A Scene-based Imitation Framework for RoboCup Clients

Kevin Lam and **Babak Esfandiari** and **David Tudino**
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6
babak@sce.carleton.ca

Abstract

We describe an effort to train a RoboCup soccer-playing agent playing in the Simulation League by capturing data from existing players, and using it in a real-time *scene recognition system*. When observing a simple rule-based, stateless agent, the trained player appears to imitate the behaviour of the original. Apart from some parameter selections, the process requires little human intervention.

Introduction

An agent's behaviour often consists of its reaction to a sequence of inputs over a period of time (Dousson 1996) given a spatial configuration of surrounding entities (Murakami *et al.* 2003). *Scenarios* can then be used to capture such behavior.

Our goal is to address the problem of training an agent in the RoboCup domain, as we believe that it is a good environment to experiment with agents that require spatial and temporal knowledge representation and reasoning. This research was built upon an initial project to develop a human interface agent to train a RoboCup agent from the actions of a human player through a GUI (Marlow 2004). The lack of an efficient human interface and the sheer speed of software agents are continuing challenges for such an approach.

Instead, our RoboCup agent would learn by observing another agent and imitating its behaviour. Since there is a known set of competent teams from the annual Robot World Cup games, it should ideally be possible to learn from, and subsequently emulate, existing teams, with as little human intervention as possible during the training process. Our goal isn't to win the competition but to evaluate the feasibility of quickly developing agents through this approach.

Contributions

We have developed an extensible framework for research in RoboCup agent imitation, which includes the following components:

- a conversion process for transforming raw RoboCup player logs into a simple representation for spatial situations, (or in short a *scene*)
- a set of algorithms to support scene recognition and matching using a customizable k-nearest-neighbor approach
- a RoboCup client agent based on this scene recognition algorithm

We provide results which demonstrate an agent's (limited) ability to imitate the high-level behaviour of another agent, with minimal tweaking or human intervention. These results demonstrate that it is possible to imitate the behaviour of simple spatial agents in RoboCup using a simple recognition algorithm.

The current scene-imitation agent is stateless, which imposes limits on the extent of which the algorithm can imitate other agent behaviours. This work outlines the challenges that must be addressed first, before taking on the learning of context and state-based behavior that would further improve the effectiveness of this approach.

RoboCup

The RoboCup (Robot World Cup) Simulation League (RoboCup 2006) is a unique testbed for the design of autonomous software agents, where agents represent players on a soccer team. RoboCup is an attempt to foster AI and intelligent robotics research — typical RoboCup agents collaborate with teammates, use high-level strategies such as goaltending, defense or offensive plays, and work toward a team goal while also thwarting the opponents'.

RoboCup is a real-time, dynamic and non-deterministic environment. There is a wide body of existing research to draw from, which implies much available data for data mining approaches.

Client agents in RoboCup must deal with temporal events as well as with an environment consisting of a 2-D space (the soccer field), with objects and other agents within that space. During each time period in a game, the server provides clients with world view and state information (subject to a noise model), using one of `see`, `hear`, or `sense_body` messages. Objects described in `see` messages may be players, the ball, goals, or the numerous lines and flags located on the field. Objects have attributes such as distance, direction, and identifying labels (e.g. a player's team and uniform number).

Clients then send a command to the server in response, typically to perform an action such as a `dash`, `kick`, or `turn`.

Current Approaches

RoboCup team agents span a variety of platforms, languages, algorithms and capabilities. Agent behaviours may be hard-coded such as Krislet (Langner 1999) or specified in a script or state machine (Dorer 1999), using planning (Stolzenburg *et al.* 2000) or reinforcement learning and using more complex architectures such as "layered learning" (Stone 1997) of skills and strategy.

Existing attempts at generating an agent's behaviour from direct observation such as Matsui's ILP agent (Matsui, Inuzuka, & Seki 2000) require prior processing including generation of predicates or other knowledge, and properly classified examples from which to train. Typically, agent observation is only used to match previously-generated profiles (as also used for adversarial plan recognition in (Steffens 2002)).

Initial Experiments

Initial attempts by students in our lab to learn an agent strategy (Krislet's) using only "off-the-shelf" machine learning tools demonstrated that learning methods such as decision tree learning or neural networks are sometimes sufficient to capture the high-level behaviour of a RoboCup agent, but with limitations in effectiveness and complexity — much intervention is still required to prune the learned tree and provide positive training instances to help the learning process.

Methodology

Our objective is to develop a soccer-playing RoboCup agent with the ability to draw from observations of other RoboCup agents and use this knowledge to guide its own decision-making process, all with limited or no domain expert (i.e. human) intervention. The process includes the following objectives:

1. *Perform data capture from logs generated by existing RoboCup clients.* Logs describe games as seen from *each player's* point of view.
2. *Store the captured data in a spatial knowledge representation format*, capturing "scenes" — snapshots of an agents behaviour at discrete times.
3. *Apply a distance calculation algorithm to compare real-time situations with previously-stored ones.* The new RoboCup agent responds to each situation by using the same actions that other RoboCup clients used in "similar" situations.

The resulting agent should exhibit behaviour similarities with the training agents, which can be measured both qualitatively (observed similarity) and by using quantitative statistical measures.

For now, actions are derived directly from immediate visual inputs — this implies that the proposed learning process cannot account for underlying motivations not directly tied to the state of the environment, decisions based on prior memory (though future work will seek to address this), or agents that rely heavily on inter-player communication. Ideal candidates for observation are agents that are mostly stateless and have simple goals tied to the positions of objects.

Agent Behaviour Modeling

A Scene Representation For RoboCup

A *scene* represents the point of view of a given player and the associated actions taken by the agent at a given time. Conveniently, RoboCup `see` messages describe this visual information. Agents send commands back to the server in response.

Each scene s is an aggregation of the objects (represented by their type, position, velocity and direction) described in the `see` message and the command sent by the agent during each time period. An entire game is thus represented as a collection of scenes $S = \{S_0, S_1, S_2, \cdots, S_n\}$. (Typically $n = 6000$.)

Graphically, a scene depicts the objects surrounding a RoboCup player, as in Figure 1.

Scenes can be as simple as a logical way of interpreting the existing raw captured log data, or they can actually be used as a higher-level knowledge representation. Scenes provide a convenient atomic unit through which data potentially be further manipulated.

Scene Discretization

A level of generalization can be introduced by partitioning the space surrounding the player into a set of regions. The field of vision may be divided into n segments, each pie-shaped segment covering $(field\_of\_vision/n)$ degrees of view. An object can now be described as being located in any one of these n "slices", effectively discretizing the angular position data — objects can now be described in relative spatial terms (*far left*, *left*, *center*, *right*, *far right*). Distances may also be discretized. The dashed lines of Figure 1 represents a possible discretization into $(5, 3)$ regions.

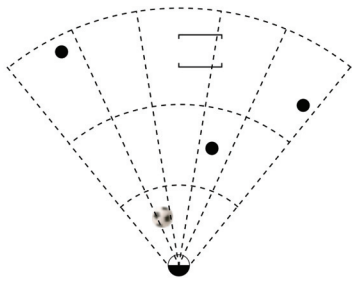

Figure 1: A RoboCup scene (dashed lines represent region-discretization).

Applying such a discretization of the scene representations can provide the following advantages:

- Matching of "similar" scenes by matching the regions that the objects belong to;
- Redundancy of "similar" scenes reduces the number of total scenes needed in the training set (and consequently, more diverse scenes can be stored resulting in greater scene coverage);
- Discretized distance is a distinction already made by the RoboCup soccer server, and scene-matching algorithms must take this into account anyway (e.g. a ball cannot be kicked unless it is considered adjacent to the player);
- Logical predicates can be extracted from examining objects in regions; this could later be used in a spatial reasoning algorithm;
- Configurable trade-off between complexity and accuracy by changing the number of regions.

Scenes may then be represented as tables, where the regions become rows and columns; each cell would contain a list of objects contained within it.

Limitations Potential problem areas to consider include:

- introduction of additional bias, through the decision on the number and boundaries of the regions in a scene;
- edging and boundary issues caused by the artificial segmentation of space, e.g. two close objects may not be classified into the same region, artificially separating them;
- conversely, overgeneralization – declaring two or more scenes to be "similar" when they might in fact describe dissimilar logical behaviours.

Boundary values should be defined at points that have semantic meaning, for example server-defined distances (`ballkickable`, etc.); these may also be determined experimentally (Marlow 2004).

Scene Recognition

Given a new situation N (expressed as a scene) and a set of observed scenes $S = \{S_0, S_1, \cdots, S_n\}$, if any scene $S_i \approx N$, then the recorded action from that scene can be reused.

Scene Recognition Algorithm

We use a k-nearest-neighbor search to find the k best scene matches and corresponding actions; these are then evaluated using a separate action selection algorithm to decide which action will ultimately be chosen. The scene recognition process follows the general algorithm shown below:

```
load the stored scenes from disk
while(game on):
     get new incoming ''see'' data from server
     convert data to scene object N
     for each stored scene x:
          d = DistanceCalculation(N, x)
          S = keep k best results
     end loop
     a = ActionSelection(S)
     send action a to server
end loop
```

The algorithm relies on two important functions, namely `DistanceCalculation` and `ActionSelection`. They are described in the following sections.

Distance Calculation

The `DistanceCalculation(N, x)` function determines a value of "distance" between two scenes. Since scenes are collections of multiple objects in space, the objects are paired (Section), and their distances are summed.

Continuous distance calculations Objects on the field are all described with a distance r and a direction θ relative to the observing agent. This can be represented by a polar coordinate $P(r, \theta)$, and distances between similar objects in two distinct scenes can readily be calculated using the cosine law.

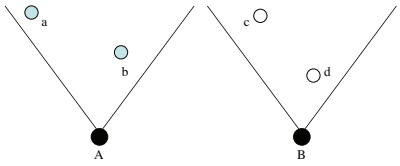

Figure 2: Two closely-matching scenes.

Discretized distance calculations In the discrete-region form, object positions are no longer represented by their position $P(r, \theta)$ but by the (x, y) grid coordinates of the region in which the object lies (also equivalent to the column and row positions of the cell when represented as a table), and a Cartesian coordinate distance may be calculated.

Object Correspondence by Type There are actually numerous different *types* of objects on the soccer field (ball, goals, lines, flags, players). There are also subtypes; e.g. different types of players (teammates and opponents). A distance between two scenes is thus the sum of the individual distances between different object types. When there are more than one object of the same type in a scene, we also need to "match" the objects correctly. Consider Figure 2, which shows two scenes A and B each with two objects. Assuming the objects in each scene are of the same type, a matches with c, and b matches with d.

Object Matching The main complexity of the distance calculation algorithm actually lies in mapping pairs of objects between scenes when there are more than one object of the same type in a scene, as in Figure 2. The problem is that of minimum-weight matching in a bipartite graph, where the cost function to be minimized is the distance between each pair of objects. Objects "missing" from one scene must also be taken into consideration. One approach is to apply a "penalty" distance for objects in one set that have no corresponding object in the other. This penalty should be inversely proportional to the distance; intuitively, this is a measure of the importance of a missing object.

A solution to this problem is Edmonds' blossom-shrinking algorithm (Cook & Rohe 1999) which runs in polynomial time.

Another possibility is to sort the objects by their distance to a common reference point (e.g. the player itself); objects are then paired off and the distances are summed. This is a simple, but non-optimal heuristic — its flaw is its inability to distinguish con-cyclic points which are not actually close to each other. On the other hand, its preference to matching close objects first can help avoid fixating on matching remote objects with minimal influence on the player's behavior, while leaving closer objects unmatched.

We have implemented both the algorithms above, and a comparison of their relative effectiveness is discussed in the Results section.

Object Weighting Finally, different types of objects may be weighted according to their perceived importance in a scene. For example, to ignore a set of objects (i.e. lines on the field may be irrelevant to a given player), weights can be set to 0.

The distance calculation for n objects present in both scenes may thus be expressed as:

$$\begin{aligned} d(S_1, S_2) &= \sum_i \left[w_i * \left(\sqrt{\sum_j (x_{ij} - y_{ij})^2} \right) \right] \quad (1) \\ S_1 &= \{x_1, x_2, \cdots, x_n\}, \\ S_2 &= \{y_1, y_2, \cdots, y_n\}, \\ i &\in \{objecttypes\} \end{aligned}$$

Weights may be derived experimentally or from knowledge of the domain. Weights could also be applied to the distance of the objects to the player. We are currently experimenting with automatic weight calibration using a genetic algorithm approach.

Action Selection

The result of the k-nearest-neighbor search is a set of k scenes, each a potential match. Each has an associated action, (e.g. `dash, kick, turn`) and parameters (power, direction, etc). When $k > 1$, the most common action (majority vote) is used here.

Parameters must also be chosen for selected actions (e.g. which direction to kick in and at what strength). The simplest approach would be to just reuse the values originally stored in the matching scene. Averages could also be taken: if there are n different instances of a `kick`, the average kick power and direction could be taken over the n results.

A logical approach could be for the agent to attempt to interpret the stored scene — i.e. was the {`kick, turn, dash`} aimed at a particular object? This could be determined by examining objects located in the scene within the direction and range of the action. This may not always be straightforward to detect, since the original agent may have taken other aspects into account (such as intercepting a moving object). At least a few degrees of error should also be allowed to account for noise in the RoboCup data.

Implementation

We developed an implementation of the scene format and a RoboCup client based on the matching algorithms described previously. Krislet (Langner 1999) was used as a starting point. Krislet is coded in Java, is easy to extend, and all its client-server communications and message parsing functions are already implemented.

To capture logs from existing games, the LogServer proxy utility (Marlow 2004) is inserted into the communications path between existing RoboCup agents and the server. We have verified experimentally that the use of LogServer does not introduce a "probe effect" in the results.

The captured log files can then be used directly for "continuous" scene recognition, or be fed to a utility which converts the observations and actions at each given time period into discrete scenes.

We implemented both continuous-form and discretized-form variations of the distance calculation algorithm. Each supports weighting of different object classes. Additionally, a "random" distance calculation was created which simply picks any scene at random, and is used to establish a statistical baseline for performance evaluation.

Experimental Results

For testing purposes, we developed a validator class which, instead of connecting to a RoboCup server, runs the recognition algorithms against input from a second scene file. This provides a method for unit testing (testing a scene file against itself (i.e. the training data)) and cross validation (i.e. use of a separate test data set). Intuitively, the use of the weights for objects and the choice of the heuristic-based distance calculation algorithm could lower unit-testing success rates while improving prediction accuracy. However, low unit-testing rates combined with low prediction accuracy might be a sign of an issue with the correctness of the algorithms.

Statistics kept include the algorithm's average execution time and the number of "correct" generated actions (where chosen actions match with the stored actions).

For our experiments, we first selected three different RoboCup agents of varying complexity:

- Krislet (Langner 1999), which uses simple decision-tree logic to express its behaviour, namely that the agent will always search for the ball, attempt to run towards it, and then attempt to kick it toward the goal.
- NewKrislet (Huang & Swenton 2003), which uses a state machine model to express goals; we consider a specific team implementation called the AntiSimpleton, which uses a simple attacker/defender strategy. The Attacker waits near the midfield line until the defenders pass the ball toward it, then attempts to score a goal (much like Krislet) before returning to its home position to await more passes from Defenders.
- CMUnited (Stone, Veloso, & Riley 1999), a RoboCup Champion team which uses a number of strategies including inter-agent communications, formation strategies, and a layered learning architecture that provides the player with skills such as passing and dribbling. This team is

chosen not because of any realistic hope of properly emulating the team but to determine how far we are from such a goal.

Teams consisted of five players each. Log files were collected for one player per team. Log of several full games were translated into stored-scene files, with a region discretization applied using a fixed size of $(3, 5)$ rows and columns.

Results

We measure the effects of varying the distance calculation algorithm, use of continuous versus discrete distance information, varying object weights ($w \in \{0, 1\}$ only at this point), scene selection algorithm and k values ($k \in \{1, 5, 15\}$ only).

The figures show both unit-test and predictive accuracy success rates. The unit-tests help us evaluate the intrinsic correctness of our algorithms, and to what extent our simplifying assumptions are viable.

Object Weight Tests We tested the predictive power of each class of objects on their own and in combination with others (using the 1-nearest neighbor discrete distance calculation algorithm). Figure 3 shows the results from the unit tests and predictive accuracy tests for each of the three scene sets.

Success rates with object classes vary wildly (some no better than random selection) suggesting a strong correlation to the original agent's decision-making. As more objects are considered, data overfitting begins to occur.

The Krislet test shows a significant spike in predictive accuracy when considering just the ball, and a smaller spike when considering the ball and goal. This is consistent with Krislet's known behaviour. This relationship is clearly seen from the statistical data. NKAS shows similar trends.

With CMUnited, no one object grouping or set of objects seems most important. The highest success rate seems to come from considering the positions of players or flags — this may be consistent with the team's known reliance on formations and passing.

Some objects, such as the ball and goal, have obvious importance to all three of the observed teams, both because they are distinct (there can only be one ball, and one goal per team) and because of their known importance in soccer.

Distance Calculation Algorithm Tests Figure 4 shows the results from each calculation algorithm for each of the three agents observed. Our heuristic-based distance calculation algorithm (matching objects in sorted order of their distance to the player), in both continuous and discrete versions, is pitted against Edmonds' "Blossom" bipartite graph matching algorithm. The "RandomDistance" calculation is used to establish a floor by which to compare the other algorithms.

Since the main difference between the distance calculation algorithms is in their object matching heuristic, in this evaluation phase it is important to give equal weight to all the objects, especially to flags, lines and players, since they are those types of more than one object. This will hurt the predictive accuracy rates: we discussed in the previous subsection the much higher importance of the ball and the goal to most players. So here, in the possible absence of good predictive accuracy rates, we are mostly interested in comparing the unit-testing results.

Perhaps surprisingly, the bipartite matching algorithm and the heuristic alternative described earlier in the paper display fairly similar success rates. It is then worth considering that with the better speed of the heuristic, more scenes could be examined within a cycle and the performance could therefore be better. This could be even more the case with the discrete version. However we think that the bulk of further improvement in performance will have to be found elsewhere, in particular by considering state and context information.

Finally, we consider the k value and the selection algorithms used to choose actions. Predictably, the accuracy on the training data is reduced as k increases, since this has the effect of matching scenes that have less and less similarity.

Combined Results The best overall combinations of parameters are listed for each team in table 1.

These combinations allow the trained agent to imitate the Krislet client and many aspects of the NewKrislet agent with reasonable success. The Krislet imitation successfully chases after the ball and kicks it to the goal. This emphasizes the importance of proper object weightings. During one demonstration, the Krislet client was placed in competition with its imitation; from a spectator's perspective, the two agents were almost indistinguishable (Krislet was slightly faster).

In the case of the NewKrislet agent (and the particular implementation of the AntiSimpleton), much of the client's behaviour was observed correctly. Where the algorithms fail is in determining when certain behaviours (e.g. run home and wait) should be taken, which was a function not only of the object positions but also of the agent's internal state.

Using the CMUnited data files, there was little resemblance to the overall behaviour with any algorithm, though the agent would occasionally exhibit some simple behaviours such as running toward the ball or occasionally kicking it, seemingly at random. This was expected, since CMUnited contains logic which considers far more than simply the positions of objects on the field.

Discretized scene matching seems to lower the predictive power and tended to cause more instances of agents "getting stuck" – possibly corresponding to cell boundaries, such as a player trying to kick the ball when it is not quite close enough to successfully do so.

Limitations and Future Work

The current scene recognition framework is subject to the following limitations:

- Does not consider some visual information such as object velocity and direction;
- Does not take into account the importance of objects based on their distance to the player;
- Only uses boolean weight allocation: it's currently an

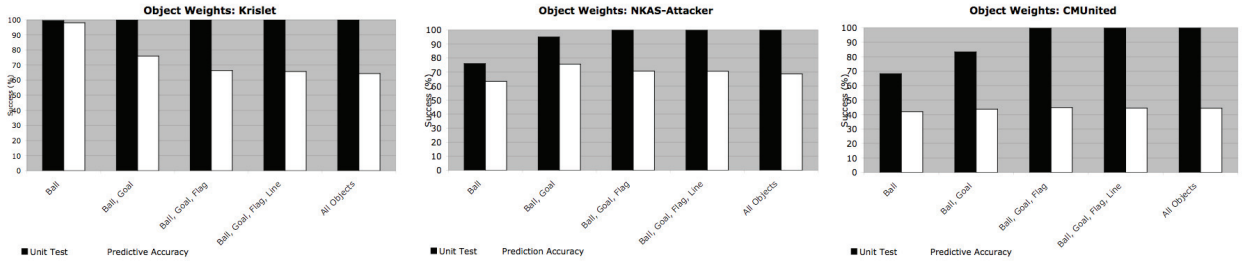

Figure 3: Varying object weights on Krislet, NKAS-Attacker and CMUnited.

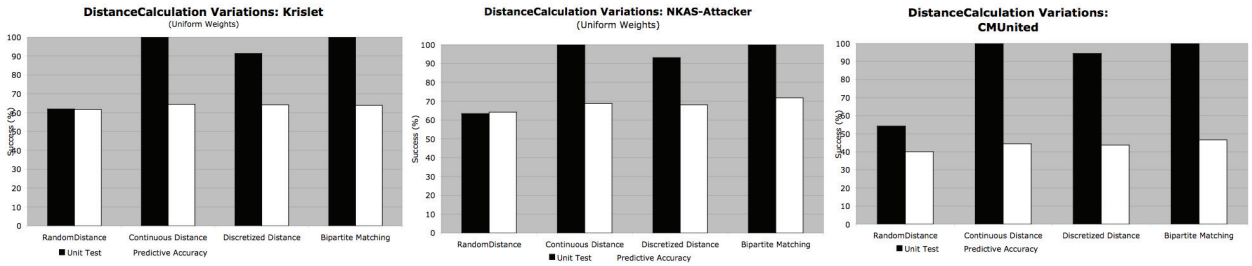

Figure 4: Varying Distance Calculation Method on Krislet, NKAS and CMUnited.

"all-or-nothing" approach when considering the importance of various objects;

- Does not consider non-visual information such as body state, game state, or memory of previous conditions.
- Human optimization required to select appropriate object parameters (weights, etc.)

Future work will address a number of the above limitations, including the following:

- Detection of "stateful behaviour" by examining the observed data and looking for scenes with the same inputs yet different outputs, i.e. $\exists x_1, x_2 | S(x_1) = S(x_2)$ but $a(x_1) \neq a(x_2)$. Large numbers of such scenes would suggest different states influencing the outputs.
- State-based behaviour could be modeled through the application of Hidden Markov Models or a similar state-machine learning approach.
- Automating some or all of the parameter adjustment process, reducing the current dependence on human observation and evaluation of the imitative agent.
- Support for a memory of previous states or objects previously seen but now out of sight, or projections of future states given object position and velocities.
- Higher-level scene representation using spatial logic primitives: this should allow a more compact and expressive representation, easier verification and validation, and a building-block for capturing tactical behavior.
- Looking beyond individual scenes, toward finding patterns or trends over sequences of scenes —the agent should also be able to trigger high-level actions that occur over a sequence of time.

Conclusions

The scene recognition framework and algorithms described here represent some initial steps toward an automated process for observation and imitation of other agents.

Our results suggest that it is possible to learn the behaviour of a RoboCup client if its behaviour can be captured in a simple logical construct such as a decision tree.

The current stateless, single-layered imitative agent is able to almost perfectly copy the behaviour of a stateless, single-layered agent (Krislet) and is fairly representative of at least some of the behaviours of more complex clients. These results are generally encouraging and suggest that with further development (including a layer of base logic and basic skills, the ability to internally represent different agent states, etc.) it may be possible to further increase the accuracy of the imitative agent.

References

Cook, W., and Rohe, A. 1999. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing* 11:138–148.

Dorer, K. 1999. Extended behavior networks for the magmafreiburg soccer team. In Coradeschi, S.; Balch, T.; Kraetzschmar, G.; and Stone, P., eds., *RoboCup-99 Team Descriptions for the Simulation League*. Stockholm, Sweden: Linkoping University Press. 79–83.

Table 1: Best Statistical Results for Agent Imitation

	Distance Calculation	**Object Weights**	k **Nearest Neighbors**
Krislet	Discrete (93%)	ball and goal	$k = 1$
NewKrislet	CellBallGoal (70%)	ball and goal	$k = 1$ or $k = 5$
CMUnited	Continuous or Discrete (44%)	ball, or flags, lines, players	$k = 5$

Dousson, C. 1996. Alarm driven supervision for telecommunication networks : II- On-line chronicle recognition. *Annals of Telecommunications* 51(9-10):501–508. CNET, France Telecom.

Huang, T., and Swenton, F. 2003. Teaching undergraduate software design in a liberal arts environment using robocup. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, 114–118. New York, NY, USA: ACM Press.

Langner, K. 1999. The Krislet Java Client. last accessed April 2006, `http://www.ida.liu.se/~frehe/RoboCup/Libs/libsv5xx.html#Krislet`.

Marlow, P. 2004. A process and tool-set for the development of an interface agent for use in the robocup environment. Master's thesis, Carleton University.

Matsui, T.; Inuzuka, N.; and Seki, H. 2000. A proposal for inductive learning agent using first-order logic. In Cussens, J., and Frisch, A., eds., *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 180–193.

Murakami, Y.; Ishida, T.; Kawasoe, T.; and Hishiyama, R. 2003. Scenario description for multi-agent simulation. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 369–376. ACM Press.

RoboCup. 2006. Robocup official site. `http://www.robocup.org`.

Steffens, T. 2002. Feature-based declarative opponent-modelling in multi-agent systems. Master's thesis, Institute of Cognitive Science Osnabruck.

Stolzenburg, F.; Obst, O.; Murray, J.; and Bremer, B. 2000. Spatial agents implemented in a logical expressible language. In Veloso, M.; Pagello, E.; and Kitano, H., eds., *RoboCup-99: Robot Soccer WorldCup III*, volume 1856. Springer.

Stone, P.; Veloso, M.; and Riley, P. 1999. The CMUnited-98 champion simulator team. In Asada, M., and Kitano, H., eds., *RoboCup-98: Robot Soccer World Cup II*. Berlin: Springer Verlag.

Stone, P. 1997. Layered learning in multiagent systems. In *AAAI/IAAI*, 819.

Xie, M.; Muhammad, A.; and Zhang, Z.-E. 2004. *Final Project Report: Zed Robocup Soccer Client*. SYSC 5103 Software Agents course project report, Carleton University.