To appear in Applied Artificial Intelligence

TR-97-11
Received 21 Apr 97

Soccer Server: a tool for research on multi-agent systems

Itsuki Noda
Hitoshi Matsubara
Kazuo Hiraki
Ian Frank

Electrotechnical Laboratory
Umezono 1-1-4, Tsukuba
Ibaraki, JAPAN 305

Abstract

This paper describes Soccer Server, a simulator of the game of soccer designed as a test-bench for evaluating multi-agent systems and cooperative algorithms. In real life, successful soccer teams require many qualities, such as basic ball control skills, the ability to carry out plans, and teamwork. We believe that simulating such behaviors is a significant challenge for Computer Science, Artificial Intelligence and Robotics technologies. It is to promote the development of such technologies, and to help define a new standard problem for research, that we have developed Soccer Server. We demonstrate the potential of Soccer Server by reporting an experiment that uses the system to compare the performance of a neural network architecture and a decision tree algorithm at learning the selection of soccer play-plans. Other researchers using Soccer Server to investigate the nature of cooperative behavior in a multi-agent environment will have the chance to assess their progress at RoboCup-97, an international competition of robotic soccer to be held in conjunction with IJCAI-97. Soccer Server has been chosen as the official server for this contest.

© Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, Ian Frank

Soccer Server: a tool for research on multi-agent systems

NODA Itsuki, MATSUBARA Hitoshi, HIRAKI Kazuo, and Ian FRANK
Electrotechnical Laboratory, Tsukuba, Ibaraki 305, Japan

August 13, 1997

Abstract

This paper describes Soccer Server, a simulator of the game of soccer designed as a test-bench for evaluating multi-agent systems and cooperative algorithms. In real life, successful soccer teams require many qualities, such as basic ball control skills, the ability to carry out plans, and teamwork. We believe that simulating such behaviors is a significant challenge for Computer Science, Artificial Intelligence and Robotics technologies. It is to promote the development of such technologies, and to help define a new standard problem for research, that we have developed Soccer Server. We demonstrate the potential of Soccer Server by reporting an experiment that uses the system to compare the performance of a neural network architecture and a decision tree algorithm at learning the selection of soccer play-plans. Other researchers using Soccer Server to investigate the nature of cooperative behavior in a multi-agent environment will have the chance to assess their progress at RoboCup-97, an international competition of robotic soccer to be held in conjunction with IJCAI-97. Soccer Server has been chosen as the official server for this contest.

Keywords: Multi-agent Systems, Machine Learning, Neural Networks

1 Introduction

The rules of *soccer*, or association football, were formulated in 1863 in the UK by the Football Association, and the game has since become one of the most widely played in the world. Soccer is played by a team of players, and it is generally accepted that the best results can only be produced by playing co-operatively. It is also a real-time game played in a dynamically changing environment about which any individual player typically only has partial information. Our belief is that these features make the modeling of soccer play a significant challenge for multi-agent systems and cooperative algorithms. The World Cup Robot Soccer Initiative (RoboCup) is an attempt to foster interest in this area. By formulating a new standard problem for research, RoboCup aims to encourage and promote developments in Computer Science, AI and Robotics technologies towards an original and significant goal.

Standard problems have long been a driving force for AI research. Historically, for example, the acceptance of the 'Turing Test' [18] focused attention on the mimicking of human behavior as a test of machine intelligence. However, at the time the Turing Test was beyond the capabilities of any practical machine technology, so attention was turned instead to easier domains such as simple, two person-games of strategy like chess and checkers. Research on these games, and on chess in particular, has generated significant advances in the theory of search algorithms and search control, as well as motivating cognitive studies into the ways that humans approach the same problems (*e.g.*, see [8, 7, 5]). Whereas chess is a static, turn-based, two-person, complete information game, soccer is dynamic, real-time, multi-person, and has incomplete information. The notion of using soccer as a new standard problem for research has therefore recently been proposed [4].

To facilitate research on the soccer domain, we have been developing a simulator of soccer, called Soccer Server. The aim of this server is to provide a test-bench for easily evaluating various multi-agent systems and cooperative algorithms. Using the server, a team of players written in any kind of programming environment with the facilities of UDP/IP socket communication and string manipulation can play a match against another team (possibly written in another programming environment). One of the activities of RoboCup is to organize a 'simulator league' of tournaments between such software soccer teams. Our Soccer Server will be used as the official server for RoboCup-97, an international simulator league to be held in conjunction with IJCAI-97.

To demonstrate the usefulness of the Soccer Server in practice, we present the results of an experiment that uses the server to investigate the learning of cooperative behavior. Specifically, we show how the selection of play-plans can be learned by two attackers who are trying to score a goal in the face of an opposing player. The

details of this experiment are given in Section 4. Before this, Section 2 gives some further justifications for the initial choice of soccer as a challenging problem, and Section 3 gives a detailed description of the design and the workings of the Soccer Server. In Section 5, we summarize our work and give some conclusions.

2 Soccer and Multi-agent Systems

The development of multi-agent systems involves research issues such as cooperation protocols, distributed control and effective communication. An important reason for choosing soccer as an example domain is that it provides a dynamic, real-time environment in which it is still relatively easy for such tasks to be classified, monitored and assessed.

2.1 Soccer as a New Standard Problem

From the standpoint of multi-agent systems, soccer is a good example of a problem that has many real-world properties, but is still moderately abstracted. A general characterization of soccer could be given as follows:

- The environment changes in a dynamic, real-time way. This is in contrast to, say, board games, where the situation is static and turn-based. In soccer, it is important for each player to plan reactively in real-time, because it is difficult to make a whole plan before play starts.
- Robustness is more important than elaboration. Complex plans that fail due to unforeseen events will typically be less useful than even simple fail-safe strategies.
- Cooperation within a team is obviously advantageous.
- Information on the state of the world is only partial. Individual players only see what is in front of them, and often their view is impaired by other players or by the distance between them and the far portions of the pitch.
- Adaptability is a desirable property. Teams that can dynamically change their approach according to the perceived playing style and strategies employed by the opponents will have an advantage.
- Verbal communication is not precise. Communication between team members is limited by time constraints, by the physical separation of the players, and often by the noise generated by supporters. It is also important for a team not to accidentally communicate its intentions to the opponents. Teamwork is therefore best supported by combining simple signals with the current situation.

In the past, various example domains have been used to evaluate the performance of multi-agent systems, but they have typically been simple and small. Soccer is complex and large, and we believe that the characteristics detailed above demonstrate that it offers many and varied ways to evaluate the behavior of multi-agent systems.

Moreover, because soccer has a long history and is a globally-known sport, many people will be able to give an intuitive assessment about what is 'good' behavior simply by observing a game. There is also scope for formalizing and incorporating the techniques and strategies that have been developed by real human soccer players and professionals.

Note that the usefulness of soccer as a test domain has already been realized by some researchers. For example, Asada and his colleagues have been using reinforcement learning to develop the skills of a soccer playing robot [20, 3, 2, 1, 19]. Their robot learns how to shoot a ball effectively by constructing internal state spaces representing the environment. Other soccer playing robots are being developed by Sahota [14, 13, 12], and Shimada et al. [15]. Also, an earlier version of the Soccer Server has been used by Stone and Veloso to investigate learning under multi-agent environments [16, 17]. They apply techniques of neural networks and machine learning to improve a player's skills and to acquire the ability to select good play-plans.

2.2 RoboCup

The World Cup Robot Soccer Initiative (RoboCup) is an attempt to foster AI and intelligent robotics research by providing a standard problem on which a wide range of technologies can be integrated and examined. RoboCup has been jointly developed and formalized [6, 4] by researchers at the Electrotechnical Laboratory, the Sony Computer Science Laboratories and Osaka University in Japan, after they realized the potential of an early

version of the Soccer Server system [9]. The first RoboCup will be held in conjunction with IJCAI-97, Nagoya, Japan.

The first RoboCup (RoboCup-97) consists of two tracks of competitions (a real robot league and simulator league), plus an exhibition of robots with soccer skills. It is expected that, with current technology, it will be very difficult to construct a team of real robots that plays soccer well. The simulator league is therefore an important focus for the independent development of the software techniques required to control multi-agent systems. In particular, the simulator league removes the problems of physical construction of robots and the symbolic representation and interpretation of the real world. This is handled by the Soccer Server — the official server of the first RoboCup simulator league — which we describe in the next section.

3 Soccer Server

Soccer Server enables a soccer match to be played between two teams of player-programs (possibly implemented in different programming systems). A match using Soccer Server is controlled using a form of client-server communication. Soccer Server provides a virtual soccer field (*e.g.*, see Figure 1) and simulates the movements of players and a ball. A client program can provide the 'brain' of a player by connecting to the Soccer Server via a computer network (using a UDP/IP socket) and specifying actions for that player to carry out. In return, the client receives information from the player's sensors. All communication between the server and each client is in the form of ASCII strings. Therefore, clients can be realized by any kind of programming environment on any architecture that has the facilities of UDP/IP sockets and string manipulation.

Figure 2 gives an overview of how the Soccer Server communicates with clients. The three main modules in the Soccer Server itself are:

1. **A field simulator module**. This calculates movements of objects and checks for collisions on the field of play.

2. **A message-board module**. This module manages the communication between the clients and Soccer Server.

3. **A referee module**. This module ensures that the rules of the game are followed.

Below, we describe each of these modules in turn. We also describe how Soccer Server introduces noise into its simulation, and give details on how the latest version of the system can be obtained.

3.1 Simulator Module: Motion of Objects

The soccer field and all objects on it are 2-dimensional. This means that, unlike in real football, the ball cannot be kicked in the air and the players cannot make use of skills such as heading or volleying. The size of the field, measured in the notional internal equivalent of meters, is 105m $\times$ 68m. Note that, in general, the parameters in the simulation are not direct equivalents of the situation in real soccer, but are tuned in order to make the server as useful as possible for the evaluation of competing multi-agent systems. For example, the width of the goals is 14.64m, double the size of ordinary goals, because the 2-dimensional nature of the simulation makes scoring more difficult.

Players and the ball are treated as circles, as can be seen more clearly in the screen capture of Figure 3. Each objects' movements are simulated one by one in the following stepwise fashion: (1) if the object is a player, any changes in the player's velocity (or the velocity of the ball) resulting from commands issued by the client controlling that player are calculated, (2) the velocity of the object is added to its position, and (3) the object's velocity decays at a pre-determined rate. Collisions are also simplified. If, after being moved, one object overlaps a second (that is, two objects collide with each other), the first object is moved back until the overlap is removed and its velocity is multiplied by -0.1.

There also exist various landmarks — flags, lines and goals — on the field, which are visible but do not move. These may be used by clients to help determine their player's position, orientation or speed and direction of movement. As we discuss below, clients are not given information on the absolute positions of their own players, but only on the relative location of other objects. Thus, these fixed objects are important reference points for the client programs' calculations.

Figure 1: Window image of Soccer Server

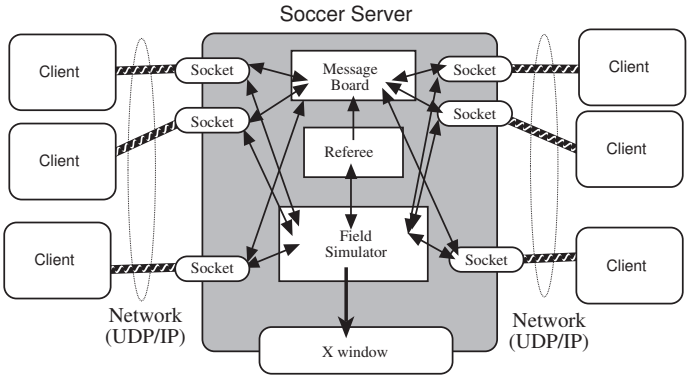

Figure 2: Overview of Soccer Server

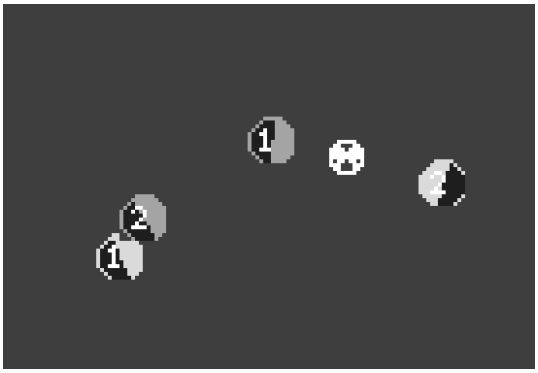

Figure 3: Close-up of players and a ball

3.2 Message Board Module: Communication Protocol

A basic principle of Soccer Server is that each client should control just one player: communication between individual clients is possible, but only by sending commands to Soccer Server. Thus, the server can connect with up to 22 clients. In practice, it is relatively easy to cheat the server and establish inter-client communication outside the server. In a competitive situation such as RoboCup, then, the 'one-client one-player' rule is effectively a gentlemans' agreement. To facilitate testing, it is of course possible for a single program to control multiple players (by establishing multiple UDP/IP socket connections), but such programs will only be permitted in RoboCup tournaments on the understanding that the control of each player is separated logically.

All communication between the server and each client is in the form of ASCII strings. The protocol of the communication consists of:

- **Control commands**: messages sent from a client to the server to control actions of the client's player (see Table 1), and
- **Sensor information**: messages sent from the server to a client describing the current state of the game from the viewpoint of the client's player (Table 2).

The level of abstraction chosen for representing these messages was an important consideration during the design of Soccer Server. One possibility was a low-level, physical description, for example allowing power values for drive motors to be specified. However, it was felt that such a representation would concentrate users' attention too much on the actual control of a players' actions, relegating true investigation of the multi-agent nature of team-play to the level of a secondary objective. Further, it is difficult to design a low-level description that is not implicitly based on a specific notion of robot hardware; for example, control of speed by drive motors is biased towards a physical implementation that uses wheels. On the other hand, a more abstract representation, maybe using tactical commands such as `pass-ball-to` and `block-shoot-course`, would produce a game in which the real-world nature of soccer becomes obscured. Thus, our representation is a compromise, using basic control commands such as `turn`, `dash`, and `kick`. To make good use of these commands, clients will need to tackle both the problems of control in an incomplete information, dynamic environment and also the best way to combine the efforts of multiple players. Thus, we achieve our goal of providing a test-bench for research on multi-agent systems that has significant real-world properties.

Note that part of the communication protocol provides a method for clients to communicate with each other. Specifically, a client can inform a message to other clients using the `say` command. This message is broadcast to all clients immediately as auditory information. However, not all the other clients are guaranteed to hear this information, since there is a maximum range of communication of 50m, and any player can only hear one message from each team during each two simulation cycles. Also, the length of the message itself is restricted. These restrictions make it impossible, in practice, to use one client in a team (*e.g.*, a 'captain') to direct and inform all the others. Instead, effective and efficient use of the `say` command is encouraged, imparting information only when it is useful and timely.

(move *X Y*) Move the player to the position (X, Y). The origin is the center mark, and the X-axis and Y-axis are toward the opponent's goal and the right touchline respectively. Thus, X is usually negative to locate a player in its own side of the field. This command is available only in the `before_kick_off` mode.
(turn *Moment*) Change the direction of the player according to *Moment*. *Moment* should be $-180 \sim 180$. The actual change of direction is reduced when the player is moving quickly.
(dash *Power*) Increase the velocity of the player in the direction it is facing by *Power*. *Power* should be $-30 \sim 100$. (The range is modifiable by a Soccer Server configuration file.)
(kick *Power Direction*) Kick the ball by *Power* to the *Direction* if the ball is near enough (the distance to the ball is less than 1.0 + `ball_size` + `player_size`.). *Power* should be $-30 \sim 100$, and *Direction* should be $-180 \sim 180$. (The ranges are modifiable by a Soccer Server configuration file.)
(say *Message*) Broadcast *Message* to all players. *Message* is informed immediately to clients as sensor information in the `(hear ...)` format described below. *Message* must be a string whose length is less than 512, and consist of alphanumeric characters and the symbols "`+-*/_.()` ".

Table 1: Control commands

(see *Time ObjInfo ObjInfo ...*) Describes visual information. *Time* indicates the current time. *ObjInfo* is information about a visible object in the following format: (*ObjName Distance Direction DistChng DirChng*) *ObjName* ::= (`player` *Teamname UNum*) | (`goal` *Side*) | (`ball`) | (`flag [l|c|r] [t|b]`) | (`flag p [l|r] [t|c|b]`) | (`line [l|r|t|b]`) where *Distance* and *Direction* are the relative distance and direction of the object respectively, and *DistChng* and *DirChng* are respectively the radius- and angle-components of change of the relative position of the object. If the object is not movable, *DistChng* and *DirChng* are omitted. As the distance to another player becomes larger and larger, more and more information about the player is lost. Specifically, *UNum* is lost in the case of players further than a certain distance, and *Teamname* is lost from players who are very far. *Vdirection* is also lost when the object is too far. The frequency and quality of this information can be selected by the client.
(hear *Time Direction Message*) Describes auditory information. *Direction* is the direction of the sender. In the case that the sender is the client itself, `Direction` is '`self`'. *Time* indicates the current time. This message is sent immediately when a client sends a (`say` *Message*) command. Judgments of the referee are also broadcast using this form. In this case, *Direction* is '`referee`'.

Table 2: Sensor information

3.3 Referee Module: Playing by the Rules

The referee module monitors the game and, as in real soccer, regulates the flow of play by making a number of decisions based on the rules of the game. Firstly, when the ball is in a goal, the referee module announces the goal to all clients, stops play, and moves the ball back to the center mark. Secondly, when the ball goes out of the field of play, the referee module announces its decision (a throw-in, corner or goal-kick), stops play, moves the ball to an appropriate re-start position, and ensures (by physically moving them) that any opposing players are at least 9.15m from the ball. Thirdly, when the alloted game time for each half has been played, the server announces 'time-up' and again stops play. Finally, once play has been re-started after a stoppage (by a successful `kick` command from the appropriate team), the referee announces that play has re-commenced.

At kick-offs, all players must be in their own half of the pitch. There is a period of 5 seconds for players to get back to their own half after a goal. During this time, the clients can use the `move` command to move their players directly to a specified position. After this, players in the opponent's half are moved to a random position in their own half by the referee.

There are no rules about the usage of hands, because players have no hands (this also means that technically there are no 'goalies', although it may still be a good tactic for a team to leave a player in its own goalmouth). The off-side rule is also ignored in the current implementation. Remaining fouls, like 'obstruction', are difficult to judge automatically, because they concern players' intentions. Therefore Soccer Server provides a way for a human user to instruct the referee to award a free kick. To do this, the user simply clicks on the offending player in the Soccer Server display window. The referee module then stops play and re-starts it with a free-kick for the opposing team from the player's position.

As a final feature, when invoked with the `-coach` option, the server provides an extra socket for a privileged client (called a coach client) that has the ability to direct all aspects of the game. The coach client can move all objects, direct the referee module to make decisions, and announce messages to all clients. This facility is extremely useful for tuning and debugging client programs, which usually involves the repeated testing of the behaviors of the clients in many situations. The utility of the `-coach` option is demonstrated by the experiment we describe in Section 4, for which its use was indispensable.

3.4 Noise

In order to reflect the nature of the real world, the server introduces various types of uncertainty into the simulation, as follows:

- Noise added to the movement of objects. The amount of this noise increases with the speed of the object.
- Noise added to command parameters. Small random numbers are added to the parameters of commands sent from clients.
- Limited command execution. The server executes only one command for each player in each simulation cycle. If a client sends more than one command in a cycle, it is not informed which one was executed.
- Inexact sensing. The further an object, the less reliable the information returned about it from the server.

The presence of these uncertainties re-enforces the importance of robust behavior, and of the reactive monitoring of the outcomes of a player's actions.

3.5 Implementation and Access Point

Soccer Server is implemented using g++ and the X-window system with the Athena widget-set. Currently, we support SunOS 4.1.x, Solaris 2, DEC OSF/1, NEWS OS, IRIX and Linux, and in the future we plan to expand this support to cover other OSs and architectures. The sources of Soccer Server are available by FTP:

```
ftp://ci.etl.go.jp/pub/soccer/server/sserver-*
```

Manuals and sample clients are also available from the following homepage:

```
http://ci.etl.go.jp/~noda/soccer/server.html
```

4 Soccer Server as a Research Tool

We have described how Soccer Server has been developed as a test-bench for multi-agent algorithms in a dynamic, real-world environment. In order to demonstrate the potential of Soccer Server, we now describe two experiments that were carried out within its framework. Each experiment involves learning the selection of a play-plan in a multi-agent environment: the first using a neural network, and the second using a decision tree algorithm.

4.1 Selection of Play-plans in Soccer

As we have previously discussed, the dynamic, multi-agent nature of soccer forces each player to react to — and plan for — the current situation in real-time, and to at least pay some attention to the behaviors and actions of other agents. In general, it is clearly difficult to write down an *a priori* set of rules for guiding behavior in such a complex environment, so our experiments will consider instead how some simple control techniques may be learned.

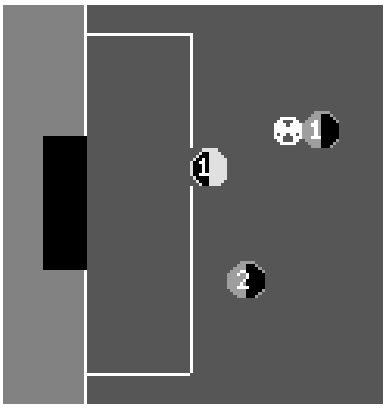

Figure 4: A 'two-on-one' situation

In particular, we will examine the simple situation of two attackers attempting to score a goal against a single opponent. An example of such a 'two-on-one' situation is shown in Figure 4. For the attackers, this kind of situation typically requires cooperative play: the offensive player can shoot the ball at the goal directly, or can pass the ball to his teammate. Considering just this binary selection, it is reasonable to argue that the following rules would serve the attacker well:

- If the opponent is blocking the shoot-course, pass the ball to teammate.
- If the opponent is blocking the pass-course, shoot the ball directly.

In order to control a player's behavior using these rules, the notions of 'blocking the shoot-course' and 'blocking the pass-course' would need to be understood. For any given situation, it may be possible to specify this by hand, but in general there are a very large number of possible play situations, including many where both play-plans may appear equally good (or bad). We therefore examine the more robust approach of acquiring a control strategy for this problem automatically.

In order to investigate this kind of learning, we used Soccer Server to carry out experiments on the selection of play-plans in two-on-one situations. The set-up of the experiments was as follows:

1. The three players are placed in the penalty area as shown in Figure 5. The main attacker is placed randomly within the box numbered 1, his teammate is placed randomly within the box numbered 2 and the defender is placed randomly in the box numbered 3. All three players face in a random direction. The ball is placed goal-side of the main attacker at a distance of 1m, on a line parallel with the touchlines.
2. The main attacker behaves in the following manner:
   (a) Collect information about the positions of all objects (players, ball and goal).

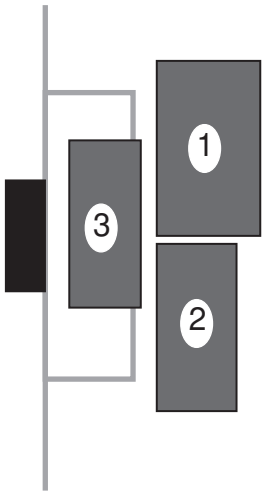

Figure 5: Areas of initial positions of players

(b) Input the information to a *selector* module.

(c) Choose either a 'pass' or a 'shoot' plan[1] according to the output of the *selector* module.

3. The defender is programmed to move towards and, if possible, to kick, the ball. In terms of sensing, this just requires the defender to `turn` until he can see the ball: a significant advantage over the main attacker. In practice, then, the defender's movement is slowed down slightly to make scoring more likely.

4. The attacker's teammate is programmed to wait for a pass and, if the pass comes, to shoot at goal.

5. The coach mode of Soccer Server is used to decide when a goal is scored, and to report the 'success' of the attack to all players. If the ball leaves the field of play, or if a time-limit is exceeded, 'failure' is reported to all players.

We carried out 2 experiments, using two types of *selectors*. First, we used a neural network trained on a sample set of data using back-propagation. Next, we applied the decision tree learning procedure of C4.5 to the same data set. The results of these experiments are given below.

4.2 Learning with Neural Networks

In the first experiment, we used a simple neural network as the *selector*. The network had eight inputs, which corresponded to the eight values of the distance and the angle from the main attacking player to his teammate, the defender, the center of the goal, and the ball. There were 30 hidden-units and 2 output-units, all fully connected. The output units were designed to give the values of O_{pass} and O_{shoot}: the expected success rate of the pass and shoot plans. Thus, when using the network to select an action in a given situation, the pass-plan is chosen with probability $O_{\text{pass}}/(O_{\text{pass}} + O_{\text{shoot}})$ and the shoot-plan is chosen with probability $O_{\text{shoot}}/(O_{\text{pass}} + O_{\text{shoot}})$.

Initially, we created a random database of 1000 situations, recording the positions of all objects from the perspective of the main attacker. For each situation we then chose, at random, whether to carry out a pass-plan or a shoot-plan and added the 'success' or 'failure' of a single simulation of this plan to the data item. The rationale behind this database was to produce information similar to that obtainable by a real team of clients playing competitively within Soccer Server. A more powerful database, from a training perspective, would have estimated the *probability* of success of *each* of the shoot- and pass-plans for every game situation, by trying each plan several times. However, such a luxury is typically not open to client programs when playing a game (typically, there is just one chance in any given situation), so we restricted ourselves to just the basic information that could reasonably be expected from an actual simulation.

[1] In this experiment, the 'pass' plan and 'shoot' plan are very simple. It is assumed that the attacker will `turn` until all the necessary data is collected and then either shoot directly at the center of the goal or pass directly at his teammate, who shoots directly at the center of the goal. In the future, we plan to investigate the learning of more complex plans containing parameters.

Using our database, we applied the back-propagation method to train the (initially random) weights of a network. We then compared the performance of the initial and the trained network when 1000 unseen situations were played within Soccer Server. The number of successes and failures before and after learning are summarized in Table 3.

Before Learning			
	pass	shoot	total
success	247(55.6%)	231(41.5%)	478(47.8%)
failure	197(44.4%)	325(58.5%)	522(52.2%)
total	444	556	1000
After Learning			
	pass	shoot	total
success	327(74.3%)	353(63.0%)	680(68.0%)
failure	113(25.7%)	207(37.0%)	320(32.0%)
total	440	560	1000

Table 3: Success rate of shoot and pass plans selected by network

Since the weights of the connections in the initial network were set randomly, it selects pass- and shoot-plans fairly equally. The situations where the pass-plan is selected result in 'success' over 50% of the time, whereas the situations where the shoot-plan is selected result in 'success' only 40% of the time (as might be expected, the configuration of players determined by Figure 5 tends make it easier to score by passing). In the trained network, however, the success rates of both plans are significantly improved.

A more detailed picture of the performance of the network can be gained from Figure 6, which shows how the output of the network changes with the direction to the opposing player (the other network inputs are fixed to represent the situation where the goal is 10 meters away at an angle of -10 degrees, the teammate is 10 meters at an angle of -95 degrees, and the opponent is 8 meters away). In this graph, the 'shoot' curve dips when the direction to the defender is close to the direction of the goal itself. On the other hand, the 'pass' curve goes down when the defender is in the same direction as the teammate. These curves can be viewed as implicitly reflecting rules to select play-plans: the area where the 'shoot' curve dips represents the situation where the defender is 'blocking the shoot-course', while the area where the 'pass' curve dips represents the defender 'blocking the pass-course'. The acquisition of these conditions can be regarded as the source of the improved success-rate for the attackers.

Note that we also found that the network reflects to some extent a model of the defender's abilities. This is demonstrated by Figure 7, which adds an extra dimension to the previous graph by allowing the distance to the defender to vary, as well as the angle. This time, only the 'shoot' output of the network is shown, but it can be seen that the surface forms a valley along the direction of the goal, and that the width of the valley bottom becomes narrower as the distance to the defender increases. Thus, the defender is regarded as having a fixed area of coverage, whose apparent angle becomes smaller as the distance from the attacker grows larger.

4.3 Learning with C4.5

In order to compare the functional differences between the neural network approach and a symbolic learning method we conducted another experiment using C4.5[10] on the same task; learning to select pass- and shoot-plans. In C4.5, learning is based on the decision tree method. We used the two possible play-plans, `pass` and `shoot`, as well as the relative positions of the objects (the two remaining players, the ball and center of the goal) as decision attributes, and 'success' and 'failure' as the two possible classes for categorization.

Using the same database of 1000 situations created for the neural network experiment, C4.5 learned a decision tree that classified 'success' and 'failure'. We utilized C4.5's rule generation mechanism to transform the decision tree into a rule set with certainty factors [11], such as the following:

```
Rule 77:
     action = shoot
     mate_dist > 8.1
     opponent_dist > 7.4
->  class success  [81.9%]
```

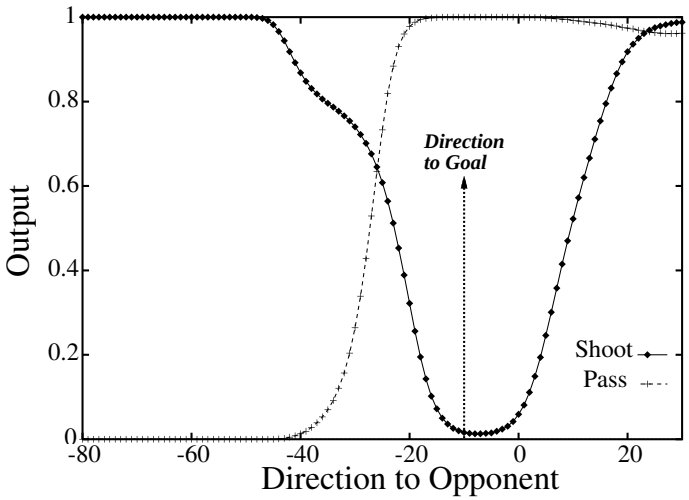

Figure 6: Change of network output with direction to opponent

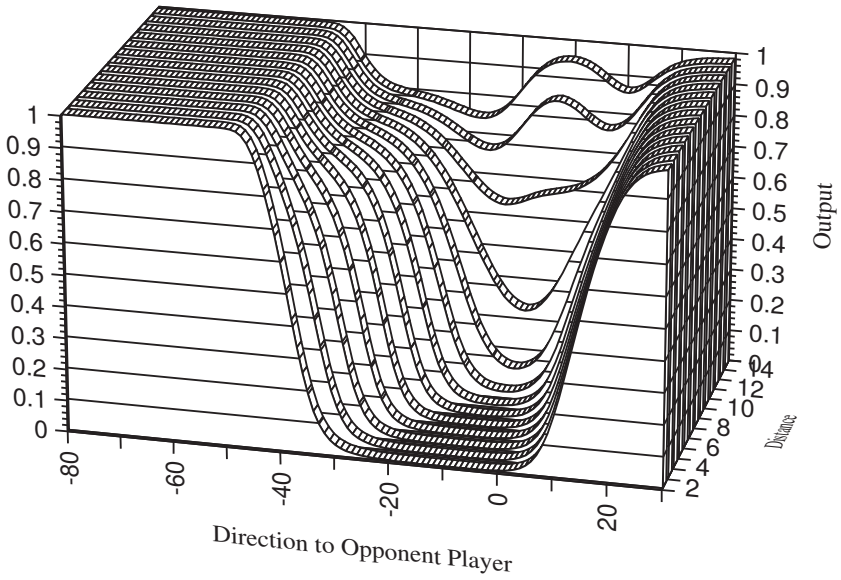

Figure 7: Change of shoot-output with distance and direction to opponent

Note that, unlike in the neural network experiment, we cannot directly use the output of the learning phase to implement a *selector*, because the information about passing and shooting is embedded as attribute values in the C4.5 decision tree. Rather than just predicting whether a given action will be a 'success' or a 'failure', the attacker must actually make a selection between the two possibilities. This problem is solved by creating a *selector* for any given situation in the following simple fashion:

1. Combine the information about the positions of the objects in the situation (in attribute-value form) with the pair `action = shoot`.
2. Combine the information about the positions of the objects in the situation (in attribute-value form) with the pair `action = pass`, to make a second data set.
3. Predict a class, 'success' or 'failure', for each of the two data sets.
4. If the class for one of the pass-plan or the shoot-plan is 'success' and the class for the other is 'failure', do the 'success' action.
5. If the both classes are 'success', do the action whose certainty-factor is higher.
6. If the both classes are 'failure', pass or shoot randomly.

Table 4 shows the rates of 'success' and 'failure', on a random data set of 1000 situations, of the pass-plans and shoot-plans chosen by this *selector*. Compared with the results of the neural network experiment, we can see that the C4.5-based selector is a little inferior, although it still manages to produce significant increases in success rates over random selection. One of the reasons for the difference in performance could be the *hyperrectangle approximation* of C4.5. Since decision tree learning in general approximates continuous regions as hyperrectangles, the numeric attributes representing the relative positions of objects may be leading to inappropriate generalizations.

Before Learning (Network)			
	pass	shoot	total
success	247(55.6%)	231(41.5%)	478(47.8%)
failure	197(44.4%)	325(58.5%)	522(52.2%)
total	444	556	1000

After Learning (Network)			
	pass	shoot	total
success	327(74.3%)	353(63.0%)	680(68.0%)
failure	113(25.7%)	207(37.0%)	320(32.0%)
total	440	560	1000

After Learning (C4.5)			
	pass	shoot	total
success	339(77.2%)	298(53.1%)	637(63.7%)
failure	100(22.8%)	263(46.9%)	363(36.3%)
total	439	561	1000

Table 4: Comparing the success rate of a random network, a trained network, and a C4.5-based selector

4.4 Discussion

Although the learning of shoot/pass plans is a very simple problem, the above results show that both back-propagation and decision tree learning can be used to improve the performance of players. However, it is important to clarify the advantages and disadvantages of each method.

For example, one of the advantages of the back-propagation method is that it has an *incremental framework*. As we have mentioned previously, a team can gain significant benefits by being able to adapt to the strategies and tactics used by the opponents. Particularly for dynamic environments like soccer, this is important. Whereas the incremental framework of the back-propagation method allows for such adaptability, the 'batch' framework

of C4.5 is not so suitable. One of our plans for further research, therefore, is to experiment with incremental decision tree learning algorithms such as ID5[21].

On the other hand, one of the merits of using symbolic methods like C4.5 is its *comprehensibility*. For instance, the example C4.5 rule given in the previous section could be rendered textually as:

> If neither the opponent player nor the teammate are nearby, shoot the ball at the goal directly.

Comprehensibility plays an important role when integrating programs from disparate sources. In the case of neural network learning, it may be problematic to combine a learned network with a pre-encoded program. However, if the output of learning system is comprehensible — such as that produced by C4.5 — it is typically easier for a human programmer to integrate it with a higher-level program.

Thus, as might be expected, decision tree learning and neural network learning each have their own advantages and disadvantages. Note, though, that both approaches (at least as formalized above) ignore the issue of *co-learning*, where each agent in a multi-agent domain adapts and learns its behavior concurrently with the changes of other agents. Especially in a dynamic environment such as soccer such a model of behavior, where each agent is not fixed and static but attempts to improve its performance along with other agents, seems more realistic. In the future, then, we plan to look at hybrid systems that synthesize network and symbolic methods, and also incorporate the possibility of co-learning.

5 Conclusion

In this paper, we have described Soccer Server, a simulator of the game of soccer in which players, controlled by individual client programs, can play a soccer match. We also demonstrated the potential of Soccer Server by reporting experiments that used the system to investigate how the selection of play-plans could be learned.

Soccer Server is primarily designed as a tool for research. It is our hope that the use of Soccer Server in the RoboCup tournaments will provide a significant research impetus and lead directly to the development and investigation of new ideas and techniques for multi-agent systems. We intend to monitor the course of this research and update the specifications of Soccer Server to keep up with its progress.

Our own experiments were carried out as a first step in a study of the learning of cooperative behavior in a multi-agent domain. The next stages of this work will involve more complex planning tasks that increase the degree of cooperation required from agents. We also intend to investigate the simultaneous development of multiple agents using co-learning.

Acknowledgments

The authors thank Hiroaki Kitano (Sony CSL), Minoru Asada (Osaka Univ.), Yasuo Kuniyoshi (ETL), Peter Stone (CMU), Milind Tambe (USC) and Eiichi Osawa (Sony CSL) for valuable suggestions and discussions on the design of the Soccer Server.

References

[1] M. Asada, S. Noda, and K. Hosoda. Action-based sensor spcae categorisation for robot learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, pages 1502–1509, 1996.

[2] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Vision-based reinforcement learning for purposive behavior acquisition. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1995.

[3] Minoru Asada, Shoichi Noda, and Koh Hosoda. Non-physical intervention in robot learning based on lfe method. In *Proc. of Machine Learning Conference Workshop on Learning from Examples vs. Programming by Demonstration*, 1995.

[4] Hiroaki Kitano et. al. Robocup synthetic agent challenge 97. In *Proc. of Fifteenth International Joint Conference on Artificial Intelligence*, 1997. (submitted).

[5] P.W. Frey. *Chess Skill in Man and Machine*. Springer Verlag, 1977.

[6] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup intiative. In *Working Notes of IJCAI Workshop: Entertainment and AI/Alife*, pages 19–24, Aug. 1995.

[7] R. Levinson, F. Hsu, J. Schaeffer, T. Marsland, and D. Wilkins. Panel: The role of chess in artificial intelligence research. In *Proceedings of the 12th IJCAI*, pages 547–552, Darling Harbour, Sydney, Australia, 1991.

[8] D. Levy, editor. *Computer Games II.* Springer Verlag, 1988.

[9] Itsuki Noda. Multi-agent soccer game server (in japanese). In *MACC '94*, 10 1994.

[10] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[11] J. R. Quinlan. Generating production rules from decision trees. In *Tenth International Joint Conferece on Artificial Intelligence*, pages 304–307, 1993.

[12] M. K. Sahota. Reactive delivation: an architecture for real-time intelligent control in dynamic environments. In *Proc. of AAAI-94*, pages 1303–1308, 1994.

[13] Michael K. Sahota and Alan K. Mackworth. Can situated robots play soccer? In *Proc. of Canadian AI-94*, 1994.

[14] Mickael K. Sahota. Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. matster thesis, Department of Computer Science, The University of British Columbia, Aug. 1993.

[15] Kazuyuki SHIMADA, Tomoaki IWASAKI, and Tsutomu HOSHINO. Evolution of strategies in iterated prisoner's dilemma and pk-soccer – can artificial life contribute to competition-type entertainments? – . JSAI Technical Report SIG-PPAI-9403-5, Japanese Society for Artificial Intelligence, Mar. 1995. (in Japanese).

[16] P. Stone and M. Veloso. Learning to pass or shoot: collaborate or do it yourself. unpublished manuscript, 1995.

[17] Peter Stone and Manuela Veloso. Towards collaborative and adversarial learning: A case study in robotic soccer. Proceedings SS-96-01, pp. 88–92, 1996 AAAI Symposium, March 1996.

[18] A.M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[19] E. Uchibe, M. Asada, and K. Hosoda. Behavior coordination for a mobile robot using modular reinforcement learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS96)*, pages 1329–1336, 1996.

[20] Eiji Uchibe, Minoru Asada, Shoichi Noda, and Koh Hosoda. The coordination of multiple behaviors for a mobil robot acquired by vision-based reinforcement learning (in japanese). Report SIG-FAI-9403-4, JSAI, Mar. 1995.

[21] P.E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 2(1):161–186, 1989.