse of dimensionality one may use hashing to group a random set of inputs into a equilanve class.

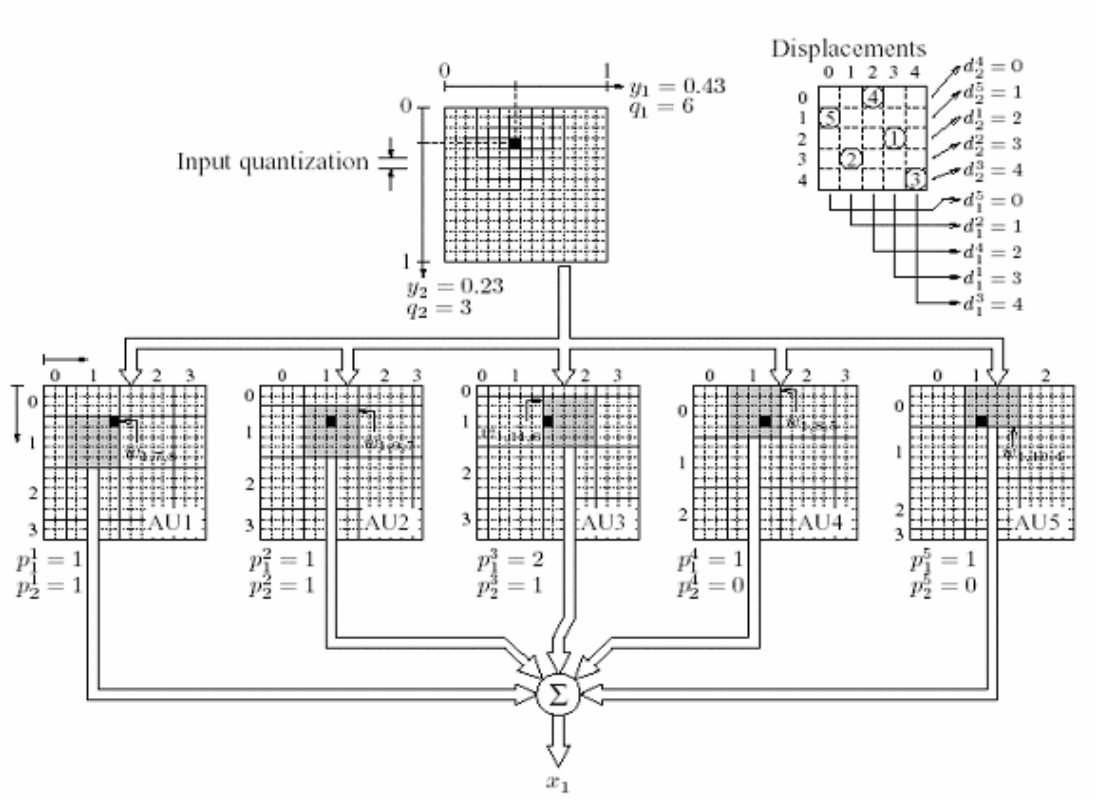


Figure 2-9 A two input CMAC (table form)

In Figure 2-9, there are 15 input quantization steps between [0, 1]. Any value between [0, 1] is mapped to an integer value between [0, 15]. The indexes (q_1, q_2) are used to look up weights in na two-dimensional lookup tables (na = 5 is the number of association neurons activated for any input). The number of AUs (association units shown in Figure 2-9) determines the generalization of a CMAC. The AU tables store one weight value in each cell, and cells are displaced along each axis by some constant. With this displacement every AU table in association layer gains ability to activate different neurons with respect each other. [20]

2.6 FEEDBACK FROM THE OTHER STUDIES

Ashley Tews and his team RoboRoos is a good example for potential field planner. Their system is based on the superposition of potential fields. In the paper [26], they are concentrated on multi-robot cooperation with potential fields. The planner which is centralised controller, examines the state of the game from the robot's perspective

and makes decisions as to that robot's next action and uses the potential fields to determine locations for the robots to carry out those actions.

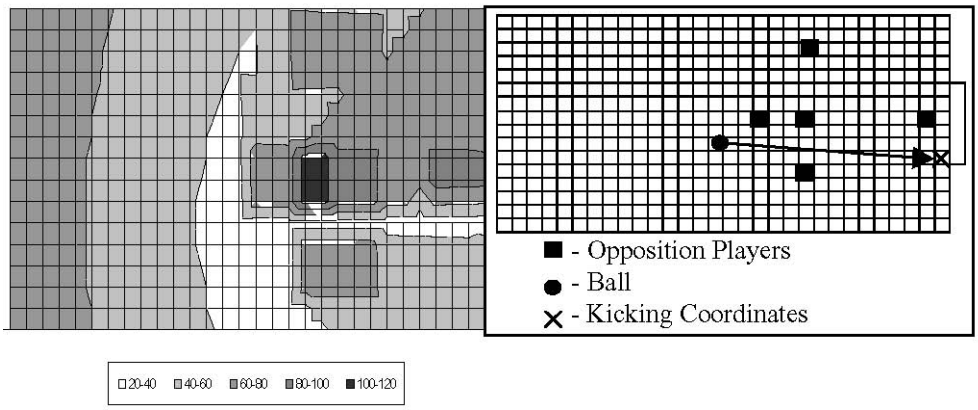


Figure 2-10 Contour plot example showing the kicking coordinate potential field

CS Freiburg [15] also used potential fields for the evaluation of the roles. Distinguishing between different areas of responsibility, 4 roles are created:

- *Active: which is in charge of dealing with the ball*
- *Strategic: which is in charge of securin the defense*
- *Support: which is in charge of supporting the team members in different areas considering the team situation, offense or defence.*
- *Goalkeeper: which is in charge of securing its goal.*

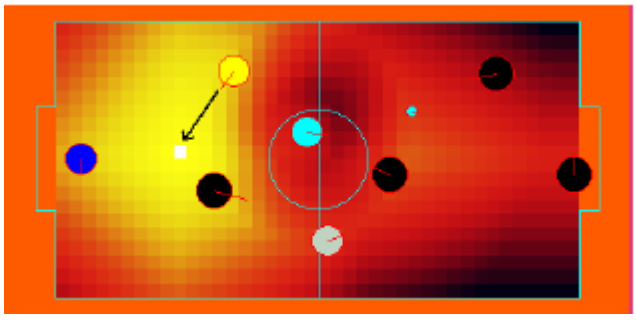


Figure 2-11 Potential field for determining the strategic position

Role distribution is similar to our study but we are not interested in which situation the tam is in.

The team members finds the most appropriate positions for each 3 role, active, strategic, support by using potential fields. After a field player has determined the best active, strategic, and support poses from its perspective, it estimates utilities for each role, which are based on the role itself and on an approximation of the time it would take the player to reach the corresponding preferred pose. The utility for a preferred pose p is calculated from the following constituents.

- Distance to the target position
- Angle necessary for the robot to turn toward the preferred position
- Objects between the player and the target position
- Angle necessary to turn the robot at the preferred position into the orientation of the preferred pose.

The total utility for the preferred pose p is computed as the the weighted sum of all criteria.

In order to decide which role to take, a player sends the utilities estimated for each role to its teammates and compares them with the received ones. Each player objective is to tale a role so that the sum of the utilities of all players is maximized.

RoboRoos and CS Freiburg find the best location to shoot, to defense and to suppport in the field using potential fields. They calculate every effecting potential field for every point in the field.

A similar study with CS Freiburg can be seen in [11]. It shows how heuristic bidding functions that use globally shared information may be used to determine which robot is the most suitable role for each task and it also describes how obstacle avoidance may be combined with coordination through the use of artificial potential fields.

$$Bid = \underbrace{\frac{\theta_{goal}}{\pi}}_{angular\ component} + \underbrace{(1 - min(1, d_{ball}))}_{distance\ component}$$

Figure 2-12 Bidding function for the primary attacker

The ClockWorkOrange[21] also uses utility functions for selecting the most appropriate role. Besides coordinating the team as a whole the Team skills module has a secomd task: selecting the next action for a robot which will benefit the team as a whole the most. In the paper, action selection can be defined as the problem of finding an optimal policy for mapping an agent's internal state to an action. Optimal is defined as maximizing a certain optimality criterion. Markov Decision Process is used in action selection algorithm.

In multi-agent Markov Decision Process, there are S states, N agent and A actions. The reward function R(s) determines receiving rewards of team of agents in state S. Each agent should choose its action to maximize the expected reward.

The policy is summarized in the paper as following:

Simulate each of the actions in set A on the current world state, evaluate the reward of the resulting world state, estimate the probability each action will succeed and calculate the utility of each action. The utility is the product of the reward and the probability of success. For each resulting world state we can repeat this process, updating the utility of the end state by multiplying the probabilities of success of the actions in the sequence leading to this end state with the reward of the end state. [4]

Hwang [22] uses Q-Learning for its cooperative strategy architecture. Temporal-difference has some advantages for solving reinforcement learning problems. TD does not require a model of the environment, of its reward and of next-state probability distributions and TD updates the estimate value just only waiting one time step, that is, it can learn without final outcome. Since accelerating the convergence rate and avoiding to a local optimum problems occured he develos its Adaptive Q-Learning algorithm The algorithm regulates the three parameters:

- learning rate
- discount rate
- temperature T in Boltzmann distribution

dynamically.

Initialize $Q(s,a)$ arbitrarily, and set TH_P, TH_N, C, β, and k

Repeat (for each episode):

Initialize s Repeat (for each step of episode):

Choose a from s using policy derived from Q

(use Boltzmann Distribution)

Take action a, observe r,

IF $TH\_N - kt < \max_a Q(s_{t+1}, a) < TH\_P + kt$

THEN $\gamma_t(\delta, \gamma_{t-1}) = \gamma_{initial}$ and $\alpha_t(\delta, t) = \alpha_{initial}$

ELSE

$\gamma_t(t, \gamma_{t-1}) = \tanh(t\gamma_{t-1})$

$\delta = r_{t+1} + \gamma_t(t, \gamma_{t-1}) \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$

$\alpha_t(\delta, t) = \tanh(\beta \mid \delta \mid / t)$

($\beta, k \in R_+$ and $\beta, k \geq 1$ and t is the number of trials.)

$T(t) = C/t$, $C \in N$

$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(\delta, t)\delta$

$s \leftarrow s'$;

until s is terminal.

Figure 2-13 Adaptive Q-Learning algorithm[22]

Q-Learning causes a lot of researchers to study on that subject because of its appropriateness to dynamic, complex and model free systems.

Q(λ) Learning is selected because of the eligibility traces utility. Tatlıdede [20] used Q(λ) Learning in his research. The state includes the relative distances between each player and the distances between each player and ball. So it has a big state space. Its system is a distributed sytem and each player decides its own action according to the state it senses. Different from him we used the oppurtunity of potential fileds and decrease the dimensionality of positional state variables, and can add more distinguishing variables like ball direction. Tatlıdede selects the basic actions like shoot, move, dribble as its action set, but we prefer to use roles like attack, defense and support.

Wiering,[3] Stone[4][23] are the ones which used CMAC as function approximater in their Reinforcement Learning Level.

CHAPTER 3

PROPOSED METHOD

3.1 SYSTEM REPRESENTATION

Soccer is a complex game where a team has to meet several requirements at the same time.The ball control is a hard and complex task, therefore it requires real talent and that is why the social community rewards soccer players. A good cooperative system includes dynamic role switching, good robot behaviors and formations.

Our system is acentralized system that learns to assign appropriate roles to the teammates according to the dynamically changing conditions. The environment is not under the control of any team, since you can not control your opponents. In most of the soccer matches the teams are aware of their opponent's strategies. So the experiences learnt while playing are extremely important.

We proposed a self-learning cooperative strategy for a robot soccer game. We select reinforcement learning because it suggests an unsupervised learning through trial and error. The sytem is free to try actions in its action set in real or simulated environment and receives rewards for all of its actions. The system's goal is to choose actions so as to maximize the expected sum of rewards over some time horizon.

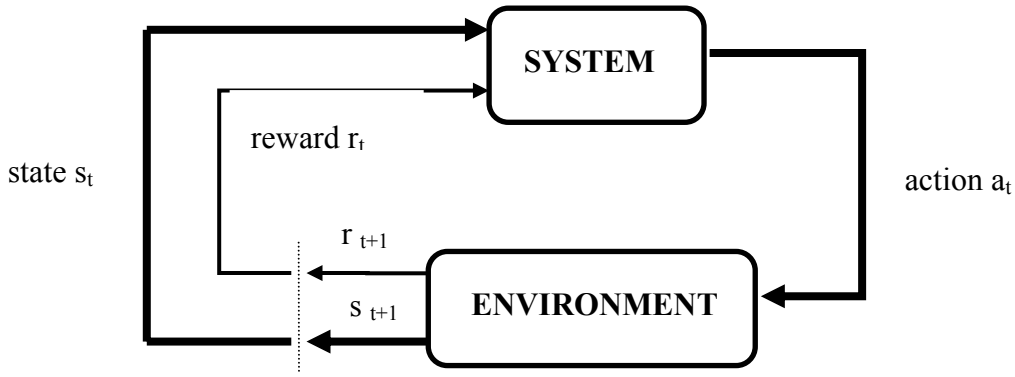


Figure 3-1 The agent-environment interaction

The soccer game environment is so dynamic that it is hard to track the fast changing environment. The sensing system should be powerful, the decision system should be fast and adaptive. To model the environment is hard, the system should learn what to do with the rewards and punishments it takes. Since the goal of the teams is to gain the matches by scoring more goals to the opponent goal, the rewards are the scores at the opponent's goal and the punishments are the scores at the home goal.

Since TD methods can learn directly from raw experience without a model of the environment's dynamics and update estimates based in part on other learned estimates, without waiting for a final outcome[24] we decide to use them in our learning level.

Learning an action-value function rather than a state-value function is more effective in our role assignment system. The aim is to select the most valuable action for the condition the system senses, so Q(s, a) values are used.

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] \qquad \text{(Eqn 3-1)}$$

where α is the learning rate and γ is the discount rate.

This update is done after every transition from a nonterminal state s_t.

It is straightforward to design an on-policy control algorithm based on the Sarsa prediction method. As in all on-policy methods, we continually estimate Q^{π} for the

behavior policy π, and at the same time change π toward greediness with respect to Q^{π}. [24]

We select Max-random exploration rule which selects the greedy action with probability P_{max} and selects a random action.

1. Generate a number ***ran*** from the uniform distribution [0, 1]
2. if ***ran*** $\leq P_{max}$
   2.1 select action with highest Q-value
   2.2 Else select random action a $\in \{A_1, A_2, ...A_n\}$

Figure 3-2 P_{max} Algorithm

The important issues in Q-learning is the problem of how to accelerate the convergence rate and how to avoid converging to a local optimum. In order to solve these issues, three parameters that can affect the performance of Sarsa-learning, which are learning rate, discount rate and exploration rate, are left variable since fixing the three parameters is not suitable for learning systems because systems will easily converge to a local optimum.

Regulating the three parameters dynamically is thus important.

For exploration purposes, it is efficient if the learning rate is taken a large value and the discount rate small in the initial stage of learning in order to ensure that systems can have more opportunities to search unknown knowledge.

During the learning phase P_{max} is increased steadily to decrease exploration rate. At the very beginning , the agent explores many actions in a given state.

We train our team during matches, each match is a trial or an episode as defined in Figure 3-3. Not only the action and state pair before the scoring of a goal is important, but we need to consider also the previous pairs that bring us reward.

Since Sarsa-learning algorithm only use one-step data, to take the effect of the previous pairs by using eligibility traces which are used to keep track of all the actions taken by the agent to reach a terminal state. The trace marks the memory

parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error.

The idea in Sarsa(λ) is to apply the TD(λ) prediction method to state-action pairs rather than to states.

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha\delta_t e_t(s, a) \qquad \text{(Eqn 3-2)}$$

where

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t), \qquad \text{(Eqn 3-3)}$$

For all *s, a*

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + 1 \qquad \text{if } s = s_t \text{ and } a = a_t; \qquad \text{(Eqn 3-4)}$$

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) \qquad \text{otherwise.} \qquad \text{(Eqn 3-5)}$$

Initialize Q(s, a) arbitrarily and e(s, a) = 0, for all s,a

Repeat (for each episode):

Initialize s, a

Repeat (for each step of episode):

Take action a, observe r, s′

Choose a′ from s′ using policy derived from Q

$\delta \leftarrow r + \gamma Q(s' ,a') – Q(s, a)$

$e(s, a) \leftarrow e(s, a) + 1$

For all s, a:

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$

$e(s, a) \leftarrow \gamma \lambda e(s, a)$

$s \leftarrow s’ ; a \leftarrow a’$

until s is terminal

Figure 3-3 Tabular Sarsa(λ)

3.2 ACTION REPRESENTATION

Soccer is a complex game where a team usually has to meet several requirements at the same time. To ensure that in any game situation a team is prepared to defend its own goal, but also ready to attack the opponent goal, the various team players have to carry out different tasks, and need to position themselves at appropriate strategic positions on the field. [15]

A team strategy is the distribution of certain roles over the available field players. We define the roles as our actions of learning system. A role consists of a specification of an agent’s internal and external behaviors.[25]

- ***Attacker:*** is in charge of dealing with the ball. Its aim is approaching the ball and bringing the ball forward to the opponent goal. Its role is important because it provides to take rewards. It should position the ball between the opponent goal and itself.
- ***Defencer:*** is in charge of securing the defense. Its aim is preventing the ball entering its own goal. It finds the most suitable position between the ball and its goal without interrupting the goalkeeper. Its role is important because it avoids to take punishments.
- ***Supporter:*** is in charge of supporting the other two players. Its aim is to stay ready to be assigned the appropriate role, attacker or defencer. It also helps the other teammates by preventing the opponent players actions.
- ***Goalkeeper:*** is in charge of securing its own goal. It stays in its goal area and moves depending on the ball's position and direction.

3.3 STATES REPRESENTATION

Since RL maps states to actions, one of the main issues of an RL application is modeling the state space. In soccer game state variables are numerous and the positional states are also continuous. The continuity in state variables is handled by quantization. Combining similar varibles decreases the number of state variables. Thus we are able to eliminate some of the state variables. CMACs method are used for primarily function approximation but also by quantization capability CMACs makes RL possible to applicable for large tasks such as soccer game. [23]

Soccer game has a dynamic environment containing a ball, two goals and players. The coordinates of these objects in the environment are important informations for the system to coordinate its team members. Some researchers [25] use these items' relative distances to agent, composed of distance r and angle θ between the normal line and the agent as state variables. With increasing number of agents, state space gets bigger and bigger and makes computation harder. So we select to use the opportunity of potential field methods.

Although we try to decrease the dimension of state vector with using resultant potential field, we can not say that we reach an reasonable dimension. Since we can not meet most of the states, CMAC value approximation method is used.

3.3.1 OUR POTENTIAL FIELD STUDIES

In our studies we used potential fields both as state variables of our reinforcement learning system and for the computation of our actions.

This allows us to benefit from the speed and flexibility of potential fields. The possible impulsive and attractive potential forces on a soccer field may be similar to the fields in Figure 3-4. [16]

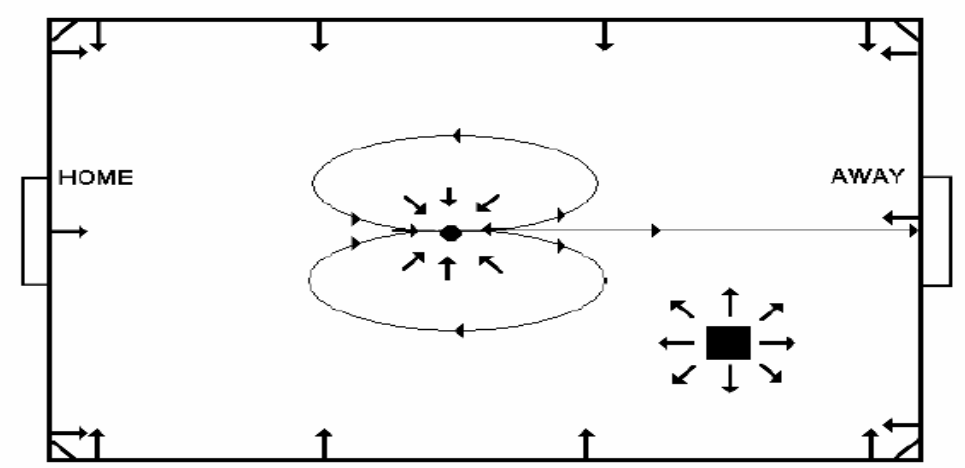


Figure 3-4 Attractive and Impulsive Forces

We select the equation below for the computation of our potential fields.

$$U = \alpha * \exp^{-\left(\frac{(x-x_0)^2}{a}+\frac{(y-y_0)^2}{b}\right)} \qquad \text{(Eqn. 3.2)}$$

The benefit of using exponential equation is the easy implementation of *variance*, or the sharpness the potential field.

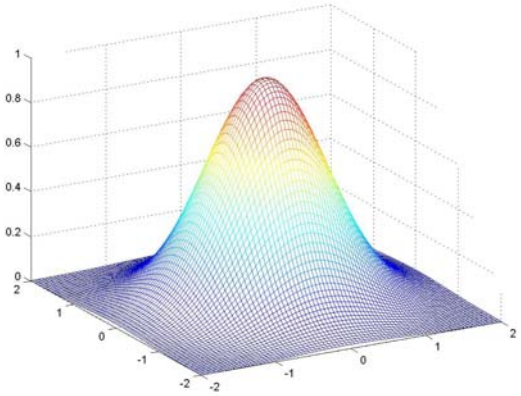


Figure 3-5 Figure of Equation 3-2 where α =1, a = 1, b = 1, x_0 = 0, y_0 = 0

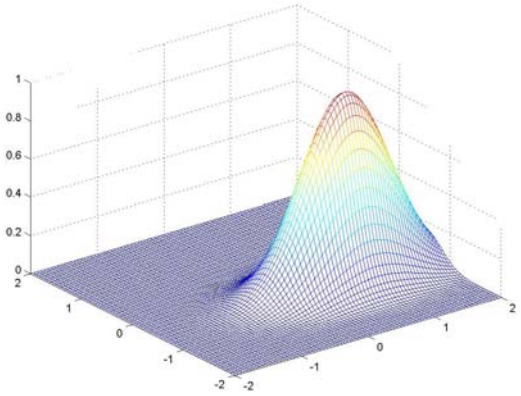


Figure 3-6 Figure of Equation 3-2 where α =1, a = 0.8, b = 0.2, x_0 = 0.8, y_0 = -0.5

As state variables, for each three team member we take the magnitudes of resultant potential field exerted on each member, and the angle of the vector which directs each member in the potential environment. For the coordination of team behaviors in the learning level , the vectors of the ball, opponents and teammates take care. Ball has the most influencing field, and it is attractive effect on the member so it takes a negative value. The opponent has an repulsive effect to prevent collisions and to find clear path to the goal. But it has the least influencing field, because the members should also block the opponents, should go near to the opponents to take ball in case. The teammate has also an impulsive effect to prevent collisions and it has an bigger

influence than opponent field because team members should not block their team members and should help them to have a clear path.

- **BALL FIELD:**

$$U = -\alpha_{ball} * \exp^{-\left(\frac{(x-x_{ball})^2}{a_{ball}} + \frac{(y-y_{ball})^2}{b_{ball}}\right)} \quad \text{(Eqn. 3-3)}$$

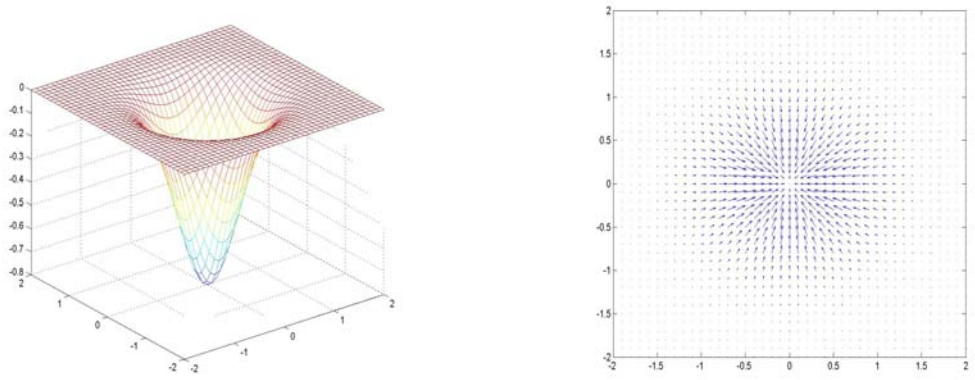


Figure 3-7 Figure of Potential Field and Vector Field of Ball Field

- **TEAMMATE FIELD:**

$$U = \alpha_{teammate} * \exp^{-\left(\frac{(x-x_{teammate})^2}{a_{teammate}} + \frac{(y-y_{teammate})^2}{b_{teammate}}\right)} \quad \text{(Eqn. 3-4)}$$

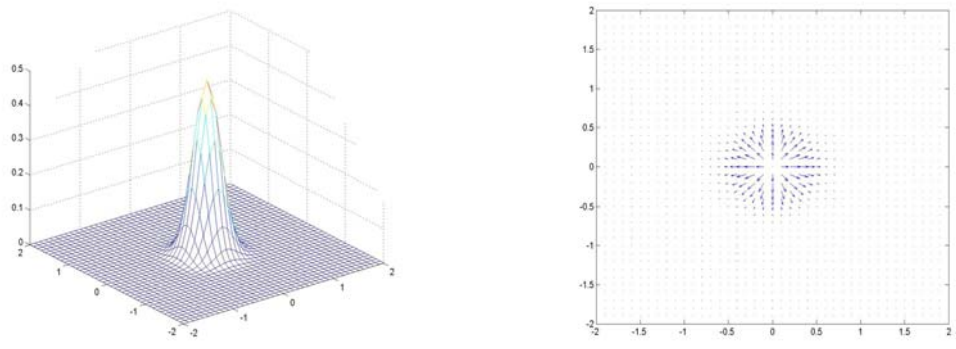


Figure 3-8 Figure of Potential Field and Vector Field of Teammate Field

- **OPPONENT FIELD:**

$$U = \alpha_{opponent} * \exp^{-\left(\frac{(x-x_{opponent})^2}{a_{opponent}} + \frac{(y-y_{opponent})^2}{b_{opponent}}\right)} \quad \text{(Eqn. 3-5)}$$

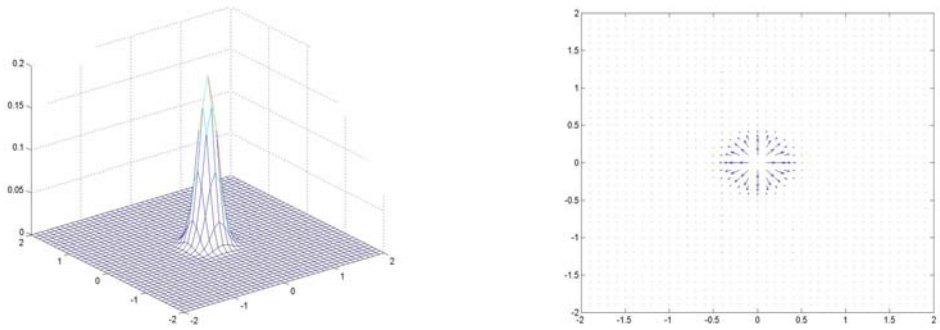


Figure 3-9 Figure of Potential Field and Vector Field of Opponent Field

In Figure 3-10, graphical illustration of the one ball positioned at (0,0) point, one opponent positioned at (1,1) point, one teammate positioned at (1,-1) point is shown.

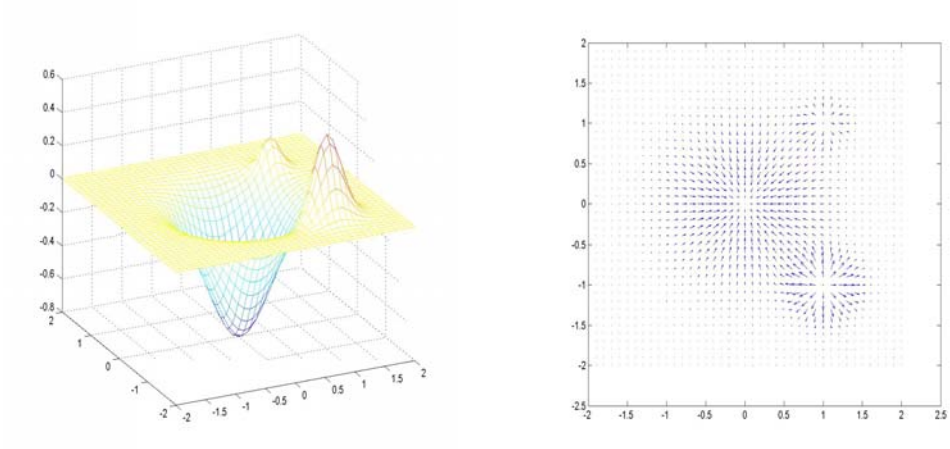


Figure 3-10 Figure of Potential Field and Vector Field of Opponent Field of one ball, one teammate and opponent in the area

Since these state values are continuos, we should quantize them to make differences between states.

3.3.2 MEMBER'S ZONE

The other state variable is the zones the team member is occupied in in that time interval.

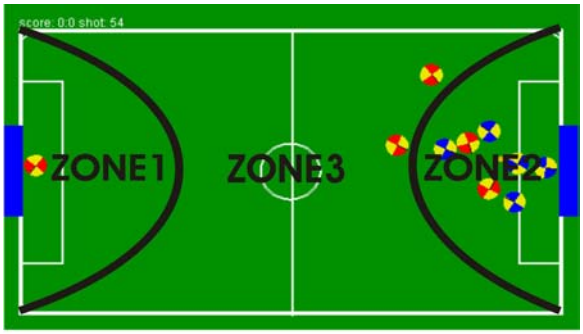


Figure 3-11 Environment Divided Into 3 Zone s

3.3.3 BALL DIRECTION

The last state variable is the ball direction. It gives knowledge about in which region the ball moves. The ball driection state is important for our sytem and it is one of the difference from the other systems. The detailed computation method is explained in 4.3.3. The state is quantized as quadrants. The region which the ball enters gives the information if the ball is approaching our goal or the opponent's goal.

3.4 REWARD REPRESENTATION

The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. The agents always learns to maximize its reward, so that it becomes a crucial input for a system. A reward signal is a prior knowledge to an agent about how to achieve what we want it to do. The reward signal is our way of communicating to the system what we want it to achieve, not how we want it achieved. [24]

Since many reward finding have been already defined, its formalization remains a crucial issue. For example, if we define the zone which the ball occupies as the reward signal and take as a positive reward the zone which it occupies near to the opponent's goal, our system could make the mistake of dribbling the ball to the defined zone but forgetting to score by shooting the opponent's goal or forgetting to defence its own goal. Because in reinforcement learning, the only aim of the system is to maximize its reward value. So dribbling the ball in that defined zone would be this maximization.

As mentioned before , in soccer game our aim is to beat the opponent team by scoring more goals to their goal. So any score in the opponent's goal provides us reward , r = 1, where r is the reward signal. And any score in our goal gives us punishment, r = -1. In any other case the system takes r signal as r = 0.

CHAPTER 4

EXPERIMENTS AND RESULTS

Teambots simulation program is selected as the testbed for our experiments. It provides a a distributed controlled system. Teambots agents has the basic abilities such as

- *Move(x,y,v):* Move to the point(x,y) with velocity v,
- *Turn(θ):* Turn its heading as defined angle θ;
- *Kick:* Kick the ball if canKick situation is available,
- *Sense:* Senses ball, opponents and teammates,

They also have communication property, they can take the directives sent and also informs the teammates about the objects he senses.

Our team has 4 members. Each member is in charge of learning except one which is assigned as goalkeeper. Its role is not only to protect his goal but also to inform the roles assigned in the role assignment level to its teammates by communicating with them.

Teambots provide distributed system based teams, but in our method we construct a centralised control system and manage this by giving the role assignment task to the goal-keeper. After constructing a communication system, we can get the information about the coordinates of the robots and the ball in the environment for using in our system. The collection of informatin, learning and role-assignment level is worked at

the interval which is seperated for the member which we assigned as goal-keeper. So we can define it as role assigner.

The Q values are stored in files which are defined as CMAC files, which are saved as dat file.[38] At the very beginning of the program it is controlled that if these are created before or not. If they are not created, it means that it is the beginning of the learning and the sytem knows nothing. If they are created, it means that learning is done before and the system knows as much as the storage of files.

The number of CMAC files is the number of our actions. Their aim is the storage of Q(s, a) values. Since our state space is so big we use the coarse coding algorithm for the generalization between them, hashing function to store the Q(s, a) values in a smaller space.

We use 4 tiles, and our storage matrices' dimension is 5000. At the beginning of the learning the matrices which contain the Q(s, a) and e(s,a) values are zero. For each state 4 indexes are created by the function ***StateToActiveIndexes*** , for the selected action a the CMAC(a) file is opened and the changing values are stored in the 4 indexes of the matrices of the CMAC (a) fie.

Each match is defined as one episode or trial as defined in Sarsa (λ). In Figure 4-1 the total algorithm for each step is shown. In Figure 4-2 the ***Role Assignment and Learning Level*** is shown in detail and in Figure 4-3 ***State Assignment*** is shown in detail.

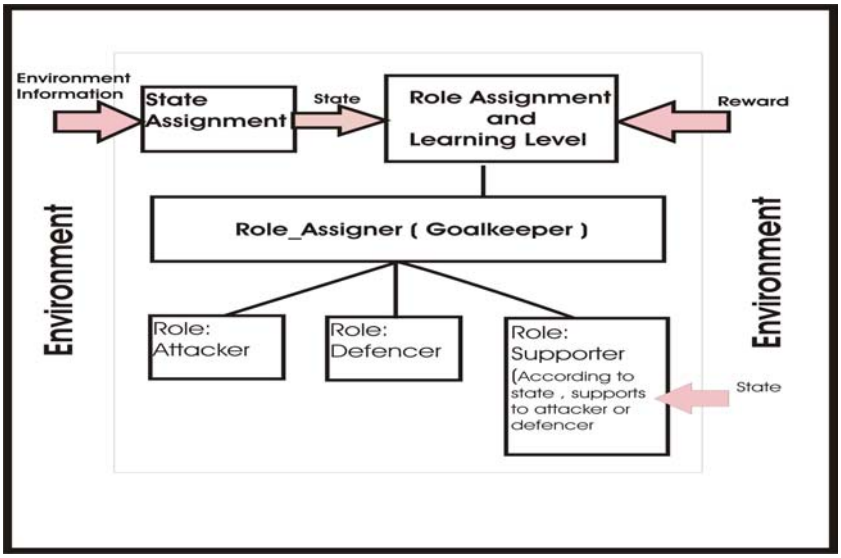


Figure 4-1 Cooperative Sytem Architecture

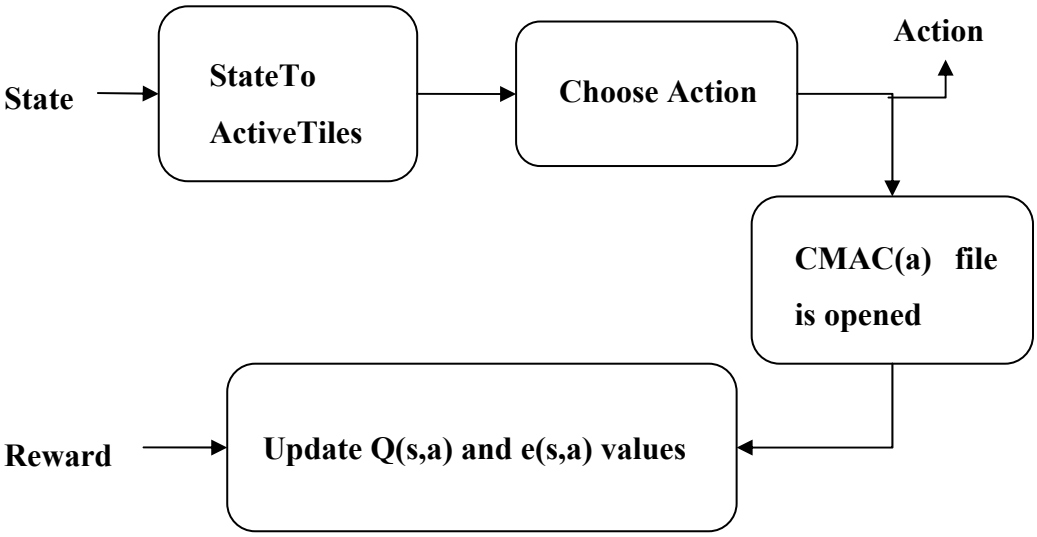


Figure 4-2 Role Assignment and Learning Level Block Diagram

- **StateToActiveTiles:** 2 dimensional hashing and tile coding diagram is shown in Figure 2.9. Our state space is 10 dimensional. After state assignment, a 10 dimensional matrix is entered to StateToActiveTiles program. 4 tiles are used in our sytem. 4 indexes between 0-4999 are created for the entering state matrix. After this step these 4 indexes are used during the learning level representing the state matrix.

- **Choose Action:** Max-random exploration rule is used as shown in Figure 3-2.
- **CMAC(a) file is opened:** For 6 actions 6 CMAC file is available. For the selected action a, CMAC(a) is opened and the storage values are presented to be worked on.
- **Update Q(s,a) and e(s,a) values:** Sarsa(λ) algorithms are used as shown in Figure 3-3. λ is selected as 0.9. Learning rate is 1 at the beginning and at each trial the learning rate is multiplied with decaylearning rate which is 0.98.

4.1 STATE ASSIGNMENT

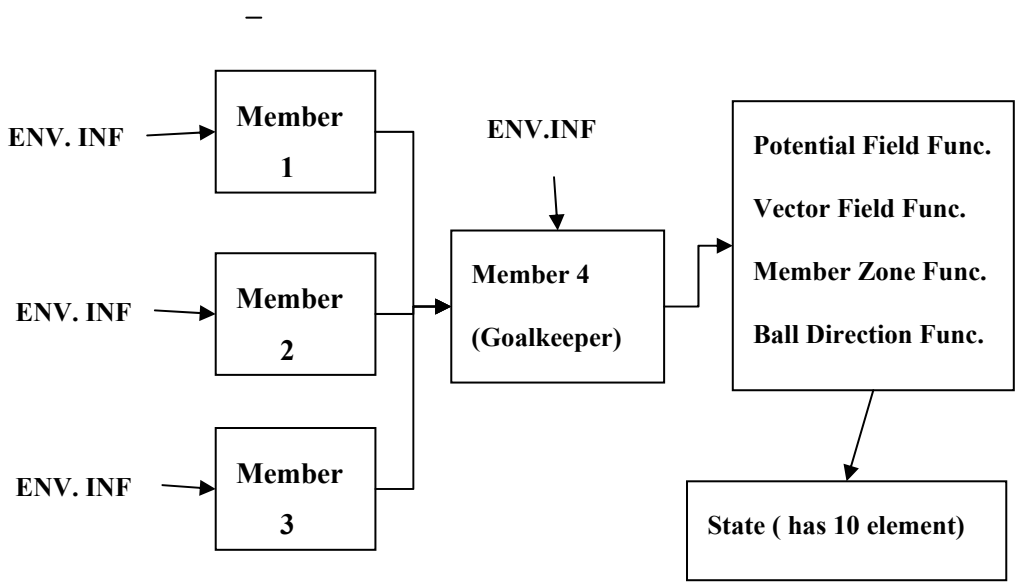


Figure 4-3 State Assignment Block Diagram

ENV. INF represents the environmental information which includes ball position, teammembers' positions, opponents' positions sensed by the teammembers. The total information is collected by goalkeeper and is used in the state equations. State space has 10 elements as shown in Table 4-1.

Table 4-1 State Elements

States	Definitions	# of Variables
S[1]	The resultant potential field magnitude effected on player with ID=1	3
S[2]	The resultant vector field angle effected on player with ID=1	4
S[3]	The resultant potential field magnitude effected on player with ID=2	3
S[4]	The resultant vector field angle effected on player with ID=2	4
S[5]	The resultant potential field magnitude effected on player with ID=3	3
S[6]	The resultant vector field angle effected on player with ID=3	4
S[7]	The zone which player with ID=1 occupied.	3
S[8]	The zone which player with ID=2 occupied.	3
S[9]	The zone which player with ID=3 occupied.	3
S[10]	The ball direction	4

4.1.1 POTENTIAL AND VECTOR FIELD FUNCTIONS

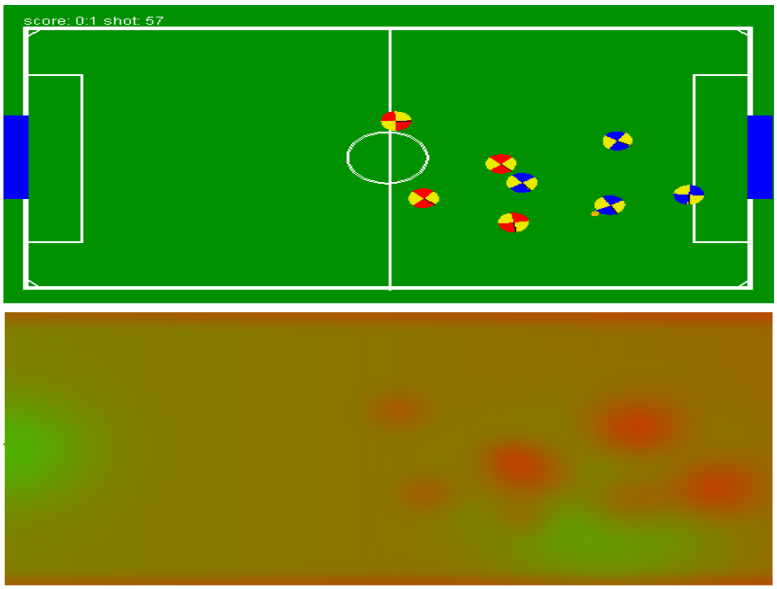


Figure 4-4TeamBots Simulation Program andPotential Field Shown Simultaneously

Since Teambots is an open source program, it lets the users change everything. For the visual following of potential fields, and the dynamically changing environment we add the potential field window by changing java file named SimulationCanvas, as shown in Figure 4-4. This potential field simulation provides us a simultaneous vision inspection of our system. Green areas represent the least values of potential fields which are the attractive fields such as ball field, opponent goal field. Red areas represent the most values of potential fields which are the impulsive fiels such as wall field, opponent field, teammate field.

The potential fields in our state determination include the ball field as attractive, and teammates and opponents fields as repulsive. The ball has the most effective field because it is important to know which member is in the ball field valley. Ball field has bigger area effect than the teammate and opponent field.

In Matlab, a GUI program is written to show the potentials fields and vector fields graphically as shown in Figure 4-5, 4-6 and 4-7. This supplies us to examine our potential fields in different occupations of the objects, find the most appropriate values of variables..

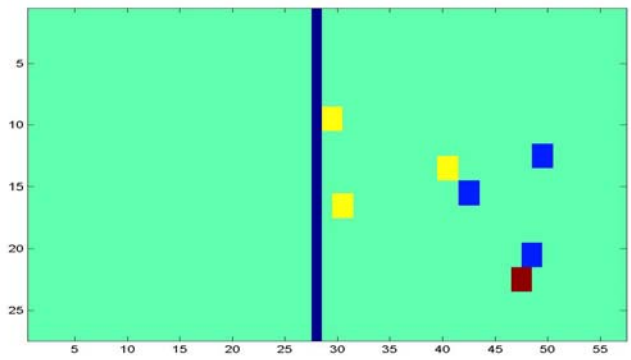


Figure 4-5 GUI Representing Objects In Figure 4-5

Yellow ones are opponents, of blue ones are teammates and the red one is the ball.

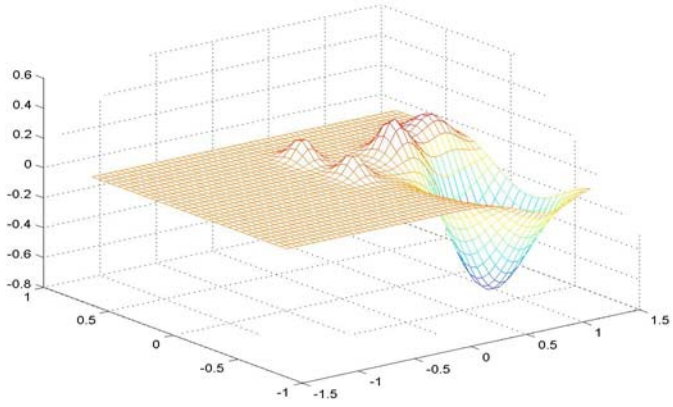


Figure 4-6Potential Fields SoccerField.

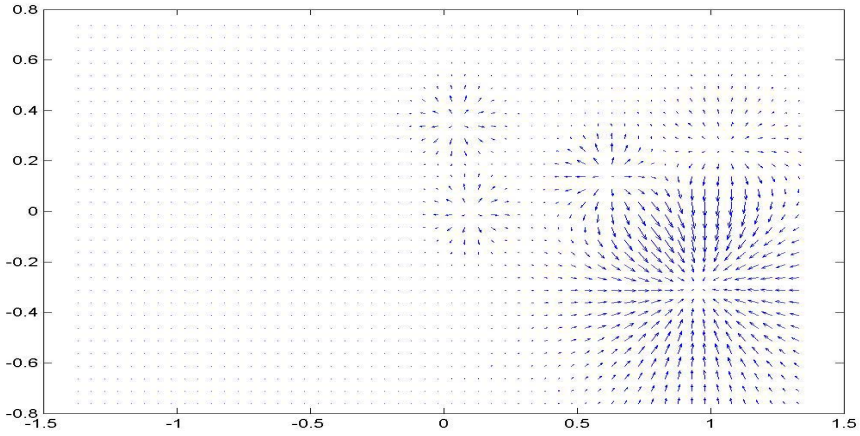


Figure 4-7Vector Fields of Objects In Figure 3-5.

4.1.2 POTENTIAL FIELD MAGNITUDE AND QUANTIZATION

- **BALL FIELD:** Ball field is the most effecting attractive field. It should get the biggest amplitude value. Besides amplitude, in large areas its influence should be noticed, so it should get also higher variance values than the other two repulsive field which are used in our states. By using our GUI program we find the most appropriate values for α_{ball} as 0.8, a_{ball} as 0.5 and b_{ball} as 0.5.

$$U = -0.8 * \exp^{-\left(\frac{(x-x_{ball})^2}{0.5}+\frac{(y-y_{ball})^2}{0.5}\right)} \quad \text{(Eqn. 4-1)}$$

- **TEAMMATE FIELD:** Teammate field is one of the repulsive field of our state representation sytem. It has a smaller absolute amplitude value than the ball field has but a higher absolute amplitude value than the opponent field has. The players should take care of colliding with their teammates. By using our GUI program we find the most appropriate values for $\alpha_{teammate}$ as 0.4, $a_{teammate}$ as 0.1 and $b_{teammate}$ as 0.1.

$$U = 0.4 * \exp^{-\left(\frac{(x-x_{teammate})^2}{0.1}+\frac{(y-y_{teammate})^2}{0.1}\right)} \quad \text{(Eqn. 4-2)}$$

- **OPPONENT FIELD:** Opponent field is one of the repulsive field of our state representation system. The players should take care of colliding with the opponents. But opponent field has the smallest absolute amplitude value and the smallest variance value since if the opponents have the ball, our players should not go away and moreover should block the opponents. By using our GUI program we find the most appropriate values for $\alpha_{teammate}$ as 0.2, $a_{teammate}$ as 0.05 and $b_{teammate}$ as 0.05.

$$U = 0.2 * \exp^{-\left(\frac{(x-x_{opponent})^2}{0.05}+\frac{(y-y_{opponent})^2}{0.05}\right)} \quad \text{(Eqn. 4-3)}$$

The resultant potential field is quantized to make a more discrete meaning to the learning level. The potential field is quantized as:

- ***Valley:*** If the value is smaller than -0.2 and quantization value is 1,
- ***Smooth****:* If the value is between -0.2 and 0.2 and quantization value is 1,

- ***Hill:*** If the value is bigger than 0.2 and and quantization value is 1.

4.1.3 VECTOR FIELD ANGLE AND QUANTIZATION

Vector Field is normally the minus gradient of the potentail field.

$$-\frac{d(U)}{d(x)} = -\alpha_{ball} * (-\frac{2*(x - x_{ball})}{a_{ball}}) * \exp^{-\left(\frac{(x-x_{ball})^2}{a_{ball}} + \frac{(y-y_{ball})^2}{b_{ball}}\right)} * d(x) \qquad \text{(Eqn 4-4)}$$

$$-\frac{d(U)}{d(y)} = -\alpha_{ball} * (-\frac{2*(y - y_{ball})}{b_{ball}}) * \exp^{-\left(\frac{(x-x_{ball})^2}{a_{ball}} + \frac{(y-y_{ball})^2}{b_{ball}}\right)} * d(y) \qquad \text{(Eqn 4-5)}$$

In our calculations, when the components are so near to the member which the potentail field is calculated for since (x - x_{ball}) goes to zero, it prevents us to calculate the exact values.

So we first find the vectors between components and members under calculation, and do vectoral addition. The angle of the vectors is the angle between component and member, and the magnitude is the potential field magnitude.

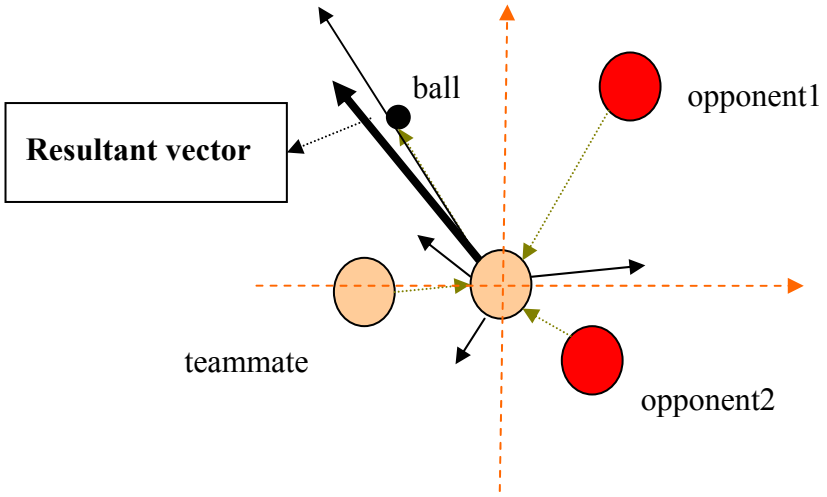


Figure 4-8 Vector Field Calculation

The resultant angle is quantized as quadrants.

4.1.4 MEMBER ZONE

We calculate the zone the member is in with our potential field equation. For the left of the area, Eqn 4-6 is used, if the member is in this area the value of U_1 will be greater than 0.3 and the state value for the member will be 1. If the member is in right of the ares, the value of the U_2 will be greater than 0.3 and the state value for the member will be 3. Otherwise the state value will be 3, which means the member is occupied in the middle area.

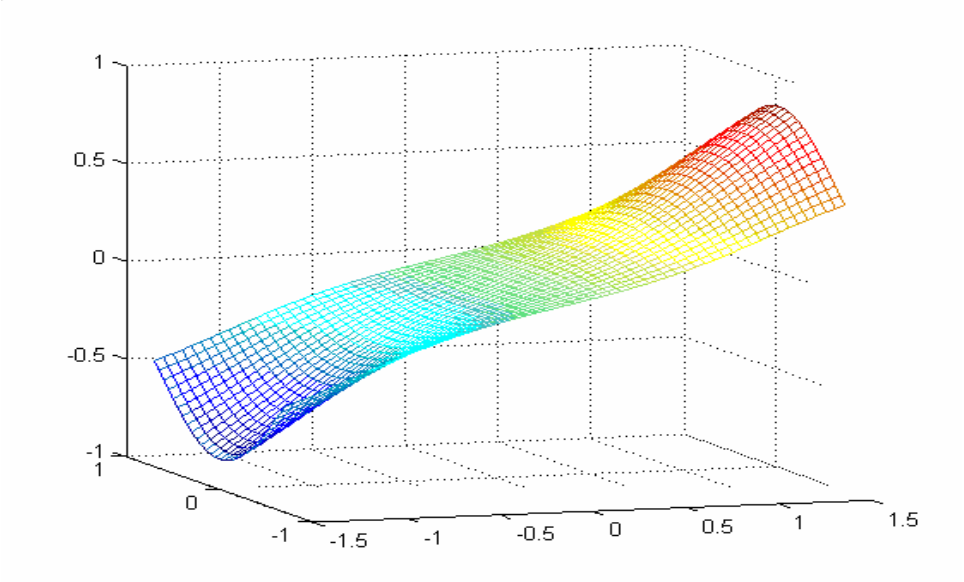


Figure 4-9 Figure of Potentail Field of Soccer Field

$$U_1 = 1 * \exp^{-\left(\frac{(x-x_{left})^2}{1.5}+\frac{(y-y_{center})^2}{0.7}\right)} \quad \text{(Eqn 4-6)}$$

$$U_2 = 1 * \exp^{-\left(\frac{(x-x_{right})^2}{1.5}+\frac{(y-y_{center})^2}{0.7}\right)} \quad \text{(Eqn 4–7)}$$

4.1.5 BALL DIRECTION

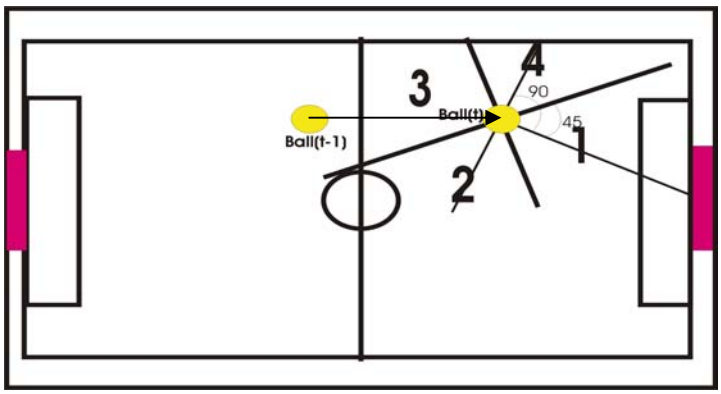


Figure 4-10 Ball direction

Computation of ball direction: At time t, a vector between ball and the center of the goal is created. A vector which is intersecting this vector with 90^o is drawn. The regions in which the ball is entering is computed by drawing two orthogonal vectors intersecting previous ones with 45^o as shown in Figure 4-10. And a vector between the coordinates of the ball at time and at time (t-1) is drawn. The state value is the region the ball direction enters.

4.2 ACTIONS

Action vector has 3 variables, which are the roles of our teammates. Our system assigns different roles to each 3 members:

- *Attacker*
- *Defencer*
- *Supporter*

Goalkeeper is fixed. It is no use in the learning but its duty is to communicate with the teammates to inform their roles.

Table 4-2 Actions

Actions	Role for Teammate 1	Role for Teammate 2	Role for Teammate 3
1	Attacker	Defencer	Supporter
2	Attacker	Supporter	Defencer
3	Defencer	Attacker	Supporter
4	Defencer	Supporter	Attacker
5	Supporter	Attacker	Defencer
6	Supporter	Defencer	Attacker

After role assignment, the algorithm shown below is run in the program.

```
if (robot=GOALIE) then
        apply Goalie Vector
else
        apply Wall Vector, Opponent Vector, Team Vector
        if ( robot = Attacker)
            apply Ball Vector1, Ball Vector2
        else if ( robot = Defencer)
            apply Defense Vector
        else if ( robot = Supporter)
            apply Supporter Vector
```

Figure 4-11 Algorithm of Actions

After assigning roles to each member, they should know what to do according to their role. Each spesific role is in influence of specific vectors bring them to the appropriate positions.

The vector calculations are same as we explained in part 4.1.3. The vector angle is the angle between the component and member and the magnitude is the potential field magnitude.

1. **WALL FIELD Vector**: It is a repulsive vector, cause the players stay a little bit far from themselves.

$$U_1 = 0.1 * \exp^{-\left(\frac{(y - y_{up})^2}{0.02}\right)} \quad \text{(Eqn 4-6)}$$

$$U_2 = 0.1 * \exp^{-\left(\frac{(y - y_{bottom})^2}{0.02}\right)} \quad \text{(Eqn 4-7)}$$

$$U_{wall} = U_1 + U_2 \quad \text{(Eqn 4-8)}$$

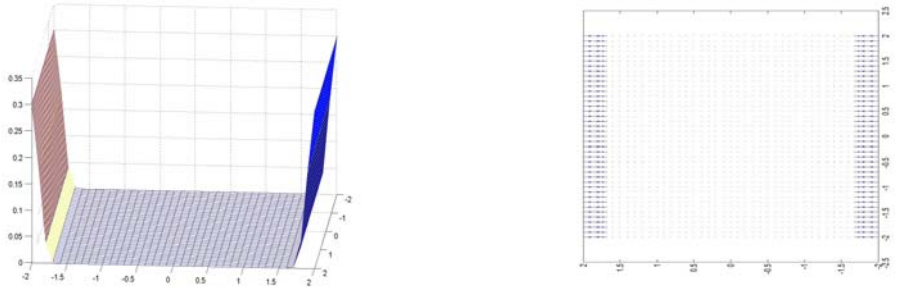


Figure 4-12 Potential Field and Vector Field of Wall

2. **OPPONENT FIELD Vector:** : It is a repulsive vector, causing the players stay far from opponents to prevent collisions. It is explained in part 4.1.3
3. **TEAM FIELD Vector:** : It is a repulsive vector, causing the players stay far from teammates to prevent collisions. It is explained in part 4.1.3

4. **BALL FIELD Vector 1 :** : It is a attractive vector, directing the players to the ball. It is explained in part 4.1.3

5. **BALL FIELD Vector 2:** We need this vector because the attacker should place between the ball and his own goal. Otherwise scoring to his own goal can happen. The member should be affected by a vector which takes him behind the ball around a circle as shown in Figure 3-13. The method and details is taken from reference[16].

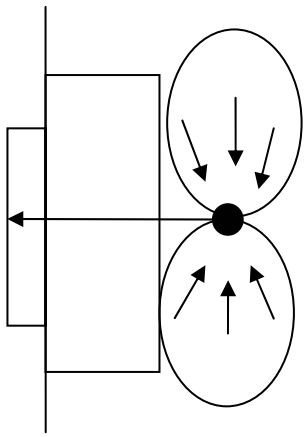


Figure 4-13 The resultant vector of Ball Field Vector 1 and Ball Field Vector 2

6. **GOALIE FIELD Vector:** The goalie should stand between the goal and the ball. We generate a potential attractive field for goalie at a point according to Equation 4.9. The direction of an attractive force is towards the source of the field.

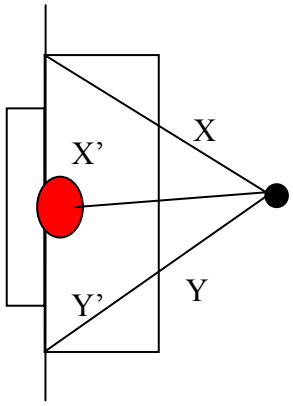


Figure 4-14 Goalie Field Vector

$$\frac{X}{Y} = \frac{X'}{Y'} \qquad \text{(Eqn. 4-9)}$$

7. **DEFENSE FIELD Vector:** The defense players try to stand in the middle of the ball and a point assigned on the goal line.

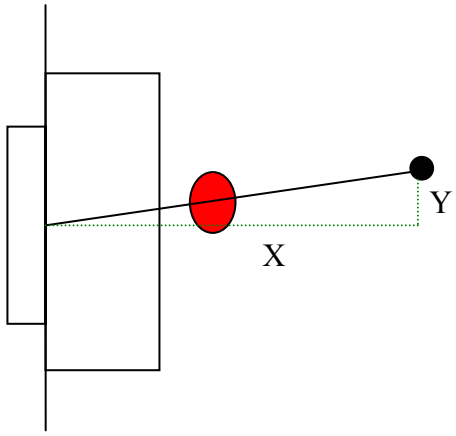


Figure 4-15 Defence Field Vector

$$X_{defencer} = \frac{X}{2} \qquad \text{(Eqn 4-10)}$$

$$Y_{defencer} = \frac{Y}{2} \qquad \text{(Eqn 4-11)}$$

8. **SUPPORTER FIELD Vector:** The last attractive field is supporter field, which creates fields behind the ball according to Equation 4-12 and 4-13. The parameter m indicates points on a circle with radius k and the ball at its center.

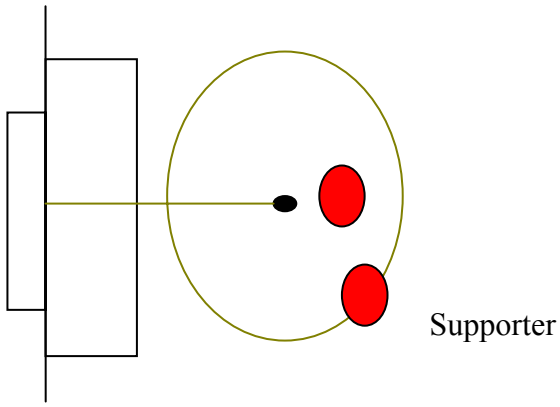


Figure 4-16 Supporter Field Vector

$$x = x_{ball} - k \times \cos(m) \qquad \text{(Eqn 4-12)}$$

$$y = y_{ball} \pm k \times \sin(m) \qquad \text{(Eqn 4-13)}$$

The parameter *m* indicates two points on a circele with radius k and ball at its center.

4.3 TRAINING EXPERIMENTS and PERFORMANCE ANALYSIS

We select 3 teams which are wriiten in Teambots simulaton program. They have different capabilities and different coopreataion systems.

We trained our team with making 600 trials for each team. The properties of the teams and the match results and performance analysis is shown in the sub-chapters.

After the performance analysis of 3 demo teams, we need a team which is as strong as us, or as weak as us. We create a new team which has members with the same abilities with our members, they can take the same roles and also are affected by the potential fields. The game strategy of the new team is the same strategy written in Figure 4-11. However, since this team has not the learning part , the role assignment is as follows:

```
If I am closest to ball
        I am attacker
Else If  I I am closest to home
        I am defender
Else I am closest to position1
        I am supporter
```

Figure 4-17 Role Assignment of the new Team

4.3.1 Training Experiments with Team GoToBall

Gotoball is a team with team members only going after the ball. The first one reaching the ball kicks the ball and then the other 3 one follows him in a line.

The results of 50 matches against GoToBall before learning and after learning are shown in Figure 4-18 and Figure 4-19.

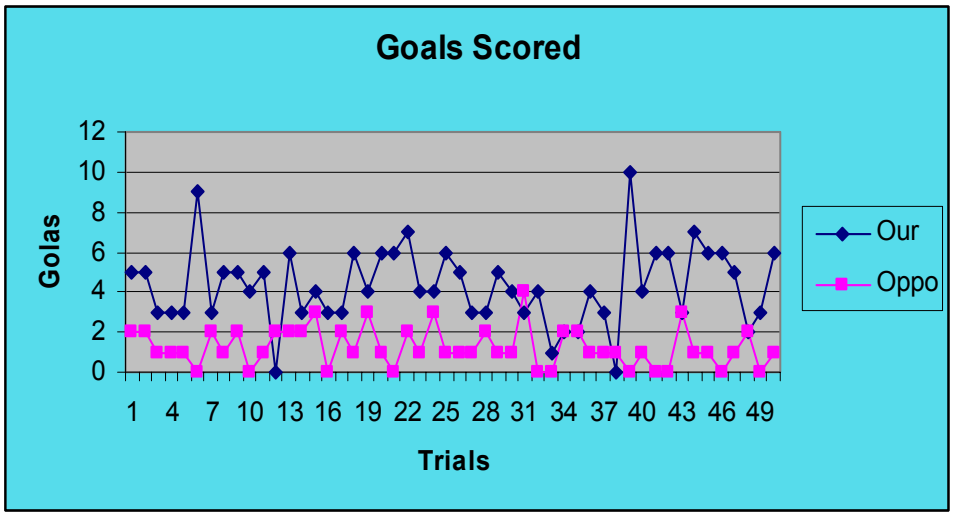


Figure 4-18 Goals Scored by GoToBall and Our team before learning

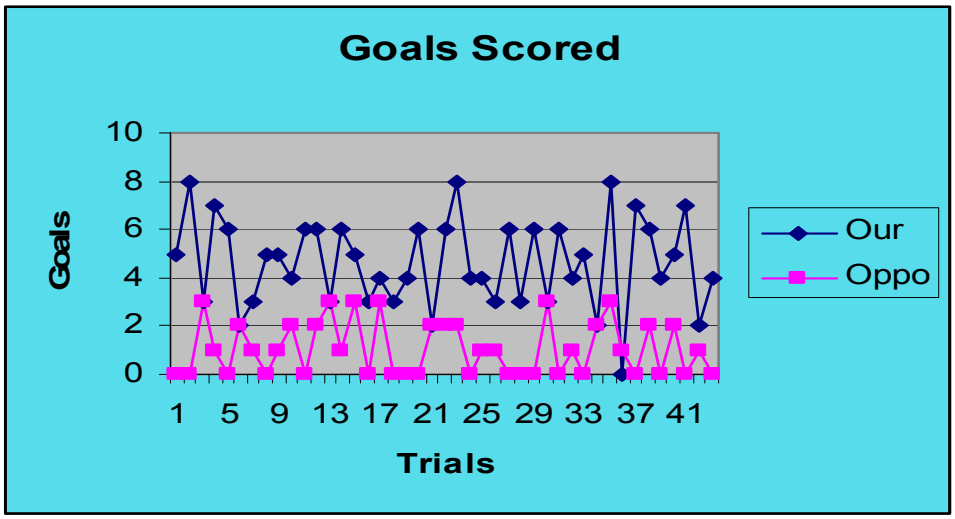


Figure 4-19 Goals Scored by GoToBall and Our team after learning

4.3.1.1 Performance Analysis

Training with them make the attacker gain abilities but this team has the disadvantage of shooting to his own goal. Not all the goals shown in Figure 4-14 and Figure 4-15 are scored by our team. Besides gaining attacker abilities, it has disadvantages that the real values can not be given to the state action pairs. In a non-appropriate state that they shoot to their own goal, we rewarded our team wrongly. But this is not the fault of the system. Since reinforcement learning is the learning by trial-error and the team learns doing what gives it the the most reward and learns to follow the traces which brings it to the reward. In Figure 4-20, a snapshot of the match against is shown. The yellow ones are GoToBall team, and the left side is their own goal. They are going to score their own goal. Although our members are staying in the middle of the field, they will get reward. The lines behind the players show the path they are following in 2-3 seconds. This is also one of the utilities of Teambots.

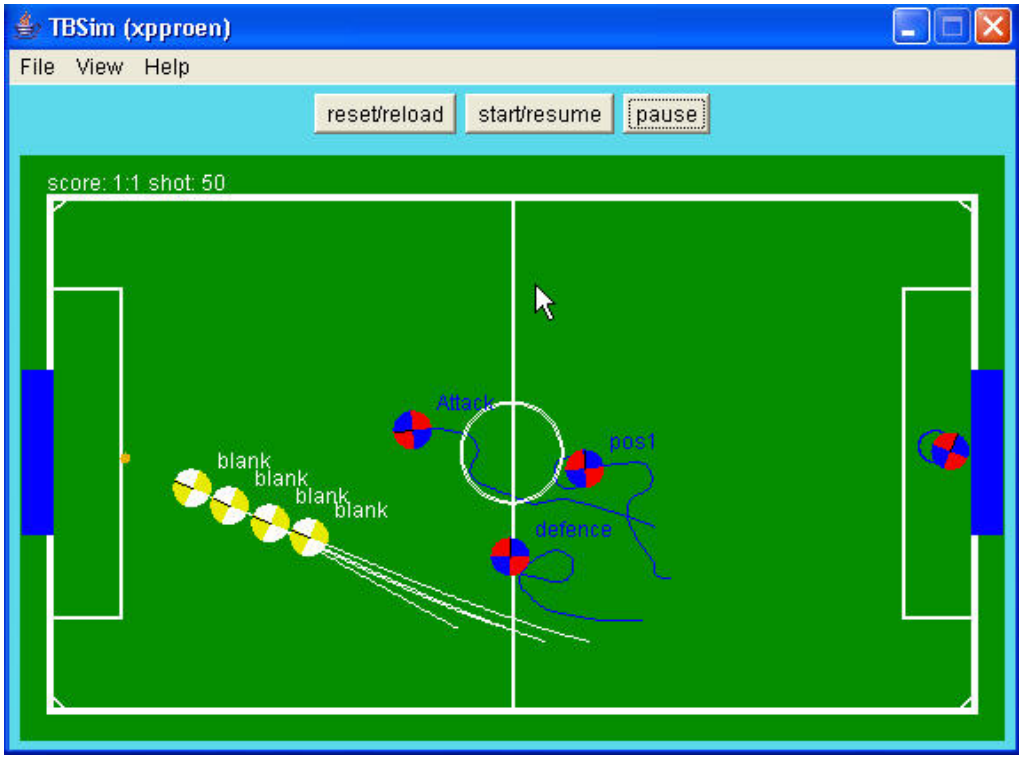


Figure 4-20 A snapshot of the match against GoToBall Team(the yellow ones)

With all experiments we notice that we have a strong defensive system, because our all players no matter attacker, defencer or supporter is affected by the ball field mostly. So even if the ball is in our zone all our players can protect our goal and kick it towards the other goal. And also our goalkeeper is not included in the role assignment learning part, he always knows his role.

We choose GoToBall team as an easy team, but since all his members going after the ball, sometimes it seems very hard to reach the ball for our team members. We confirm our statement that opponent impulsive field's strength should be small although obstacle avoidance is possible.

After GoToBall team we tried some teams like BasicTeam which has more complicated strategy than GoToBall, and see that they do not confuse our team as much as GoToBall Team. Like GoToBall team, some teams have one special aim, like only protecting own goal. They do not score a goal but do not let the opponent team score, training with such teams can make our learning strategy develop wrongly.

4.3.2 Training Experiments with Team BrianTeam

BrianTeam is a stronger team which do not score their own goal like GoToBall. But their one disadvantage is their lack of goalkeeper. The team does not have an assigned goalkeeper, the players at the back behave like defencer and protect their goal.

The results of 50 matches against BrianTeam before learning and after learning are shown in Figure 4-21 and Figure 4-22.

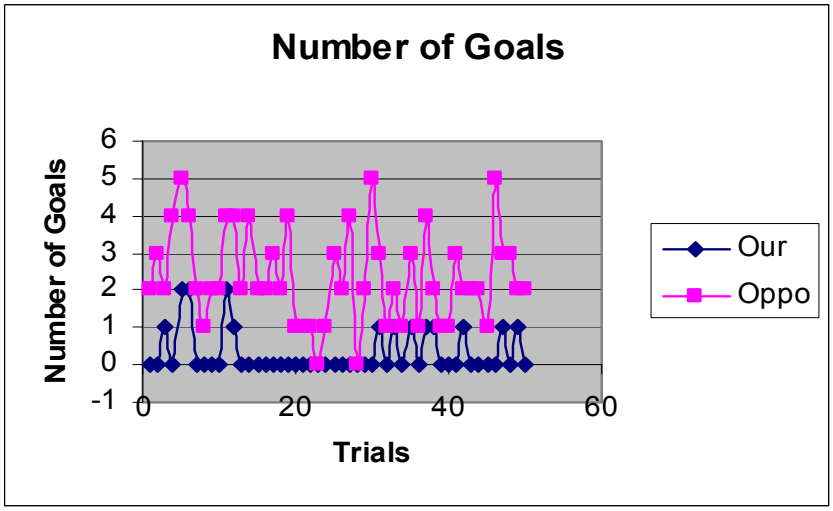


Figure 4-21 Goals Scored by BrianTeam and Our team before learning

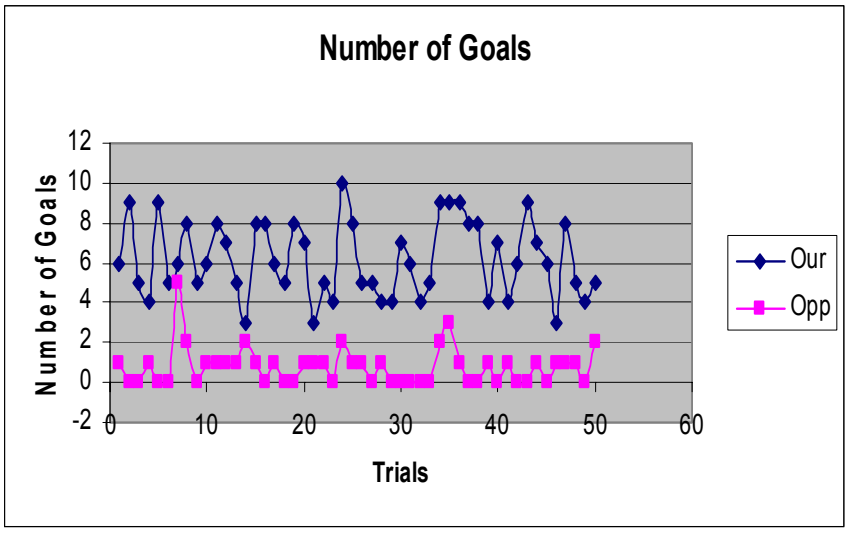


Figure 4-22 Goals Scored by BrianTeam and Our team after learning

4.3.2.1 Performance Analysis

Their disadvantage of not having goalkeeper provides us advantage, our members find the chance to score and gain reward. If they have very strong defensive capability it will become harder for our team to find the opportunity to gain reward as experienced with AIKOHomoG team.

We do 35 matches against BrianTeam with our Q-values gained while training with GoToBall Team to show that it is important how you train your team is important. Although we know that we can beat the BrianTeam after training with BrianTeam we can not get the same success after training our team with GoToBall Team.

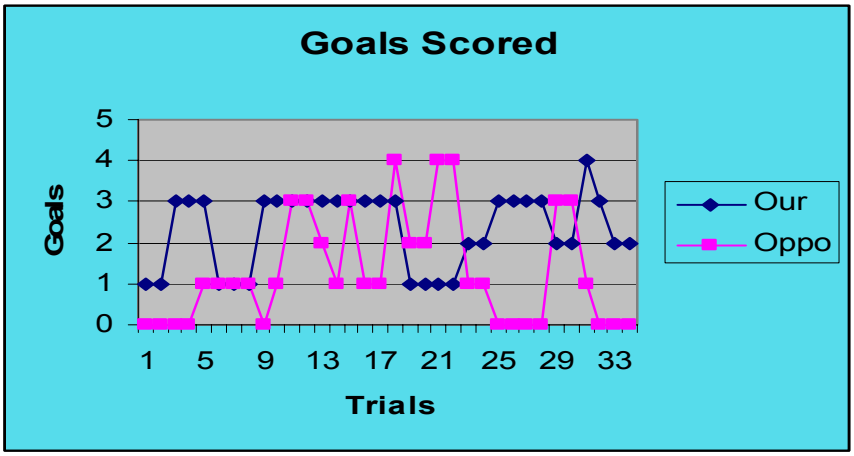


Figure 4-23 Goals Scored by BrianTeam and Our team after training with GoToBall

4.3.3 Training Experiments with Team AIKOHomo

Although we say that we have good skills, our team has not properties of passing the ball to team members, or using the wall pass. With a team that has more complex abilities we should trian our team to develop our defensive abilities.

AIKOHomo is the strongest team in our team set of consisting 25 teams, it has rarely lets our members go to their goal, tracks the ball and moves fastly and blocks the our member that has the ball. Its members pass the ball to the most appropriate member in the team strategy.

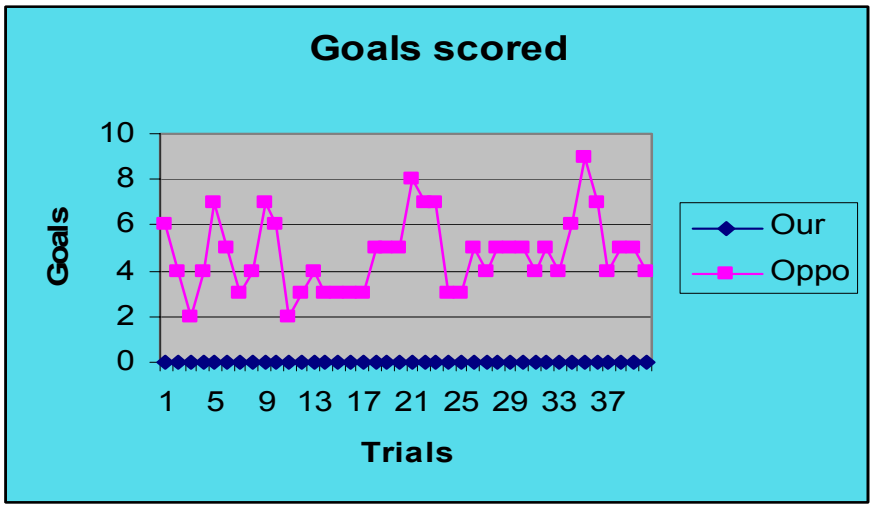


Figure 4-24 Goals Scored by AIKOHomoG and Our team before learning

Figure 4-25 Goals Scored by AIKOHomoG and Our team after learning

4.3.3.1 Performance Analysis

After training we manage to score opponents goal, the numbers of the opponents goals decrease also which shows that we learn to assign appropriate role assignment while ball is near our goal. Before learning, since it randomly selects the roles, it could make the mistake of changing its role while securing its goal as a defence player to support and place itself at a position around ball but not kick and secure the

goal. During training the team learns not to do these mistakes, because if it makes it get punishment.

4.3.4 Training Experiments with New Team

As explained before, after training several teams we decided to train our team with a team equal to ur team. But since its role assignment algorithm is rule based as shown in Figure 4-17, it gives simple but effective decisions.

The results of the matches against New Team before learning and after learning are shown in Figure 4-26 and Figure 4-27.

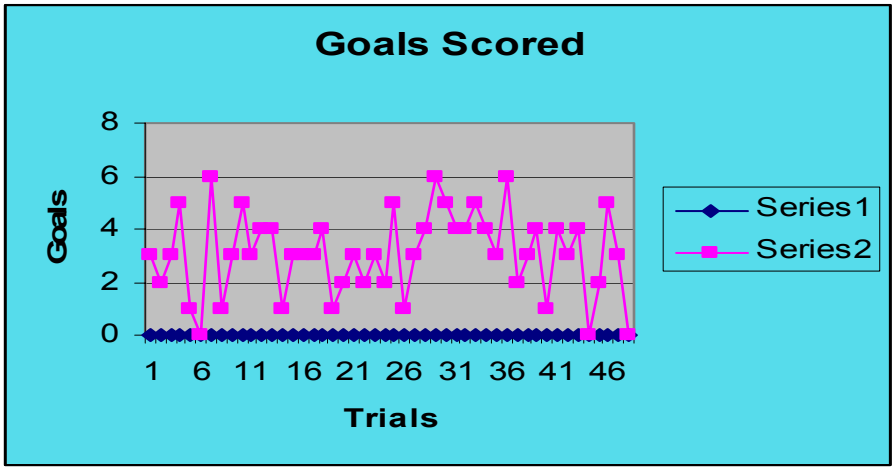


Figure 4-26 Goals Scored by NewTeam and Our team before learning

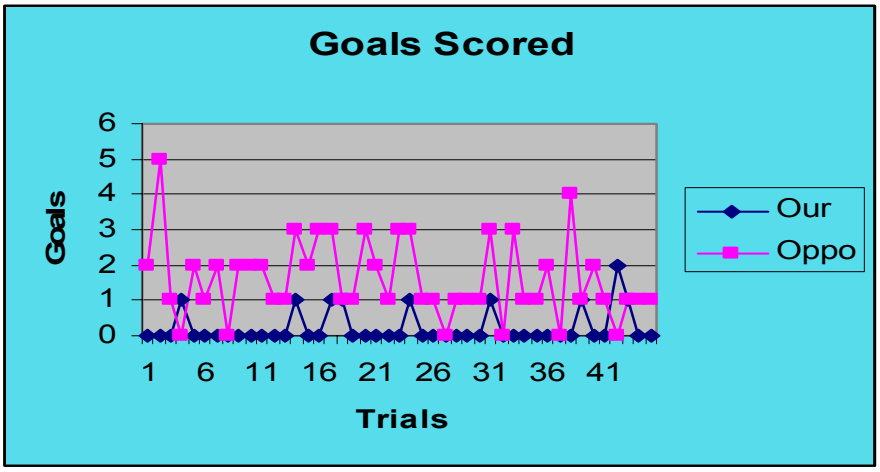


Figure 4-27 Goals Scored by NewTeam and Our team after learning

4.3.4.1 Performance Analysis

At first we could not get efficient result, since it has the same abilities with us, it does not shoot his own goal, fast and is mostly affected by ball. The same things can be written as in part 4.3.31. But the main difference from the other set trainings is the free ball persistence.

The free ball problem occurred at most among these two teams. Free ball decision is given when the ball could not move. In some situations, especially when the ball is between 2 opposite team members and wall, and oscillating continuosly, the system can not give the free ball decision because the ball is moving and then two team should wait the new cycle to begin at the middle of the field. Our team learn to persist and not to change its role during this interval. Because it leaves , it gives an oppurtunity to the New Team to score and our team takes punishment.

Another experiment we did with New Team. We trained our team against New Team with exploration rate equal to 0. But we notice that it has the problem of local optimum and gives usually the same action since it does not explore while training.

Figure 4-28 A snapsnot from the match between New Team and our Team

Figure 4-29A snapsnot from the match between New Team and our Team

Figure 4-28 and 4-29 shows the faults of the system, the members which is the most available for attack role is not assigned and the same action persists a while.

So many training experiments could be done, first training with a basic team to improve the attacking behaviors, then with values in the system with a more complicated team to improve offensive behaviors. But it is a reality the system develops itself with the rewards it takes, if at the same situations it could get negative rewards because of the opponent team complexity, the system will decrease the value of that state action pair.

With whom you train your system is important. New and specific exercises to your system should be developed and the team should train with them, in a control of a trainer.

CHAPTER 5

CONCLUSIONS

5.1 CONCLUSIVE REMARKS

In this research, we designed and tested a reinforcement learning system which takes its input from a competitive, dynamic world like roboccer and gives its output as the coordination of some agents like the team members. For the simulation of our team we used Teambots program, besides some difficulties like freeball decision problem, or its distributed controlled system it has many advantages since it is fully open source and provides various team programs. In our experiments we used these teams as opponents.

In reinforcement learning , learning is achieved through trial and error interactions with the dynamic environment. The main factors affected the learned policy are generalization of state space, value functions and opponent team. In our system potential field method is used to reduce state space dimension and to take the agents to the most attracting positions in the field. According to our results, we can say that potential field method is very effective in such dynamic environments. Sarsa(λ)-Learning shows good performance results in terms of speed of execution and convergence rate.

However it is hard to build competition level team by using only standard modeling. In our system the success of the team does not only belong to the reinforcement learning system success. Because rewards effect our system state action selectivity, if we have very basic roles, even if the system chooses the best action we can not get reward to increase its selectivity

5.2 FUTURE WORK

For future work reward shaping can be studied. In our system only negative and positive goals are used as rewards, but changing of the ball direction could be added to the system as feedback to show how their members are effective during game except shooting goals. But in this study, the researcher should be careful because the team could make the mistake of forgetting its main goal while maximizing its reward in simpler methods. Reward formalization is a crucial issue and should be studied and developed carefully.

As stated in the results chapter, a trainee system could be developed to train our system under control like the human teams do. Instead of training the team with arbitrary teams, a trainee system which make the team learn special traces will improve the system's performance. The team learns what to do against a team but its training depends only to the performance of the opponent team. Special trainee systems could be developed to learn different strategies.

REFERENCES

[1] Jong-Hwan Kim and Prahlad Vadakkepat, “Multi-Agent Systems: A Survey from the Robot-soccer Perspective”, Int. J. Intelligent Automation and Soft Computing, 6: (1) 3-17, 2000.

[2] Albus, J. S., “ Data Storage in the Cerebellar Model Articulation Controller (CMAC)” , Transactions of the ASME: Journal of Dynamic Systems, Measurement and Control, pp. 228-233, September 1975.

[3] Barto, A., Sutton, R., and Watkins, C., “Learning and sequential decision making”, Learning and Computational Neuroscience (ed), Cambridge, MA, MIT Press, 1990.

[4] Stone, P., Kuhlmann, G., “ Guiding a Reinforcement Learner with Natural Language Advice: Initial Results in Robocup Soccer “, Proceedings of the AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems, pp. 30-35, July 2004.

[5] Holland, J.H., “Properties of bucket brigade algorithm”, First Int. Conf. on Genetic Algorithms and their Applications, Pittsburg, PA, (1985), pp 1-7.

[6] Fayek, R. E., et al. “A System Architecture for a Mobile Robot Based on Activities and a Blackboard Control Unit”, IEEE Proc. Int. Conf. Robotics and Auto., U.S.A, (1993), pp 267-274.

[7] Shim, H.-S., Jung, M.-J., Kim, H.-S., Kim, J.-H., and Prahlad V. “A hybrid control structure for vision based soccer robot system.”, Int. J. of Intelligent Automamtion and Soft Computing.

[8] Lung Chi Kwong, “Research in Collaborative Team Behavior of Autonomous Soccer Robots”, City University of Hong Kong Computer Science.

[9] Khatib,O., “Real-time Obstacle Avoidance for Manipulators and Mobile Robots”, The International Journal Of Robotics Research 5(1986), pp 90-98.

[10] M. Hassoun, Y. Demazeau, C. Laugier., “ Motion control for a car-like robot: potential field and multi-agent approaches” Proceedings of International Conference on Intelligent Robots and Systems (IROS), Raleigh, USA, 1992.

[11] Douglas Vail, Manuela Veloso, ” Multi-Robot Dynamic Role Assignment and Coordination Through Shared Potential Fields”, Kluwer Academic Publishers, 2003.

[12] T.Balch and R.Arkın, “Behavior-based formation control for multi-robot teams”, IEEE Transactions on Robotics and Automation, Vol.12, No. 6, pp. 926-939, December 1998.

[13] T.Balch and M.Hybinette, “Social potentials for scalable multirobot formations”, In Proceedings of IEEE International Conference on Robotics and Automation (ICRA-2000), 2000.

[14] Thilo Weigel, Willi Auerbach, Markus Dietl, Burkhard Dümler, Jens-Steffen Gutmann, Kornel Marko, Klaus Müller, Bernhard Nebel, Boris Szerbakowski, and Maximilian Thiel, “CS Freiburg: Doing the right thing in a group”, Lecture Notes İn Computer Science, 2019:52-63,2001.

[15] CS Freiburg: “Coordinating Robots for Successful Soccer Playing” , IEEE Transactions on Robotics and automation, Vol.18, No.5, October 2002.

[16] Kaplan, K., “Design and Implementation of Fast Controllers for Mobile Robots.”, Boğaziçi, M.S. Thesis, 2003.

[17] Samuel, A. L., “ Some Studies in Machine Learning Using the Game of Checkers” IBM Journal on Research and Development, Vol. 3, pp. 210-229, 1959.

[18] Sutton, R. S., “Learning to Predict by the Methods of Temporal Differences” Machine Learning, Vol. 3, No.1, pp.9-44, August 1988.

[19] Leslie Pack Kaelbling, Michael L.Littman, Andrew W.Moore, “Reinforcement Learning: A Survey”, Journal of Artificial Intelligence Research 4(1996), pp 237-285.

[20] Tatlıdede, Utku, “Reinforcement Learning of Multi-Agent Team Behavior”, Boğaziçi, M.S. Thesis, 2003.

[21] Spaan, M., “ Team play among soccer robots”, Master’s thesis Artificial Intelligence University of Amsterdam.

[22] Kao-Shing Hwang, Shun-Wen Tan, Chien-Cheng Chen, “ Cooperative Strategy Based on Adaptive Q-Learning for Robot Soccer Systems”

[23] Stone, P., R. S. Sutton and S. Signh, “ Reinforcement Learning for 3 vs. 2 Keepaway”, in A. Birk, S. Coradeschi and S. Tadokoro (eds.), RoboCup-2001: Robot Soccer World Cup V, Springer Verlag, Berlin, 2002.

[24] R. S. Sutton, A. Barto, “Reinforcement Learning, An Introduction” , MIT Press,1998.

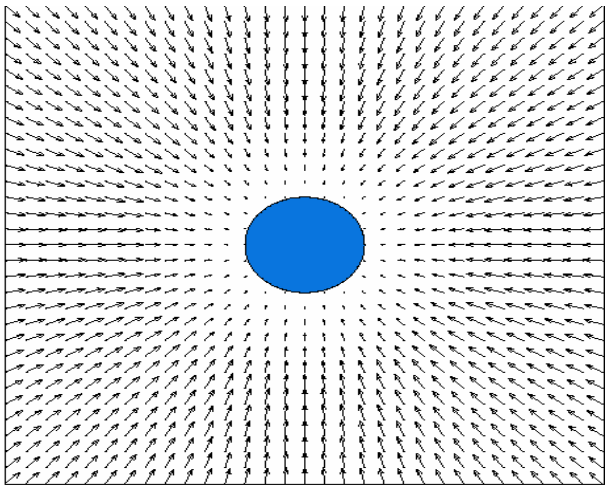
[25] P. Stone and M. Veloso. “ Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork’’ , Artificial Intelligence, 1999.

[26] Tews, A., Wyeth G. “Multi-Robot Coordination in the Robot Soccer” Environment Proceedings of the Australian Conference on Robotics and Automation (ACRA '99).

APPENDIX

GENERATION of POTENTIAL FIELD proposed by Michael A. Goodrich

ATTRACTIVE FIELD



One way to think of a potential field is to think of it as a mapping from one vector into another vector. For the 2-D navigation in the figure, it is the mapping from the vector

$v = [x, y]^T$ into the gradient vector $\Delta = [\Delta x, \Delta y]^T$ (the superscript T represents "transpose" - I use it because I like column vectors better than row vectors). Now, we could find Δ by defining some vector function of v and then taking the gradient of this function, but I prefer a more direct approach. To generate the above fields, I just defined Δx and Δy in terms of v as follows:

- Let (x_G, y_G) denote the position of the goal. Let r denote the radius of the goal.

Let $v = [x,y]^T$ denote the (x,y) position of the agent.

- Find the distance between the goal and the agent
  : $d = \sqrt{(x_G - x)^2 + (y - y_G)^2}$
- Find the angle between the agent and the goal $\theta = \tan^{-1}\left(\frac{y_G - y}{x_G - x}\right)$ (I use the atan2 function because it gives you the angle in the correct quadrant.)

- Set Δx and Δy according to the following:

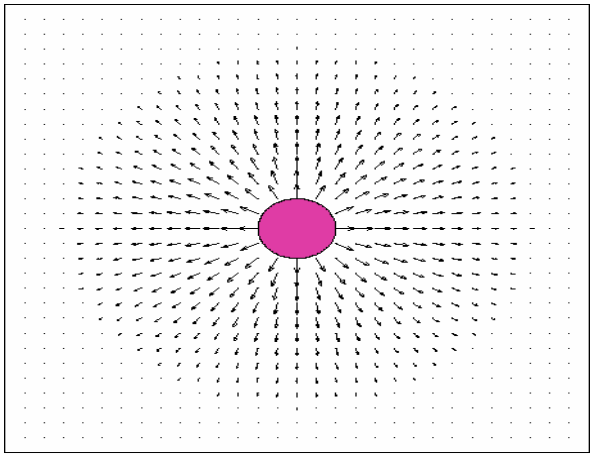
$$\text{if } d < r \qquad \Delta x = \Delta y = 0$$

$$\text{if } r \le d \le s + r \qquad \Delta x = \alpha(d - r)\cos(\theta) \text{ and } \Delta y = \alpha (d - r)\sin(\theta)$$

$$\text{if } d > s + r \qquad \Delta x = \alpha s\cos(\theta) \text{ and } \Delta y = \alpha s\sin(\theta)$$

This sets up a goal as a circle with radius r. When the agent reaches the goal no forces from the goal act upon it, whence when d < r both Δx and Δy are set to zero. The field has a spread of s and the agent reaches the extent of this field when d = s + r. Outside of this circle of extent, the vector magnitude is set to the maximum possible value. Within this circle of extent but outside of the goal's radius, the vector magnitude is set proportional to the distance between the agent and the goal. I include the constant $\alpha > 0$ so that the strength of the field can be easily scaled.

REPULSIVE FIELD



- Let (x_G,y_G) denote the position of the obstacle. Let r denote the radius of the obstacle.

Let $v = [x,y]^T$ denote the (x,y) position of the agent.

- Find the distance between the goal and the agent
  : $d = \sqrt{(x_O - x)^2 + (y - y_O)^2}$
- Find the angle between the agent and the goal $\theta = \tan^{-1}\left(\frac{y_O - y}{x_O - x}\right)$ (I use the atan2 function because it gives you the angle in the correct quadrant.)

- Set Δx and Δy according to the following:

if $d < r$ $\quad \Delta x = -\text{sign}(\cos(\theta))\infty \quad \Delta y = -\text{sign}(\sin(\theta))\infty$

if $r \le d \le s + r$ $\quad \Delta x = -\beta(s + r - d)\cos(\theta)$ and $\Delta y = -\beta(s + r - d)\sin(\theta)$

if $d > s + r$ $\quad \Delta x = \Delta y = 0$