Towards Collaborative and Adversarial Learning: A Case Study in Robotic Soccer

Peter Stone and Manuela Veloso*
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{pstone,veloso}@cs.cmu.edu
http://www.cs.cmu.edu/{~pstone,~mmv}

In *International Journal of Human Computer Studies*, 48, 1998.

Abstract

Soccer is a rich domain for the study of multiagent learning issues. Not only must the players learn low-level skills, but they must also learn to work together and to adapt to the behaviors of different opponents. We are using a robotic soccer system to study these different types of multiagent learning: low-level skills, collaborative, and adversarial. Here we describe in detail our experimental framework. We present a learned, robust, low-level behavior that is necessitated by the multiagent nature of the domain, namely shooting a moving ball. We then discuss the issues that arise as we extend the learning scenario to require collaborative and adversarial learning.

1 Introduction

Soccer is a rich domain for the study of multiagent learning issues. Teams of players must work together in order to put the ball in the opposing goal while at the same time defending their own. Learning is essential in this task since the dynamics of the system can change as the opponents' behaviors change. The players must be able to adapt to new situations.

Not only must the players learn to adapt to the behaviors of different opponents, but they must learn to work together. Soon after beginning to play, young soccer players learn that they cannot do everything on their own: they must work as a team in order to win.

---

*This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U. S. Government.

However, before players can learn any collaborative or adversarial techniques, they must first acquire some low-level skills that allow them to manipulate the ball. Although some low-level skills, such as dribbling, are entirely individual in nature, others, such as passing and receiving passes, are necessitated by the multiagent nature of the domain.

Our approach to multiagent learning in this complex domain is to develop increasingly complex layers of learned behaviors from the bottom up. Beginning with individual skills that are appropriate in a multiagent environment, we identify appropriate behavior parameters and learning methods for these parameters. Next, we incorporate these learned individual skills into higher-level multiagent learning scenarios. Our ongoing goal is to create a full team of agents that use learned behaviors at several different levels to reason strategically in a real-time environment.

In this article, we present detailed experimental results of our successful use of neural networks for learning a low-level behavior. This learned behavior, namely shooting a moving ball, is crucial to successful action in the multiagent domain. It also equips our clients with the skill necessary to learn higher-level collaborative and adversarial behaviors. We illustrate how the learned individual skill can be used as a basis for higher level multiagent learning, discussing the issues that arise as we extend the learning task.

2 Multiagent Learning

Multiagent learning is the intersection of Multiagent Systems and Machine Learning, two subfields of Artificial Intelligence (see Figure 1). As described by Weiß, it is "learning that is done by several agents and that becomes possible only because several agents are present" (Weiß, 1995). In fact, in certain circumstances, the first clause of this definition is not necessary. We claim that it is possible to engage in multiagent learning even if only one agent is actually learning. In particular, if an agent is learning to acquire skills to *interact* with other agents in its environment, then regardless of whether or not the other agents are learning simultaneously, the agent's learning is multiagent learning. Especially if the learned behavior enables additional multiagent behaviors, perhaps in which more than one agent *does* learn, the behavior is a multiagent behavior. Notice that this situation certainly satisfies the second clause of Weiß's definition: the learning would not be possible were the agent isolated.

Traditional Machine Learning typically involves a single agent that is trying to maximize some utility function without any knowledge, or care, of whether or not there are other agents in the environment. Examples of traditional Machine Learning tasks include function approximation, classification, and problem-solving performance improvement given empirical data. Meanwhile, the subfield of Multiagent Systems, as surveyed in (Stone & Veloso, 1997), deals with domains having multiple agents and considers mechanisms for the interaction of independent agents' behaviors. Thus, multiagent learning includes any situation in which an agent *learns* to *interact* with *other agents*, even if the other agents' behaviors are static.

The main justification for considering situations in which only a single agent learns to be multiagent learning is that the learned behavior can often be used as a basis for more complex interactive behaviors. For example, this article reports on the development of a low-level learned

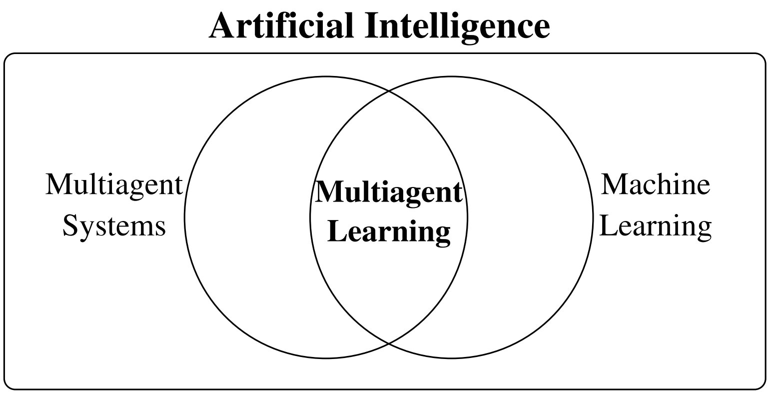


Figure 1: Multiagent Learning is at the intersection of Multiagent Systems and Machine Learning, two subfields of Artificial Intelligence.

behavior in a multiagent domain. Although only a single agent does the learning, the behavior is only possible in the presence of other agents, and, more importantly, it enables the agent to participate in higher-level collaborative and adversarial learning situations. When multiagent learning is accomplished by layering learned behaviors one on top of the other, as in this case, all levels of learning that involve interaction with other agents contribute to, and are a part of, multiagent learning.

There are some previous examples of single agents learning in a multiagent environment which are included in the multiagent learning literature. One of the earliest multiagent learning papers describes a reinforcement learning agent which incorporates information that is gathered by another agent (Tan, 1993). It is considered multiagent learning because the learning agent has a cooperating agent and an adversary agent with which it learns to interact.

Another such example is a negotiation scenario in which one agent learns the negotiating techniques of another using Bayesian Learning methods (Zeng & Sycara, 1996). Again, this situation is considered multiagent learning because the learning agent is learning to interact with another agent: the situation only makes sense due to the presence of multiple agents. This example represents a class of multiagent learning in which a learning agent attempts to model other agents.

One final example of multiagent learning in which only one of the agents learns is a training scenario in which a novice agent learns from a knowledgeable agent (Clouse, 1996). The novice learns to drive on a simulated race track from an expert agent whose behavior is fixed.

The one thing that all of the above learning systems have in common is that the learning agent is *interacting* with other agents. Therefore, the learning is only possible due to the presence of these other agents, and it may enable higher-level interactions with these agents. These characteristics define the type of multiagent learning described in this article.

3 Robotic Soccer

Robotic soccer has recently been emerging as a challenging topic for Artificial Intelligence researchers interested in machine learning, multiagent systems, reactive behavior, strategy acquisition, and several other areas of Artificial Intelligence. This section briefly summarizes previous research and describes our research platform.

3.1 Related Work

A ground-breaking system for Robotic Soccer, and the one that served as the inspiration for our work, is the Dynamo System developed at the University of British Columbia (Sahota, Mackworth, Barman, & Kingdon, 1995). This system was designed to be capable of supporting several robots per team, but most work has been done in a 1 vs. 1 scenario. Sahota used this system to introduce a decision making strategy called *reactive deliberation* which was used to choose from among seven hard-wired behaviors (Sahota, 1993). Although this approach worked well for a specific task, ML is needed in order to avoid the cumbersome task of hard-wiring the robots for every new situation and also for expanding into the more complex multiagent environment.

Modeled closely after the Dynamo system, the authors have also developed a real-world robotic soccer system (Achim, Stone, & Veloso, 1996). The main differences from the Dynamo system are that the robots are much smaller, there are five on each team, and they use Infra-red communication rather than radio frequency.

The Robotic Soccer system being developed in Asada's lab is very different from both the Dynamo system and from our own (Asada, Noda, Tawaratsumida, & Hosoda, 1994a; Asada, Uchibe, Noda, Tawaratsumida, & Hosoda, 1994b). Asada's robots are larger and are equipped with on-board sensing capabilities. They have been used to develop some low-level behaviors such as shooting and avoiding as well as a RL technique for combining behaviors (Asada et al., 1994a, 1994b). By reducing the state space significantly, he was able to use RL to learn to shoot a stationary ball into a goal. His best result in simulation is a 70% scoring rate. He has also done some work on combining different learned behaviors with a separate learned decision mechanism on top (Asada et al., 1994b). While the goals of this research are very similar to our own, the approach is different. Asada has developed a sophisticated robot system with many advanced capabilities, while we have chosen to focus on producing a simple, robust design that will enable us to concentrate our efforts on learning low-level behaviors and high-level strategies. We believe that both approaches are valuable for advancing the state of the art of robotic soccer research.

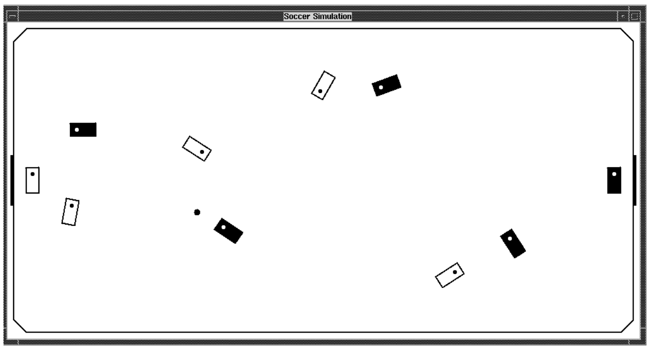
Although real robotic systems, such as those mentioned above and the many new ones being built for robotic soccer tournaments (Stone, Veloso, & Achim, 1996; Kitano, Kuniyoshi, Noda, Asada, Matsubara, & Osawa, 1997), are needed for studying certain robotic issues, it is often possible to conduct research more efficiently in a well-designed simulator. Several researchers have previously used simulated robotic soccer to study ML applications. Using the Dynasim soccer simulator (Sahota, 1996, 1993), Ford et al. used a Reinforcement Learning (RL) approach with sensory predicates to learn to choose among low-level behaviors (Ford, Boutilier, & Kanazawa, 1994). Using the simulator described below, Stone and Veloso used Memory-based Learning to allow a player to learn when to shoot and when to pass the ball (Stone & Veloso, 1996). In the RoboCup Soccer Server (Noda, 1995) Matsubar et al. also used a neural network to allow a player to learn when to shoot and when to pass (Matsubara, Noda, & Hiraki, 1996).

The work described in this article builds on previous work by learning a more difficult behavior: shooting a *moving* ball into a goal. The behavior is specifically designed to be useful for more complex multiagent behaviors as described in Section 5.

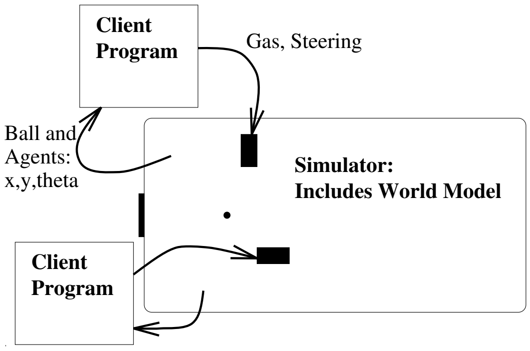
3.2 The Simulator

Although we are pursuing research directions with our real robotic system (Achim et al., 1996), the simulator facilitates extensive training and testing of learning methods. All of the research reported in this article was conducted in simulation.

Both the simulator and the real-world system are based closely on systems designed by the Laboratory for Computational Intelligence at the University of British Columbia (Sahota et al., 1995). In particular, the simulator's code is adapted from their own code, for which we thank Michael Sahota whose work (Sahota, 1993) and personal correspondence (Sahota, 1994) has been motivating and invaluable. The simulator facilitates the control of any number of agents and a ball within a designated playing area. Care has been taken to ensure that the simulator models real-world responses (friction, conservation of momentum, etc.) as closely as possible. Sensor noise of variable magnitude can be included to model more or less precise real systems. A graphic display allows the researcher to watch the action in progress, or the graphics can be toggled off to speed up the rate of the experiments. Figure 2(a) shows the simulator graphics.



(a)



(b)

Figure 2: **(a)** The graphic view of our simulator. Eventually teams of five or more agents will compete in a real-time game of robotic soccer. **(b)** The interface between the clients and the simulator.

The simulator is based on a client-server model in which the server models the real world and reports the state of the world while the clients control the individual agents (see Figure 2(b)). Since the moving objects in the world (i.e. the agents and the ball) all have a position and orientation, the simulator describes the current state of the world to the clients by reporting the x, y, and θ coordinates of each of the objects indicating their positions and orientations. The clients periodically send throttle and steering commands to the simulator indicating how the agents should move. The simulator's job is to correctly model the motion of the agents based on these commands as well as the motion of the ball based on its collisions with agents and walls. The parameters modeled and the underlying physics are described in more detail in Appendix A.

The clients are equipped with a path planning capability that allows them to follow the shortest path between two positions (Latombe, 1991; Reeds & Shepp, 1991). For the purposes of this article, the only path planning needed is the ability to steer along a straight line. This task is not trivial since the client can only control the agent's motion at discrete time intervals. Our algorithm controls the agent's steering based on its offset from the correct heading as well as its distance from the line to be followed. This algorithm allows the agent to be steering in exactly the right direction within a small distance from the line after a short adjustment period. Thus the agent is able to reliably strike the ball in a given direction. For more details on the line following method, see Appendix B.

4 Learning a Low-Level Multiagent Behavior

The low-level skill that we focus on in this article is the ability to shoot a moving ball. Although a single agent does the shooting, we consider it a multiagent learning scenario since the ball is typically moving as the result of a pass from a teammate: it is possible only because other agents are present. Furthermore, as discussed in Section 5, this skill is necessary for the creation of higher level multiagent behaviors. However, in order to use the skill in a variety of situation, it must be as robust and situation-independent as possible. In this section, we describe in detail how we created a robust, learned behavior in a multiagent scenario.

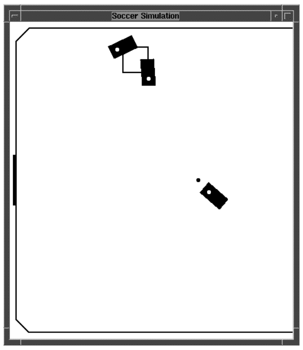
In all of our experiments there are two agents: a *passer* accelerates as fast as possible towards a stationary ball in order to propel it between a *shooter* and the goal. The resulting speed of the ball is determined by the distance that the passer started from the ball. The shooter's task is to time its acceleration so that it intercepts the ball's path and redirects it into the goal. We constrain the shooter to accelerate at a fixed constant rate (while steering along a fixed line) once it has decided to begin its approach. *Thus the behavior to be learned consists of the decision of when to begin moving: at each action opportunity the shooter either starts or waits.* Once having started, the decision may not be retracted. The key issue is that the shooter must make its decision based on the observed field: the ball's and the shooter's (x, y, θ) coordinates reported at a (simulated) rate of $60Hz$.[1] The method in which the shooter makes this decision is called its *shooting policy*.

Throughout our experiments, the shooter's initial position varies randomly within a continuous range: its initial heading varies over 70 degrees and its initial x and y coordinates vary independently over 40 units as shown in Figure 3(a).[2] The two shooters pictured show the extreme possible starting positions, both in terms of heading and location.

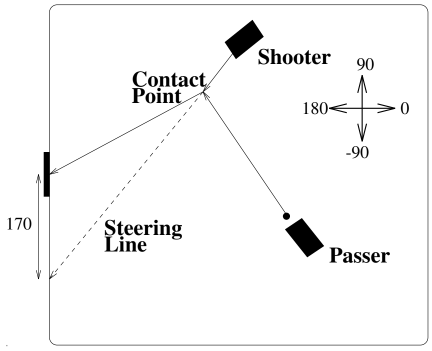
[1] The Dynamo system reported the coordinates at a rate of 60Hz using an overhead camera and color-coded objects (Sahota, 1993).

[2] For reference, the width of the field (the side with the goal) shown in figure 3(a) is 480 units. The width of the goal is 80 units.

Since the ball's momentum is initially across the front of the goal, the shooter must compensate by aiming wide of the goal when making contact with the ball (see Figure 3(b)). Before beginning its approach, the shooter chooses a point wide of the goal at which to aim. Once deciding to start, it then steers along an imaginary line between this point and the shooter's initial position, continually adjusting its heading until it is moving in the right direction along this line (see Appendix B). The line along which the shooter steers in the *steering line*. The method in which the shooter chooses the steering line is called its *aiming policy*.



(a)



(b)

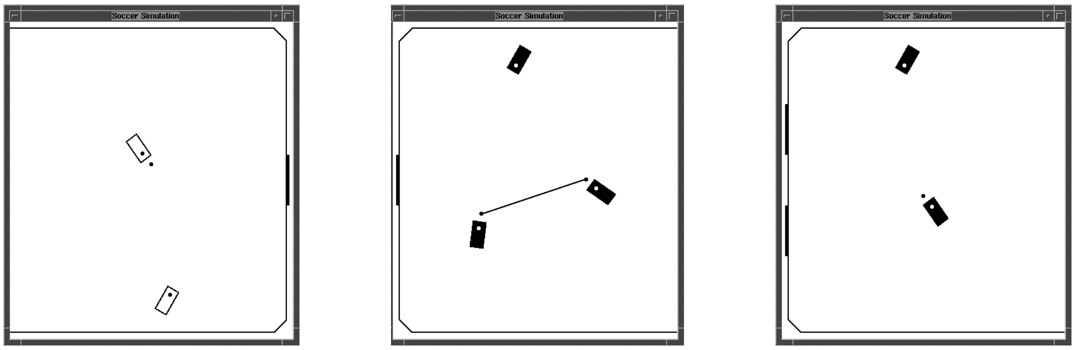
Figure 3: **(a)** The initial position for the experiments in this paper. The agent in the lower part of the picture, the *passer*, accelerates full speed ahead until it hits the ball. Another agent, the *shooter* then attempts to redirect the ball into the goal on the left. The two agents in the top of the figure illustrate the extremes of the range of angles of the shooter's initial position. The square behind these two agents indicates the range of the initial position of the *center* of the shooter. **(b)** A diagram illustrating the paths of the ball and the agents during a typical trial.

The task of learning a shooting policy has several parameters that can control its level of difficulty. First, the ball can be moving at the same speed for all training examples or at different speeds. Second, the ball can be coming with the same trajectory or with different trajectories. Third, the goal can always be in the same place during testing as during training, or it can change locations (think of this parameter as the possibility of aiming for different parts of the goal). Fourth, the training and testing can occur all in the same location, or the testing can be moved to a different *action quadrant*: a symmetrical location on the field. To acquire a robust behavior, we perform and report a series of experiments in which we increase the difficulty of our task incrementally. We develop a learning agent to test how far training in a limited scenario can extend. Table 1 indicates how the article is organized according to the parameters varied throughout our experiments. Figure 4 illustrates some of these variations.

We use a supervised learning technique to learn the task at hand. Throughout the article, all training is done with simulated sensor noise of up to about 2 units for x and y and 2 degrees for θ. All reported success rates are based on at least 1000 trials.

	Fixed	Variable
Ball Speed	Section 4.1	Sections 4.2, 4.3, 4.4
Ball Trajectory	Sections 4.1, 4.2	Sections 4.3, 4.4
Goal Location	Sections 4.1, 4.2, 4.3	Section 4.4
Action Quadrant	Sections 4.1, 4.2, 4.3, 4.4—training	Sections 4.1, 4.2, 4.3, 4.4—testing

Table 1: The parameters that control the difficulty of the task and the sections in which they are varied.



(a) (b) (c)

Figure 4: Variations to the initial setup: **(a)** The initial position in the opposite corner of the field (a different action quadrant); **(b)** Varied ball trajectory: The line indicates the ball's possible initial positions, while the passer always starts directly behind the ball facing towards a fixed point. The passer's initial distance from the ball controls the speed at which the ball is passed; **(c)** Varied goal position: the placement of the higher and lower goals are both pictured at once.

4.1 Fixed Ball Motion

We began our experimentation with the ball always being passed with the same trajectory and the same speed for all training and testing examples. With this condition of *fixed ball motion*, the shooter could always aim at the same point wide of the goal, guaranteeing that if contact was made, the ball would be propelled in the right direction. That is to say, the shooter used a constant aiming policy. We determined that with the trajectory (140^o) and speed ($\approx$ 135 units/sec) of the ball we were initially using, the shooter would score when contacting the ball if its steering line was such that it aimed 170 units wide of the center of the goal (illustrated in Figure 3(b)). This point remains constant throughout this section and Section 4.2.

Before setting up any learning experiments, we found a simple fixed shooting policy that would allow the shooter to score consistently when starting at the exact center of its range of initial positions. Starting at this position, the shooter could score consistently if it began accelerating when the ball's distance to its projected point of intersection with the agent's path reached 110 units or less. We call this policy the *simple shooting policy*. However, this simple policy was clearly not appropriate for the entire range of shooter positions that we considered: when using this policy while starting at random positions, the shooter scored only 60.8% of the time.

4.1.1 Choosing the inputs

Convinced that a ML technique could provide a significantly better shooting policy, we decided to try using a neural network as our initial attempt. We plan to experiment with other ML techniques on the same task in the future. We first considered how we should structure the neural network in order to learn a function from the current state of the world to an indication of whether the shooter should start accelerating or remain still and wait. The output of this function was fairly straightforward. It would indicate whether starting to accelerate in a world state described by the input values was likely to lead to a goal (outputs close to 1) or a miss (outputs close to 0). However, deciding how to represent the world state, i.e. the inputs to the neural network, represented a core part of our research.

One option was to use coordinates (x, y, θ) for both the shooter and the ball. However, such inputs would not have generalized beyond the very limited training situation. Furthermore, they would have led to a higher dimensional function (6) than turned out to be necessary. Instead we chose to use just 3 easily-computable coordinate-independent predicates.

Since the line along which the agent steered was computed before it started moving (the line connecting the agent's initial position and the point 170 units wide of the goal), and since the ball's trajectory could be estimated (with some error due to noise) after getting two distinct position readings, the shooter was able to determine the point at which it hoped to strike the ball, or the *Contact Point*. It could then cheaply compute certain useful predicates:

- *Ball Distance*: the ball's distance to the Contact Point;
- *Agent Distance*: the agent's distance to the Contact Point; and
- *Heading Offset*: the difference between the agent's initial heading and its desired heading.

The physical meaning of these inputs is illustrated in Figure 5(a). These inputs proved to be sufficient for learning the task at hand. Furthermore, since they contained no coordinate-specific information, they enabled training in a narrow setting to apply much more widely as shown at the end of this section.

4.1.2 Gathering training data

Recall that when the shooter started in the center of its range, it would score using the simple shooting policy: it began moving when the Ball Distance was 110 units or less. However, to get a diverse training sample, we replaced this shooting policy with a *random shooting policy* of the form "at each opportunity, begin moving with probability $\frac{1}{x}$." To help choose x, we determined that the shooter had about 25 decision opportunities before the ball moved within 110 units of the Contact Point. Since we wanted the shooter to start moving before or after these 25 decision cycles with roughly equal probability so as to get a balanced training sample, we solved the equation $(\frac{x-1}{x})^{25} = .5 \Rightarrow x \approx 37$. Hence, when using the random shooting policy, the shooter started moving with probability 1/37 at each decision point.

Using this shooting policy, we then collected training data. Each instance consisted of four numbers: the three inputs (Ball Distance, Agent Distance, and Heading Offset) at the time that the shooter began accelerating and a 1 or 0 to indicate whether the shot was successful or not. A shot was successful only if it went directly from the front of the shooter into the goal as illustrated in

Figure 3(b): a trial was halted unsuccessfully if the ball hit any corner or side of the shooter, or if the ball hit any wall other than the goal.

Running 2990 trials in this manner gave us sufficient training data to learn to shoot a moving ball into the goal. The success rate using the random shooting policy was 19.7%. In particular, only 590 of the training examples were positive instances.

4.1.3 Training

Using this data, we were able to train a NN that the shooter could use as a part of a learned shooting policy that enabled it to score consistently. We tried several configurations of the neural network, settling on one with a single layer of 2 hidden units and a learning rate of .01. Each layer had a bias unit with constant input of 1.0. We normalized all other inputs to fall roughly between 0.0 and 1.0 (see Figure 5(b)). The target outputs were .9 for positive examples (successful trials) and .1 for negative examples. Weights were all initialized randomly between -0.5 and 0.5. The resulting neural network is pictured in Figure 5(b). This neural network was not the result of an exhaustive search for the optimal configuration to suit this task, but rather the quickest and most successful of about 10 alternatives with different numbers of hidden units and different learning rates. Once settling on this configuration, we never changed it, concentrating instead on other research issues.

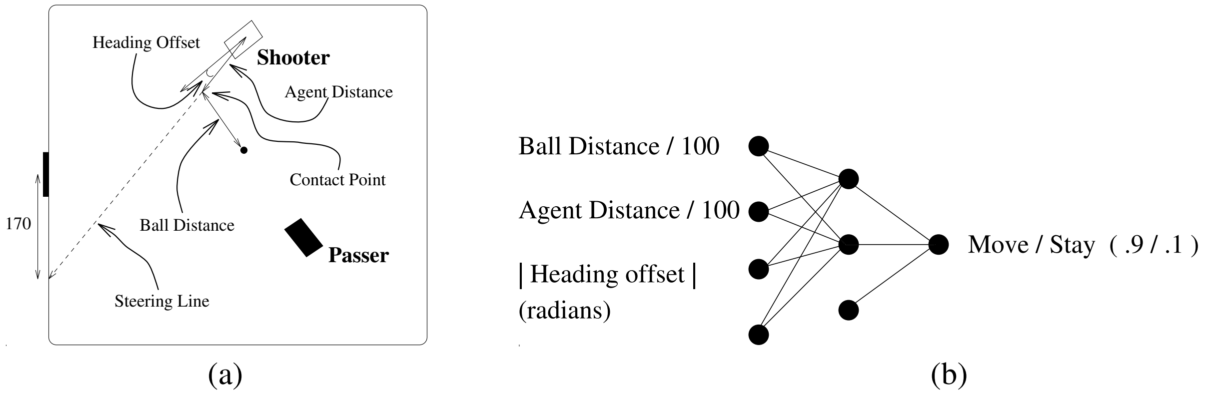


Figure 5: **(a)** The predicates we used to describe the world for the purpose of learning to shoot a moving ball are illustrated. **(b)** The neural network used to learn the shooting policy. The neural network has 2 hidden units with a bias unit at both the input and hidden layers.

Training this neural network on the entire training set for 3000 epochs resulted in a mean squared error of .0386 and 253 of the examples misclassified (i.e. the output was closer to the wrong output than the correct one). Training for more epochs did not help noticeably. Due to the sensor noise during training, the concept was not perfectly learned.

4.1.4 Testing

After training, a neural network could be used very cheaply—just a single forward pass at each decision point—to decide when to begin accelerating. Notice that within a single trial, only one input to the neural network varied: the Ball Distance decreased as the ball approached the Contact Point. Thus, the output of the neural network tended to vary fairly regularly. As the ball began approaching, the output began increasing first slowly and then sharply. After reaching its peak, the output began decreasing slowly at first and then sharply. The optimal time for the shooter to begin accelerating was at the peak of this function, however since the function peaked at different values on different trials, we used the following *3-input neural network shooting policy*:

- Begin accelerating when Output $\geq$ .6 AND Output $<$ Previous output - .01.

Requiring that Output $\geq$ .6 ensured that the shooter would only start moving if it "believed" it was more likely to score than to miss. Output $<$ Previous output - .01 became true when the output of the neural network was just past its peak. Requiring that the output decrease by at least .01 from the previous output ensured that the decrease was not due simply to sensor noise.

Using this learned 3-input neural network shooting policy, the shooter scored **96.5%** of the time. The results reported in this section are summarized in Table 2.

Initial Shooter Position	Shooting Policy	Success (%)
Constant	Simple	100
Varying	Simple	60.8
Varying	Random	19.7
Varying	3-input NN	**96.5**

Table 2: Results before and after learning for fixed ball motion.

Even more important than the high success rate achieved when using the learned shooting policy was the fact that the shooter achieved the same success rate in each of the four symmetrical reflections of the training situation (the four *action quadrants*). **With no further training, the shooter was able to score from either side of the goal on either side of the field.** Figure 4(a) illustrates one of the three symmetrical scenarios. The world description used as input to the neural network contained no information specific to the location on the field, but instead captured only information about the relative positions of the shooter, the ball, and the goal. Thanks to these flexible inputs, training in one situation was applicable to several other situations.

4.2 Varying the Ball's Speed

Encouraged by the success and flexibility of our initial solution, we next varied a parameter of the initial setup to test if the solution would extend further. Throughout Section 4.1, the passer started 35 units away from the ball and accelerated full-speed ahead until striking it. This process consistently propelled the ball at about 135 units/sec. To make the task more challenging, we varied the ball's speed by starting the passer randomly within a range of 32–38 units away from the ball. Now the ball would travel towards the shooter with a speed of 110–180 units/sec.

Before making any changes to the shooter's shooting policy, we tested the policy trained in Section 4.1 on the new task. However, we found that the 3-input neural network was not sufficient to handle the varying speed, giving a success rate of only 49.1% due to the mistiming of the acceleration. In order to accommodate for this mistiming, we added a fourth input to the neural network in order to represent the speed of the ball: *Ball Speed*

The shooter computed the Ball Speed from the ball's change in position over a given amount of time. Due to sensor noise, the change in position over a single time slice did not give an accurate reading. On the other hand, since the ball slowed down over time, the shooter could also not take the ball's total change in position over time. As a compromise, the shooter computed the Ball Speed from the ball's change in position over the last 10 time slices (or fewer if 10 positions had not yet been observed).

To accommodate for the additional quantity used to describe a world state, we gathered new training data. As before, the shooter used the random shooting policy during the training trials. This time, however, each training instance consisted of *four* inputs describing the state of the world plus an output indicating whether or not the trial was a successful one. Of the 5737 samples gathered for training, 963—or 16.8%—were positive examples.

For the purposes of training the new neural network, we scaled the Ball Speed of the ball to fall between 0.0 and 1.0: $\frac{speed-110}{70}$. Except for the fourth input, the new neural network looked like the one pictured in Figure 5(b). It had two hidden units, a bias unit at each level, and a learning rate of .001. Training this neural network for 4000 epochs resulted in a mean squared error of .0512 with 651 of the instances misclassified.

Using this new neural network and the same decision function over its output as before, or the *4-input neural network shooting policy*, our shooter was able to score **91.5%** of the time with the ball moving at different speeds. **Again, this success rate was observed in all four action quadrants.** The results from this section are summarized in Table 3.

Shooting Policy	Success (%)
3-input NN	49.1
Random	16.8
4-input NN	**91.5**

Table 3: When the Ball Speed varies, an additional input is needed.

4.3 Varying the Ball's Trajectory

To this point, we have shown that the inputs we used to train our neural network have allowed training in a particular part of the field to apply to other parts of the field. In this section we describe experiments that show that the 4-input neural network shooting policy described in Section 4.2 is flexible still further. **All experiments in this section use exactly the same shooting policy with no further retraining.**

Since the inputs are all relative to the Contact Point, we predicted that the trajectory with which the ball approached the shooter would not affect the performance of the neural network adversely. In order to test this hypothesis we changed the initial positions of the ball and the passer so that the ball would cross the shooter's path with a different heading (90^o as opposed to 140^o). This variation in the initial setup is illustrated in Figure 6. The ball's speed still varied as in Section 4.2.

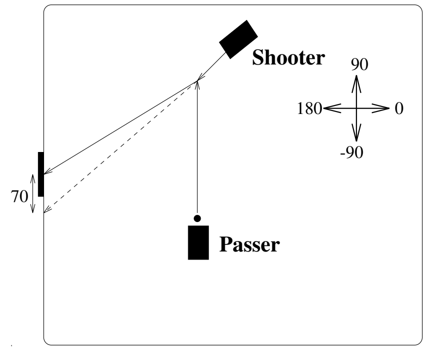


Figure 6: When the ball approached with a trajectory of 90^o instead of 140^o, the shooter had to change its aiming policy to aim 70 units wide of the goal instead of 170 units wide.

Indeed, the shooter was able to consistently make contact with the ball and redirect it towards the goal. However, it never scored because its steering line was aiming it too wide of the goal. Due to the ball's changed trajectory, aiming 170 units wide of the goal was no longer an appropriate aiming policy.

With the ball always coming with the same trajectory, we could simply change the steering line such that the shooter was aiming 70 units wide of the center of the goal. Doing so led to a success rate of **96.3%**—even better than the actual training situation. This improved success rate can be accounted for by the fact that the ball was approaching the agent more directly so that it was slightly easier to hit. Nonetheless, it remained a difficult task, and it was successfully accomplished with no retraining. **Not only did our learned shooting policy generalize to different areas of the field, but it also generalized to different ball trajectories.**

Having used the same shooting policy to successfully shoot balls moving at 2 different trajectories, we knew that we could vary the ball's trajectory over a continuous range and still use the same policy to score. The only problem would be altering the shooter's aiming policy. Thus far we had chosen the steering line by hand, but we did not want to do that for every different possible trajectory of the ball. In fact, this problem gave us the opportunity to put the principal espoused by this article into use once again.

For the experiments described in the remainder of this section, the ball's initial trajectory ranged randomly over 63^o ($82^o - 145^o$). Figure 4(b) illustrates this range. The policy used by the shooter to decide when to begin accelerating was exactly the same one learned in Section 4.2 with no retraining of the neural network. On top of this neural network, we added a new one to determine the direction in which the shooter should steer. The steering line determined by this direction and the shooter's current position could then be used to compute the predicates needed as input to the original neural network.

For this new neural network we again chose parameters that would allow it to generalize beyond the training situation. The new input we chose was the angle between the ball's path and the line connecting the shooter's initial position with the center of the goal: the *Ball-Agent Angle*. The output was the *Angle Wide* of this second line that the shooter should steer. These quantities are illustrated in Figure 7(a). We found that performance improved slightly when we added as a second input an estimate of the ball's speed. However, since the neural network in the 4-input neural network shooting policy could only be used after the shooter determined the steering line, we used an estimate of Ball Speed after just two position readings. Then, the shooter's steering line was chosen by the learned aiming policy and the 4-input neural network shooting policy could be used to time the shot.

To gather training data for the new neural network, we ran many trials with the ball passed with different speeds and different trajectories while the Angle Wide was set to a random number between 0.0 and 0.4 radians. Using only the 783 positive examples, we then trained the neural network to be used for the shooters' learned aiming policy. Again, the inputs and the outputs were scaled to fall roughly between 0.0 and 1.0, there were two hidden units and two bias units, the weights were initialized randomly, and the learning rate was .01. The resulting neural network is pictured in Figure 7(b). Training this neural network for 4000 epochs gave a mean squared error of .0421.

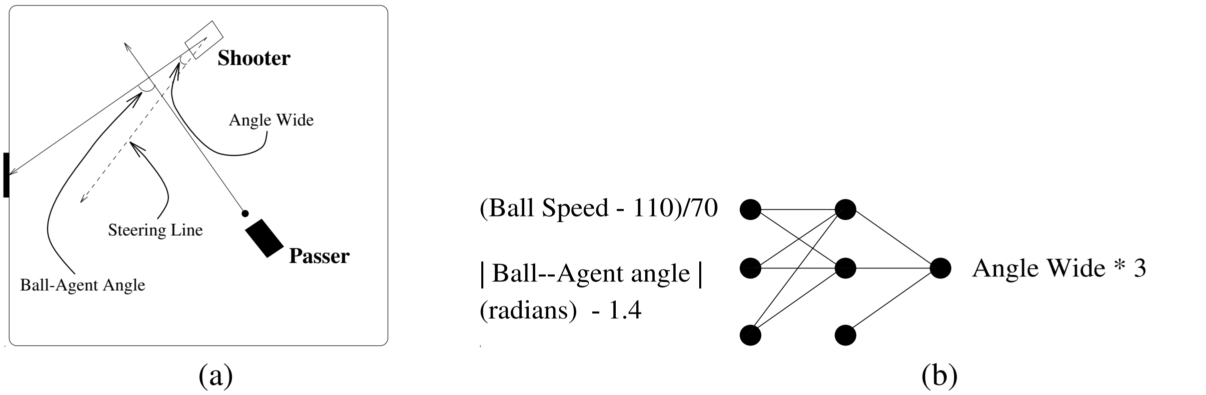


Figure 7: **(a)** The predicates we used to describe the world for the purpose of learning an aiming policy. **(b)** The neural network used to learn the aiming policy. The neural network has 2 hidden units with a bias unit at both the input and hidden layers.

Using this *2-input neural network aiming policy* to decide where to aim and the old 4-input neural network shooting policy to decide when to accelerate, the shooter scored **95.4%** of the time while the ball's speed and trajectory varied. Using a neural network with just 1 input (omitting the speed estimate) and one hidden unit, or the *1-input neural network aiming policy*, gave a success rate of 92.8%. Being satisfied with this performance, we did not experiment with other neural network configurations.

4.4 Moving the Goal

To test if the learned aiming policy could generalize beyond its training situation, we then moved the goal by a goal-width both down and up along the same side of the field (see Figure 4(c)). One can think of this variation as the shooter aiming for different parts of a larger goal (see Section 5.2). Changing nothing but the shooter's knowledge of where the goal was located, the shooter scored **84.1%** of the time on the lower goal and **74.9%** of the time on the upper goal (see Table 4). The discrepancy between these values can be explained by the greater difficulty of shooting a ball in a direction close to the direction that it was already travelling. As one would expect, when the initial situation was flipped so that the shooter began in the lower quadrant, these two success rates also flipped: the shooter scored more frequently when shooting at the higher goal. Notice that in this case, the representation of the output was as important as the inputs for generalization beyond the training situation.

Aiming Policy	Goal Position	Success (%)
1-input NN	Middle	92.8
2-input NN	Middle	**95.4**
2-input NN	Lower by one goal-width	**84.1**
2-input NN	Higher by one goal-width	**74.9**

Table 4: When the ball's trajectory varied, a new aiming policy was needed. Results are reasonable even when the goal is moved to a new position.

5 Higher Level Multiagent Extensions

Shooting a moving ball is crucial for high-level collaborative and adversarial action in the soccer domain. The learned shooting behavior is robust in that it works for different ball speeds and trajectories, and it is situation independent in that it works for different action quadrants and goal locations. These qualities enable it to be used as a basis for higher level behaviors.

Although soccer players must first learn low-level skills, soccer is inherently a strategic, high-level task: a team of the best-skilled individual players in the world would be easily beaten if they could not work together. Similarly, an unskilled team that works together to exploit a weakness of a better team, will often be able to prevail.

This section describes how the robust shooting template developed in Section 4 can be built upon in both collaborative and adversarial directions. In particular, from a collaborative standpoint, the learned shooting skill can be used by the passer as well. Rather than passing a stationary ball to the shooter, the passer can redirect a moving ball in exactly the same way as the shooter, only aiming at a point in front of the shooter instead of at the goal. On the other hand, adversarial issues can be studied by introducing a defender which tries to block the shooter's attempts.

5.1 Cooperative Learning

For the task of shooting a moving ball, the passer's behavior was predetermined: it accelerated as fast as it could until it hit the ball. We varied the velocity (both speed and trajectory) of the ball by simply starting the passer and the ball in different positions.

However, in a real game, the passer would rarely have the opportunity to pass a stationary ball that is placed directly in front of it. It too would have to learn to deal with a ball in motion. In particular, the passer would need to learn to pass the ball in such a way that the shooter could have a good chance of putting the ball in the goal (see Figure 8).

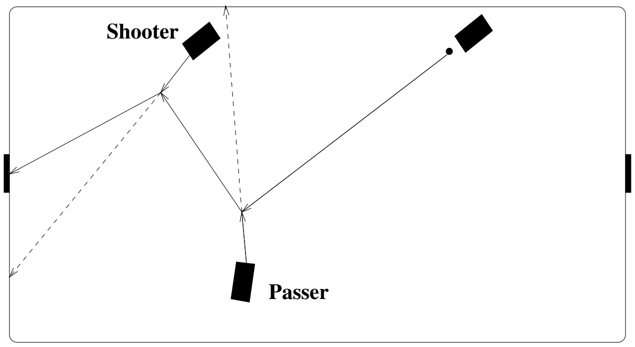


Figure 8: A collaborative scenario: the passer and the shooter must both learn their tasks in such a way that they can interact successfully.

Our approach to this problem is to use the low-level template learned in Section 4 for both the shooter and the passer. By fixing this behavior, the agents can learn an entirely new behavior level without worrying about low-level execution. Notice that once the passer and shooter have learned to cooperate effectively, any number of passes can be chained together. The receiver of a pass (in this case, the "passer") simply must aim towards the next receiver in the chain (the "shooter").

The parameters to be learned by the passer and the shooter are the point at which to aim the pass and the point at which to position itself respectively. While in Section 4 the shooter had a fixed goal at which to aim, here the passer's task is not as well-defined. Its goal is to redirect the ball in such a way that the shooter has the best chance of hitting it. Similarly, before the ball is passed, the shooter must get itself into a position that gives the passer the best chance at executing a good pass.

Figure 9 illustrates the inputs and outputs of the passer's and shooter's behavior in this collaborative scenario. Based on the *Passer-Shooter Angle*, the passer must choose the *Lead Distance*, or the distance from the shooter (on the line connecting the shooter to the goal) at which it should aim the pass.[3] Notice that the input to the passer's learning function can be manipulated by the shooter before the passer aims the pass. Based on the passer's relative position to the goal, the shooter can affect the Passer-Shooter Angle by positioning itself appropriately. Since the inputs and outputs to these tasks are similar to the task learned in Section 4, similar neural network techniques can be used.

Since both the passer and the shooter are learning, this scenario satisfies both aspects of Weiß's definition of multiagent learning: more than one agent is learning in a situation where multiple agents are necessary. The phenomenon of the different agents' learning parameters interacting directly is one that is common among multiagent systems. The novel part of this approach is the layering of a learned multiagent behavior on top of another.

Continuing up yet another layer, the passing behavior can be incorporated and used when a player is faced with the decision of which teammate to pass to. Chaining passes as described above

[3]The point at which the passer should aim the pass is analogous to the goal location in Section 4. The passer can still choose its own trajectory in the same way as in Section 4.

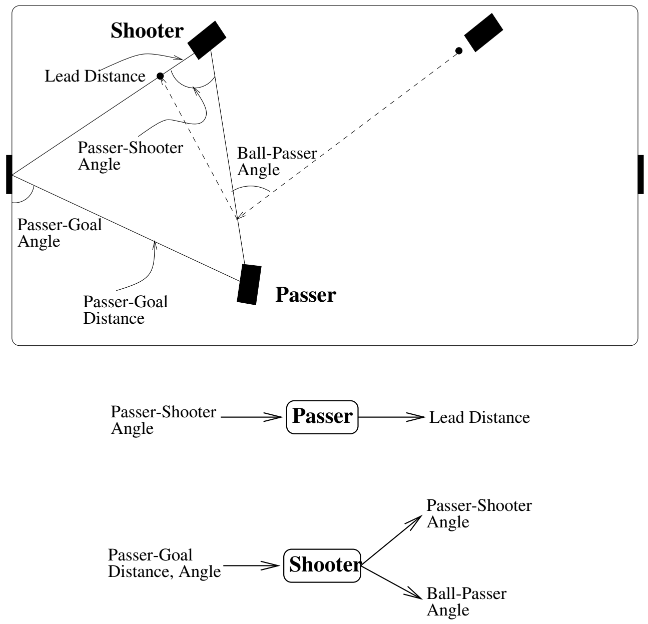


Figure 9: The parameters of the learning functions for the passer and the shooter.

assumes that each player knows where to pass the ball next. However, in a game situation, given the positioning of all the other players on the field, the receiver of a pass must choose where it should send the ball next. In a richer and more widely used simulator environment (Noda, 1995), the authors have used decision tree learning to enable a passer to choose from among possible receivers in the presence of defenders (Stone & Veloso, 1998). In addition, in this current simulator, they successfully reimplemented the neural network approach to learning to intercept a moving ball as described in this article.

5.2 Adversarial Learning

At the same time as teammates are cooperating and passing the ball amongst themselves, they must also consider how best to defeat their opponents. The shooting template developed in Section 4 can be incorporated into an adversarial situation by adding a defender.

In Section 4, the shooter was trained to aim at different goal locations. The small goal was used to train the shooter's accuracy. However, in competitive situations, the goal must be larger than a player so that a single player cannot entirely block the goal. In order to make this adversarial situation fair, the goal is widened by 3 times (see Figure 10).

As in the collaborative case, our approach to this problem involves holding the learned shooting template fixed and allowing the clients to learn behaviors at a higher level. Based on the defender's position, the shooter must choose to shoot at the upper, middle, or lower portion of the goal. Meanwhile, based on the shooter's and the ball's positions as well as its own current velocity, the defender must choose in which direction to accelerate. For simplicity, the defender may only set it's throttle to full forward, full backwards, or none. Since the outputs of both the shooter's

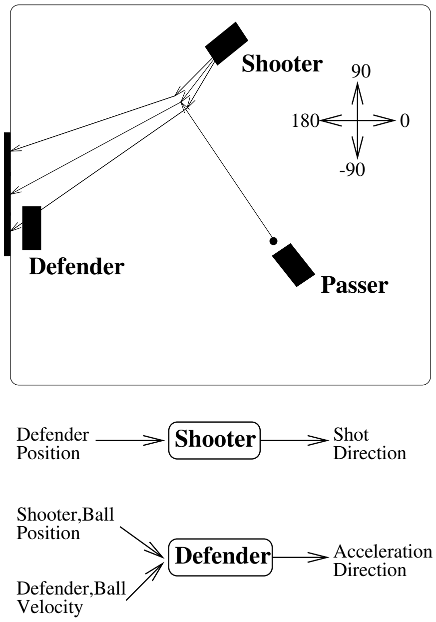


Figure 10: An adversarial scenario: the defender learns to block the shot, while the shooter simultaneously tries to learn to score.

and the defender's learning functions are discrete, Reinforcement Learning techniques, such as Q-learning (Kaelbling, Littman, & Moore, 1996), are possible alternatives to neural networks.

The defender may be able to learn to judge where the shooter is aiming before the shooter strikes the ball by observing the shooter's approach. Similarly, if the defender starts moving, the shooter may be able to adjust and aim at a different part of the goal. Thus as time goes on, the opponents will need to co-evolve in order to adjust to each other's changing strategies. Note that a sophisticated agent may be able to influence the opponent's behavior by acting consistently for a period of time, and then drastically changing behaviors so as to fool the opponent.

At the next higher level of behavior, this adversarial scenario can be extended by combining it with the collaborative passing behavior. When receiving the ball, the passer (as in Figure 8) could decide whether to pass the ball or shoot it immediately, based on the defender's motion. This extra option would further complicate the defender's behavior. Some related results pertaining to the decision of whether to pass or shoot appear in (Stone & Veloso, 1996).

6 Discussion and Conclusion

Robotic Soccer is a rich domain for the study of multiagent learning issues. There are opportunities to study both collaborative and adversarial situations. However, in order to study these situations, the agents must first learn some basic behaviors that are necessitated by the multiagent nature of the domain. Similar to human soccer players, they can first learn to make contact with a moving ball, then learn to aim it, and only then start thinking about trying to beat an opponent and about team-level strategies.

This article presents a robust, low-level learned behavior and presents several ways in which it can be extended to and incorporated into collaborative and adversarial situations. Our ongoing research agenda includes improving the low-level behaviors while simultaneously working on the presented collaborative and adversarial learning issues. The goal is to create high-level learned strategic behaviors by continuing to layer learned behaviors.

References

Achim, S., Stone, P., & Veloso, M. (1996). Building a dedicated robotic soccer system. In *Proceedings of the IROS-96 Workshop on RoboCup*.

Asada, M., Noda, S., Tawaratsumida, S., & Hosoda, K. (1994a). Purposive behavior acquisition on a real robot by vision-based reinforcement learning. In *Proc. of MLC-COLT (Machine Learning Confernce and Computer Learning Theory) Workshop on Robot Learning*, pp. 1–9.

Asada, M., Uchibe, E., Noda, S., Tawaratsumida, S., & Hosoda, K. (1994b). Coordination of multiple behaviors acquired by vision-based reinforcement learning. In *Proc. of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems 1994 (IROS '94)*, pp. 917–924.

Clouse, J. A. (1996). Learning from an automated training agent. In Weiß, G., & Sen, S. (Eds.), *Adaptation and Learning in Multiagent Systems*. Springer Verlag, Berlin.

Ford, R., Boutilier, C., & Kanazawa, K. (1994). Exploiting natural structure in reinforcement learning: Experience in robot soccer-playing.. Unpublished Manuscript.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kitano, H., Kuniyoshi, Y., Noda, I., Asada, M., Matsubara, H., & Osawa, E. (1997). RoboCup: A challenge problem for AI. *AI Magazine*, *18*(1), 73–85.

Latombe, J.-C. (1991). A fast path planner for a car-like indoor mobile robot. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 659–665.

Matsubara, H., Noda, I., & Hiraki, K. (1996). Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pp. 63–67 Menlo Park,CA. AAAI Press. AAAI Technical Report SS-96-01.

Noda, I. (1995). Soccer server : a simulator of RoboCup. In *Proceedings of AI symposium '95*, pp. 29–34. Japanese Society for AI.

Press, W. H. (1988). *Numerical Recipes in C: the art of scientific computing*, pp. 710–713. Cambridge University Press, Cambridge.

Reeds, J. A., & Shepp, R. A. (1991). Optimal paths for a car that goes both forward and backward.. *Pacific Journal of Mathematics*, *145(2)*, 367–393.

Sahota, M. (1994). Personal correspondence..

Sahota, M. K. (1993). Real-time intelligent behaviour in dynamic environments: Soccer-playing robots.. Master's thesis, University of British Columbia.

Sahota, M. K. (1996). Dynasim user guide. http://www.cs.ubc.ca/nest/lci/soccer.

Sahota, M. K., Mackworth, A. K., Barman, R. A., & Kingdon, S. J. (1995). Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 3690–3663.

Stone, P., & Veloso, M. (1996). Beating a defender in robotic soccer: Memory-based learning of a continuous function. In Touretzky, D. S., Mozer, M. C., & Hasselmo, M. E. (Eds.), *Advances in Neural Information Processing Systems 8*, pp. 896–902 Cambridge, MA. MIT press.

Stone, P., & Veloso, M. (1997). Multiagent systems: A survey from a machine learning perspective. Tech. rep. CMU-CS-97-193, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Stone, P., & Veloso, M. (1998). A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence*, *12*.

Stone, P., Veloso, M., & Achim, S. (1996). Collaboration and learning in robotic soccer. In *Proceedings of the Micro-Robot World Cup Soccer Tournament* Taejon, Korea. IEEE Robotics and Automation Society.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pp. 330–337.

Weiß, G. (1995). Distributed reinforcement learning. *Robotics and Autonomous Systems*, *15*, 135–142.

Zeng, D., & Sycara, K. (1996). Bayesian learning in negotiation. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pp. 99–104 Menlo Park,CA. AAAI Press. AAAI Technical Report SS-96-01.

A Simulator Physics

The physics of our simulator are based largely on those embedded in the simulator used by the the Dynamo group (Sahota, 1993) with some minor adjustments. The ball has a radius of 5.0 units, mass of 0.333 units, and a drag of 30 units/sec^2. This value of drag is typical for a ping-pong ball (30 cm/sec^2).

The agents are all 40.0 units long by 20.0 units wide with a mass of 10.0 units. The agent's position is updated according to the steering and throttle commands it receives using the fourth-order Runge-Kutta formula (Press, 1988).

Collisions between objects in the world are handled as follows. The ball bounces off walls elastically and stops if it enters a goal, while agents stop if they collide with one another or with a wall. Collisions between the ball and an agent take into account momenta of the colliding objects as well as the corners on the agents.

B Line Following

For the clients in this article, the only path planning needed is the ability to steer along a straight line. Given a line and periodically its own position, the client's goal is to eventually be travelling precisely parallel to the line within a certain distance from the line. This distance, or the *tolerance*, is a parameter, but throughout this article it is set to 5 units. Since the agent's width is 20 units, this tolerance allows the client to strike the ball as expected.

The client begins by computing its distance from the line as well as its offset from the correct heading (the slope of the line). If the client's distance is greater than the tolerance, then the agent steers sharply towards the line. In particular, it sets its steering in proportion to its distance from the line minus the tolerance. If the distance is no more than the tolerance (or once it has become so), then the client steers towards the line with a force proportional to its heading offset. Thus, when the client is heading parallel to the line, its wheel is centered.

- *DistanceFromLine* = the perpendicular distance from the center of the client to the target line.
- *HeadingOffset* = the client's heading − the line's slope.
- if (*DistanceFromLine* > *tolerance*)
  Set steering angle towards the line in proportion to *DistanceFromLine*.
- else (*DistanceFromLine* ≤ *tolerance*)
  Set steering angle towards the line in proportion to the *HeadingOffset*.

Although this method does not force the client to travel exactly along the given line, it does allow the client to be moving in precisely the right direction. Thus it is able to reliably strike the ball in a particular direction.