Keepaway Soccer: A Machine Learning Testbed

Peter Stone and Richard S. Sutton

AT&T Labs — Research
180 Park Ave.
Florham Park, NJ 07932
{pstone,sutton}@research.att.com
http://www.research.att.com/{∼pstone,∼sutton}

Abstract. RoboCup simulated soccer presents many challenges to machine learning (ML) methods, including a large state space, hidden and uncertain state, multiple agents, and long and variable delays in the effects of actions. While there have been many successful ML applications to portions of the robotic soccer task, it appears to be still beyond the capabilities of modern machine learning techniques to enable a team of 11 agents to successfully learn the full robotic soccer task from sensors to actuators. Because the successful applications to portions of the task have been embedded in different teams and have often addressed different subtasks, they have been difficult to compare. We put forth keepaway soccer as a domain suitable for directly comparing different machine learning approaches to robotic soccer. It is complex enough that it can't be solved trivially, yet simple enough that complete machine learning approaches are feasible. In keepaway, one team, "the keepers," tries to keep control of the ball for as long as possible despite the efforts of "the takers." The keepers learn individually when to hold the ball and when to pass to a teammate, while the takers learn when to charge the ball-holder and when to cover possible passing lanes. We fully specify the domain and summarize some initial, successful learning results.

1 Introduction

RoboCup simulated soccer has been used as the basis for successful international competitions and research challenges [4]. It is a fully *distributed*, *multiagent* domain with both *teammates* and *adversaries*. There is *hidden state*, meaning that each agent has only a partial world view at any given moment. The agents also have *noisy sensors and actuators*, meaning that they do not perceive the world exactly as it is, nor can they affect the world exactly as intended. In addition, the perception and action cycles are *asynchronous*, prohibiting the traditional AI paradigm of using perceptual input to trigger actions. *Communication* opportunities are limited, and the agents must make their decisions in *real-time*. These italicized domain characteristics combine to make simulated robotic soccer a realistic and challenging domain [10].

In principle, modern machine learning methods are reasonably well suited to meeting the challenges of RoboCup simulated soccer. For example, reinforcement learning is all about sequential decision making, achieving delayed goals, and handling noise and stochasticity. However, RoboCup soccer is a large and difficult instance of many of the issues which have been addressed in small, isolated cases in previous reinforcement learning research. Despite substantial previous work (e.g., [1, 13, 15, 9]), the extent to which modern reinforcement learning methods can meet these challenges remains an open question.

Similarly, evolutionary learning methods are promising candidates for learning complex behaviors. Indeed, there have been two previous attempts to learn the entire simulated RoboCup task via genetic programming (GP) [5, 2]. While both efforts were initially intended to test the ability of GP to scale to the full, cooperative robotic soccer task, the first system ended up evolving over hand-coded low-level behaviors, and the second achieved some successful individual behaviors but was unable to generate many collaborative team behaviors. Whether this approach can be scaled up to produce more successful teams remains to be seen.

One frustration with these and other machine learning approaches to RoboCup is that they are all embedded within disparate systems, and often address different subtasks of the full soccer problem. Therefore, they are difficult to compare in any meaningful way.

In this paper, we put forth keepaway soccer as a subtask of robotic soccer that is suitable for directly comparing different machine learning methods. The remainder of the paper is organized as follows. In Section 2 we describe keepaway and specify the sensors and actuators for this task. In Section 3 we summarize the successful results of our reinforcement learning approach to this task [12]. Section 4 concludes.

2 Keepaway Soccer

We consider a subtask of RoboCup soccer, *keepaway*, in which one team, the *keepers*, is trying to maintain possession of the ball within a limited region, while another team, the *takers*, is trying to gain possession. Whenever the takers take possession or the ball leaves the region, the *episode* ends and the players are reset for another episode (with the keepers being given possession of the ball again).

Parameters of the task include the size of the region, the number of keepers, and the number of takers. Figure 1 shows a screen shot of 3 keepers and 2 takers (called 3 vs. 2, or 3v2 for short) playing in a 20m x 20m region.

All of our work uses version 5.25 of the standard RoboCup soccer simulator [6] but is directly applicable to more recent versions. An omniscient coach agent manages the play, ending episodes when a taker gains possession of the ball for a set period of time or when the ball goes outside of the region. At the beginning of each episode, the coach resets the location of the ball and of the players semi-randomly within the region of play. The takers all start in one corner (bottom left). Three randomly chosen keepers are placed one in each of the three

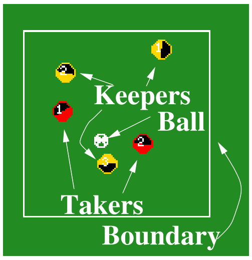

Fig. 1. A screen shot from a 3 vs. 2 keepaway episode in a 20m x 20m region.

remaining corners, and any keepers beyond three are placed in the center of the region. The ball is initially placed next to the keeper in the top left corner.

In the RoboCup soccer simulator, agents typically have limited and noisy sensors: each player can see objects within a 90^o view cone, and the precision of an object's sensed location degrades with distance. To simplify the problem, we gave our agents (both keepers and takers) a full 360^o of vision. We have not yet determined whether this simplification was necessary or helpful.

Keepaway is a subproblem of robotic soccer. The principal simplifications are that there are fewer players involved; they are playing in a smaller area; and the players are always focused on the same high-level goal—they don't need to balance offensive and defensive considerations. Nevertheless, the skills needed to play keepaway well are also very useful in the full problem of robotic soccer. Indeed, ATT-CMUnited-2000—the 3rd-place finishing team in the RoboCup-2000 simulator league—incorporated a successful hand-coded solution to an 11 vs. 11 keepaway task [11].

Keepaway is a complex enough task that simple, hand-coded solutions are not likely to yield the best behaviors. However, it is also simple enough that complete machine learning approaches are feasible. These complete approaches can then be directly compared. In order to do so, it is necessary that the different systems work with similar action and feature spaces. In the remainder of this section, we specify the actions and state variables in our formulation of the task.

2.1 State Variables and Actions

In this learning problem, the learners choose from among mid-level actions constructed from skills. Those skills include

HoldBall(): Remain stationary while keeping possession of the ball in a position that is as far away from the opponents as possible.

PassBall(k): Kick the ball directly towards keeper k.

GetOpen(): Move to a position that is free from opponents and open for a pass from the ball's current position (using SPAR [16]).

GoToBall(): Intercept a moving ball or move directly towards a stationary ball.

BlockPass(k): Move to a position between the keeper with the ball and keeper k.

All of these skills except PassBall(k) are simple functions from state to a corresponding action; an invocation of one of these normally controls behavior for a single time step. PassBall(k), however, requires an extended sequence of actions—getting the ball into position for kicking, and then a sequence of kicks in the desired direction [10]; a single invocation of PassBall(k) influences behavior for several time steps. Moreover, even the simpler skills may last more than one time step because the player occasionally misses the step following them; the simulator occasionally misses commands; or the player may find itself in a situation requiring it to take a specific action, for instance to self-localize. In these cases there is no new opportunity for decision making until two or more steps after invoking the skill.

Such possibilities may present problems for machine learning approaches. One candidate for dealing with them is to treat the problem as a *semi-Markov* decision process, or SMDP [7, 3]. An SMDP evolves in a sequence of jumps from the initiation of each SMDP action to its termination one or more time steps later, at which time the next SMDP action is initiated. SMDP actions that consist of a subpolicy and termination condition over an underlying decision process, as here, have been termed *options* [14].

2.2 Keeper State Variables

Our task specification involves learning by the keepers when in possession[1] of the ball. Keepers not in possession of the ball are required to select the Receive option:

Receive: If a teammate possesses the ball, or can get to the ball faster than this keeper can, invoke GetOpen() for one step; otherwise, invoke GoToBall() for one step. Repeat until a keeper has possession of the ball or the episode ends.

A keeper in possession, on the other hand, is faced with a genuine choice. It may hold the ball, or it may pass to one of its teammates. That is, it chooses an option from {Holdball, PassK_2ThenReceive, PassK_3ThenReceive, ..., PassK_nThenReceive} where the Holdball option simply executes HoldBall() for one step (or more if, for example, the server misses the next step) and the Pass$k$ThenReceive options involve passes to the other keepers. The keepers are numbered by their closeness to the keeper with the ball: K_1 is the keeper with the ball, K_2 is the closest keeper to it, K_3 the next closest, and so on up to K_n, where n is the number of keepers. Each Pass$k$ThenReceive is defined as

Pass$k$ThenReceive: Invoke PassBall(k) to kick the ball toward teammate k. Then behave and terminate as in the Receive option.

[1] "Possession" in the soccer simulator is not well-defined because the ball never occupies the same location as a player. One of our agents considers that it has possession of the ball if the ball is close enough to kick it.

The keepers' learning process thus searches a constrained policy space characterized only by the choice of option when in possession of the ball. Examples of (non-learned) policies within this space include:

Random: Choose randomly among the n options, each with probability $\frac{1}{n}$.

Hold: Always choose HoldBall()

Hand-coded:

If no taker is within 10m, choose HoldBall();
Else If teammate k is in a better location than the keeper with the ball and the other teammates, and the pass is likely to succeed (using the CMUnited-99 pass-evaluation function, which is trained off-line using the C4.5 decision tree training algorithm [8]), then choose PassBall(k);
Else choose HoldBall().

We turn now to the representation of state used by the keepers. Note that in our formulation the state variables are only needed when one of the keepers is in possession of the ball (were keepers to learn where to move when they don't have the ball, additional state variables would be needed). On these steps the keeper determines a set of state variables, computed based on the positions of: the keepers K_1–K_n, ordered as above; the takers T_1–T_m (m is the number of takers), ordered by increasing distance from K_1; and C, the center of the playing region (see Figure 2 for an example with 3 keepers and 2 takers). Let $dist(a, b)$ be the distance between a and b and $ang(a, b, c)$ be the angle between a and c with vertex at b. For example, $ang(K_3, K_1, T_1)$ is shown in Figure 2. With 3 keepers and 2 takers, we used the following 13 state variables:

- $dist(K_1, C)$, $dist(K_2, C)$, $dist(K_3, C)$
- $dist(T_1, C)$, $dist(T_2, C)$
- $dist(K_1, K_2)$, $dist(K_1, K_3)$
- $dist(K_1, T_1)$, $dist(K_1, T_2)$
- $\mathrm{Min}(dist(K_2, T_1), dist(K_2, T_2))$
- $\mathrm{Min}(dist(K_3, T_1), dist(K_3, T_2))$
- $\mathrm{Min}(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2))$
- $\mathrm{Min}(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2))$

This list generalizes naturally to additional keepers and takers, leading to a linear growth in the number of state variables.

2.3 Takers

The takers are relatively simple, choosing only options of minimum duration (one step, or as few as possible given server misses) that exactly mirror the available skills. When a taker has the ball, it tries to maintain possession by invoking HoldBall() for a step. Otherwise, it chooses an option that invokes one of {GoToBall(), BlockPass(K_2), BlockPass(K_3), ..., BlockPass(K_n)} for one step or as few steps as permitted by the server. In case no keeper has the ball (e.g., during a pass), K_1 is defined here as the keeper predicted to next gain possession. Examples of (non-learned) policies within the taker policy space include:

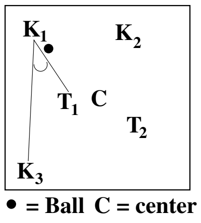

Fig. 2. The state variables used for learning with 3 keepers and 2 takers. Keepers and takers are numbered by increasing distance from K_1, the keeper with the ball.

Random-T: Choose randomly from the n options, each with probability $\frac{1}{n}$.
All-to-ball: Always choose the GoToBall() option.
Hand-coded-T:
If fastest taker to the ball, or closest or second closest taker to the ball: choose the GoToBall() option;
Else let k be the keeper with the largest angle with vertex at the ball that is clear of takers: choose the BlockPass(k) option.

The takers' state variables are similar to those of the keepers. As before, C is the center of the region. T_1 is the taker that is computing the state variables, and T_2–T_m are the other takers ordered by increasing distance from K_1. $K_i$mid is the midpoint of the line segment connecting K_1 and K_i for $i \in [2, n]$ and where the K_i are ordered based on increasing distance of $K_i$mid from T_1. That is, $\forall i, j$ s.t. $2 \leq i < j$, $dist(T_1, K_i\text{mid}) \leq dist(T_1, K_j\text{mid})$. With 3 keepers and 3 takers, we used the following 18 state variables:

- $dist(K_1, C)$, $dist(K_2, C)$, $dist(K_3, C)$
- $dist(T_1, C)$, $dist(T_2, C)$, $dist(T_3, C)$
- $dist(K_1, K_2)$, $dist(K_1, K_3)$
- $dist(K_1, T_1)$, $dist(K_1, T_2)$, $dist(K_1, T_3)$
- $dist(T_1, K_2\text{mid})$, $dist(T_1, K_3\text{mid})$
- $\text{Min}(dist(K_2\text{mid}, T_2), dist(K_2\text{mid}, T_3))$
- $\text{Min}(dist(K_3\text{mid}, T_2), dist(K_3\text{mid}, T_3))$
- $\text{Min}(ang(K_2, K_1, T_2), ang(K_2, K_1, T_3))$
- $\text{Min}(ang(K_3, K_1, T_2), ang(K_3, K_1, T_3))$
- number of takers closer to the ball than T_1

Once again, this list generalizes naturally to different numbers of keepers and takers.

3 Sample Experiments

We have conducted initial experiments in this keepaway domain using the SARSA(λ) reinforcement learning algorithm [12]. Our main results to date have focused on learning by the keepers in 3v2 keepaway in a 20x20 region. For the opponents (takers) we used the Hand-coded-T policy (note that with just 2 takers, this policy is identical to All-to-ball). To benchmark the performance of the learned keepers, we first ran the three benchmark keeper policies, Random, Hold, and Hand-coded, as laid out in Section 2.2. Average episode lengths for these three policies were 5.5, 4.8, and 5.6 seconds respectively. Figure 3 shows histograms of the lengths of the episodes generated by these policies.

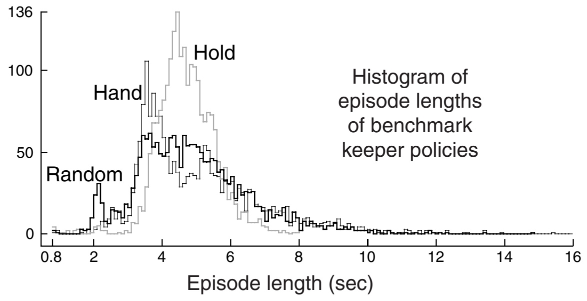

Fig. 3. Histograms of episode lengths for the 3 benchmark keeper policies in 3v2 keepaway in a 20x20 region.

We then ran a series of eleven runs with learning by the keepers against the Hand-coded-T takers. Figure 4 shows learning curves for these runs. The y-axis is the average time that the keepers are able to keep the ball from the takers (average episode length); the x-axis is training time (simulated time $\approx$ real time). The performance levels of the benchmark keeper policies are shown as horizontal lines.

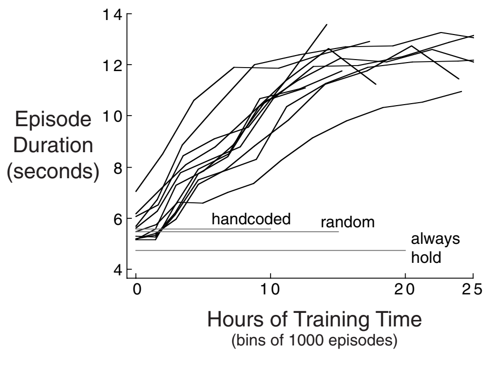

Fig. 4. Multiple successful runs under identical characteristics: 3v2 keepaway in a 20x20 region against hand-coded takers.

This data shows that we were able to learn policies that were much better than any of the benchmarks. All learning runs quickly found a much better policy than any of the benchmark policies, including the hand-coded policy. A better policy was often found even in the first data point, representing the first 1000 episodes of learning. Qualitatively, the keepers appear to quickly learn roughly how long to hold the ball, and then gradually learn fine distinctions regarding when and to which teammate to pass.

Next, we applied the same algorithm to learn policies specialized for different region sizes. In particular, we trained the takers in 15x15 and 25x25 regions without changing any of the other parameters, including the representation used for learning. The results shown in Table 3 indicate that the learned policies again outperformed all of the benchmarks. In addition, it is apparent that policies trained in the same size region as they are tested (shown in boldface), performed better than policies trained in other region sizes. In general it is easier for the keepers to keep the ball in a larger field since the takers have further to run. Thus, we observe a general increase in possession time from left to right in the table. These results indicate that different policies are best on different field sizes. Thus, were we to take the time to laboriously fine-tune a hand-coded behavior, we would need to repeat the process on each field size. On the other hand, the same learning mechanism is able to find successful policies on all three field sizes without any additional human effort.

		Testing Field Size		
Keepers		15x15	20x20	25x25
Trained on field of size	15x15	**11.0**	9.8	7.2
	20x20	10.7	**15.0**	12.2
	25x25	6.3	10.4	**15.0**
Benchmarks	Hand	4.3	5.6	8.0
	Hold	3.9	4.8	5.2
	Random	4.2	5.5	6.4

Table 1. Performance (possession times) of keepers trained with various field sizes (and benchmark keepers) when tested on fields of various sizes.

Finally, we applied our learning algorithm to learn policies for a slightly larger task, 4 vs. 3 keepaway. Figure 5 shows that the keepers learned a policy that outperformed all of our benchmarks in 4v3 keepaway in a 30x30 region. In this case, the learning curves still appear to be rising after 40 hours: more time may be needed to realize the full potential of learning.

We have also begun exploring taker learning. As noted in Section 2.3, the representation used for the keepers may not be suited to taker learning. Nonetheless, using this representation, takers were able to learn policies that outperformed the Random and All-to-ball taker policies. The best learned policies discovered so far perform roughly equivalently to the Hand-coded-T policy.

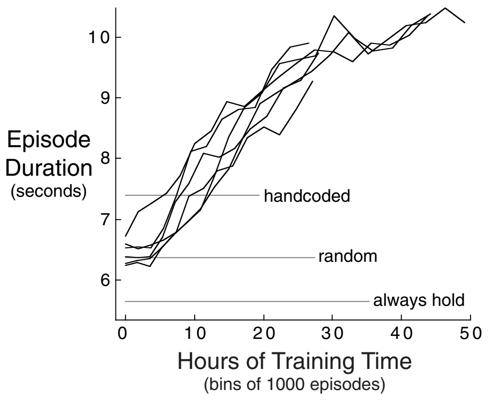

Fig. 5. Multiple successful runs under identical characteristics: 4v3 keepaway in a 30x30 region against hand-coded takers.

Our on-going research is aimed at improving the ability of the takers to learn by altering their representation and/or learning parameters. One promising line of inquiry is into the efficacy of alternately training the takers and the keepers against each other so as to improve both types of policies.

4 Conclusion

Keepaway soccer is a simplification of the full RoboCup soccer task that is suitable for solution with machine learning methods. Our previous research in the domain has demonstrated that a machine learning approach can outperform a reasonable hand-coded solution as well as other benchmark policies.

We put this domain forth as a candidate for use for direct comparisons of different machine learning techniques on a concrete task. It is complex enough that it can't be solved trivially, yet simple enough that complete machine learning approaches are feasible.

In our formulation, learning is done over a very large state space but with only a few possible actions at any given time. The domain could be incrementally expanded by introducing more actions. For example, as a first step, the agents could learn precisely where to move rather than calling functions such as GetOpen() or BlockPass(). Eventually, the agents could learn directly from the simulator low-level, parameterized action space.

References

[1] Tomohito Andou. Refinement of soccer agents' positions using reinforcement learning. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 373–388. Springer Verlag, Berlin, 1998.

[2] David Andre and Astro Teller. Evolving team Darwin United. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Verlag, Berlin, 1999.

[3] S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. pages 393–400, San Mateo, CA, 1995. Morgan Kaufmann.

[4] Hiroaki Kitano, Milind Tambe, Peter Stone, Manuela Veloso, Silvia Coradeschi, Eiichi Osawa, Hitoshi Matsubara, Itsuki Noda, and Minoru Asada. The RoboCup synthetic agent challenge 97. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 24–29, San Francisco, CA, 1997. Morgan Kaufmann.

[5] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, pages 398–411, Berlin, 1998. Springer Verlag.

[6] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12:233–250, 1998.

[7] M. L. Puterman. *Markov Decision Problems*. Wiley, NY, 1994.

[8] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[9] M. Riedmiller, A. Merke, D. Meier, A. Hoffman, A. Sinner, O. Thate, and R. Ehrmann. Karlsruhe brainstormers—a reinforcement learning approach to robotic soccer. In Peter Stone, Tucker Balch, and Gerhard Kraetszchmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*. Springer Verlag, Berlin, 2001.

[10] Peter Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

[11] Peter Stone and David McAllester. An architecture for action selection in robotic soccer. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[12] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[13] Peter Stone and Manuela Veloso. Team-partitioned, opaque-transition reinforcement learning. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*. Springer Verlag, Berlin, 1999. Also in *Proceedings of the Third International Conference on Autonomous Agents*, 1999.

[14] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063. The MIT Press, 2000.

[15] Eiji Uchibe. *Cooperative Behavior Acquisition by Learning and Evolution in a Multi-Agent Environment for Mobile Robots*. PhD thesis, Osaka University, January 1999.

[16] Manuela Veloso, Peter Stone, and Michael Bowling. Anticipation as a key for collaboration in a team of agents: A case study in robotic soccer. In *Proceedings of SPIE Sensor Fusion and Decentralized Control in Robotic Systems II*, volume 3839, Boston, September 1999.