Soccer-Swarm: A Visualization Framework for the Development of Robot Soccer Players

Lorin Hochstein, Sorin Lerner, James J. Clark, and Jeremy Cooperstock

Centre for Intelligent Machines
Department of Computer Engineering
McGill University
Montreal, Quebec, Canada H3A2A7

ABSTRACT

This paper proposes a framework for the rapid development of high-level, domain-independent AI strategies targeted at the RoboCup competition. This framework, developed within the Swarm simulation system, provides a layer of abstraction that allows strategies to be easily ported from one domain to another. Additionally, the framework provides a powerful and extendable visualization tool that should significantly decrease development and debugging time of high-level strategies.

KEYWORDS

Robocup, Artificial Intelligence, Visualization, Swarm

INTRODUCTION

The RoboCup competition [1] presents Artificial Intelligence researchers with the challenge of developing soccer-playing agents who must co-operate to achieve a goal while immersed in a noisy environment. The development of such agents can be a difficult and time-consuming task, since high-level strategies depend on the correct operation of basic tasks such as passing and dribbling, which in turn depend on the agent having an accurate model of the world. These agents can also be notoriously difficult to debug, since it is difficult to determine what exactly is going wrong when agents do not behave as expected.

This paper presents an agent development and visualization framework which aims to simplify the design and debugging of soccer agents. The framework provides graphical visualization tools, which can simplify the task of developing and debugging strategies. These tools give the developer a better picture of how the agent is behaving, as well as motivate the development of novel, graphical-based strategies. Additionally, the framework provides a layer of abstraction that separates the task of high-level strategy design from the domain-dependent aspects. This layer of abstraction should allow high-level strategies to be ported easily from one domain to another, so that a strategy that works for agents interacting in a simulation can be made to work just as well for agents implemented as robots.

BACKGROUND

RoboCup is an initiative that proposes that, like the playing of chess, the playing of Soccer can serve as a standard AI problem. As Kitano et. al. [1] suggest, the design of autonomous agents that play soccer presents many challenges. Among these are handling the dynamic and unpredictable nature of the environment and dealing with incomplete information about the world. The design of effective cooperative or collaborative behaviors in multi-agent systems is one of the most important challenges that researchers face [2].

To encourage the design of soccer agents, a yearly tournament is held, where teams of autonomous soccer playing agents compete against each other. There are several leagues in the competition, mainly divided onto two categories: the real robot league, where agents are real robots, and the simulation league, where agents are software programs playing in a simulated environment.

Although our framework targets both of these leagues, it has to date only been applied to the simulation league. The architecture of the simulation league is very simple. A server, called the soccer server, simulates the motion of the ball and of the players on the field. Each player communicates with the server through UDP sockets: the server sends visual and auditory percepts to the agent, while the agent sends actions back to the server, indicating what it wishes to do.

PURPOSE OF THE FRAMEWORK

Among the challenges presented by the game of soccer, we have concentrated on two: evaluating the behavior of individual agents within the structure of an emergent team strategy, and designing agents that can function in both the simulated league, and the real robot league.

Evaluating agent behaviors in a team structure

Soccer is a dynamic multi-agent problem, and as such is difficult to observe. Indeed, watching twenty-two agents play in real-time provides at best a global understanding of the game. In addition, because this global view does not indicate what each player is thinking, it is very difficult to evaluate why a certain co-operative strategy succeeds or fails.

One common solution to this problem is to remove the real-time component when observing the game. Thus, one can make a log of each agent's thinking process, and then examine these log files once the game is over. Apart from losing the real-time component of soccer, this approach has a far greater limitation: it tries to evaluate team behavior by looking at individual players, without considering what teammates or opponents are doing. In other words, by focusing on each player, the global picture of the field has faded away.

The idea then is to try to get the best of both worlds: see what each individual player is thinking, but at the same time get a global view of the soccer field. This can be achieved by overlapping the graphical display of agent-specific information with the display of the game while it is being played. For example, the player having the ball might display where it intends to dribble the ball, and to which player it will then pass.

Such graphical methods will help evaluate how individual behaviors fit into the global team strategy, and thus can be used to explain how and why certain team strategies work. In fact, exactly as graphs are used today to backup scientific claims, we hope that our "real-time soccer graphs" will backup claims about co-operative behaviors.

Designing multi-platform agents

The other challenge that we chose to tackle is the integration of robotics with high-level AI. Ideally, a high-level AI strategy should be developed once and then deployed in different environments without massive restructuring. In order to achieve this, the high level AI strategy, which is independent of the environment in which it is deployed, will be isolated from the low-level interaction with the environment. This will allow high level AI strategies to be transplanted seamlessly from a software agent to a real robot (or vice-versa) or from one real robot to another.

THE DEVELOPMENT FRAMEWORK

The Swarm Simulation System

Our development framework is based on the Swarm Simulation System [3]. Swarm is a collection of software libraries for simulating multi-agent systems. It includes a discrete event simulator that provides a set of graphical widgets for visualization. Swarm was developed by the Santa Fe Institute to provide a standardized set of tools for complex-systems researchers, with the aim of sparing developers from the effort of developing their own discrete event simulators, as well as providing a standard framework which would allow a fair evaluation of results.

Swarm forces the programmer to organize agents into hierarchical collections called model swarms, and provides graphical displays to the user through objects called observer swarms. Every Swarm program requires a model swarm; for graphical feedback an observer swarm is also required.

*Model swarms***.** A model swarm consists of at least two items, a collection of agents, and a schedule. Model swarms are recursive, so the agents can themselves be model swarms. The schedule defines the order in which agents act.

*Observer swarms***.** An observer swarm provides graphical feedback to the user by probing the model swarm and displaying relevant information in a format that is meaningful for the user. Swarm provides a simple but powerful collection of GUI tools that allow the programmer to develop images such as graphs, histograms, and polygon-based images with a much simpler interface than the underlying Tcl/Tk widgets.

Soccer-Swarm Model Swarm

The model swarm contains all the objects that are modeled in the simulation, namely the players and the soccer field. The interaction between these objects is governed by a schedule, which in our case is a simple round robin schedule: at each simulation step, control is given in turn to every player, and finally to the soccer field. The model swarm for our soccer agent visualization is depicted in figure 1.

When the soccer field gets its turn to act, it will do whatever is necessary to advance the simulation of the environment. In Figure 1, the soccer field is drawn in dashed lines, which indicates that it is not implemented by the framework itself, but rather by the developers who will apply the framework to a particular environment. Thus, the soccer field can be implemented in any way that is appropriate for the domain at hand. One possibility is to directly program the soccer field as a simulation. Another possibility is to have the soccer field relay information from some other source, for example the soccer server in the simulation league, or even the real world. The case where the soccer field gets information

from the real world is in fact very interesting, since it allows our framework to control a robot in real-time.

When a player gets its turn to act, it will have the opportunity to interact with the soccer field through a generic interface that can be adapted for each specific domain. The player can retrieve percepts from the soccer field, and send actions back. Although similar to the interaction between the soccer server and the client in the simulation league, the interface between objects in the Model Swarm is much more general, since the information passed need not be specific to the simulation league. For example, the percepts can range from sonar data to video data, while the actions can include commands to robotic arms.

The design of the player itself is split in two components, both of which are drawn in figure 1 with dashed lines, which again means that these components will be implemented when the framework is applied. The two components in the player are the Field Abstraction Layer (FAL), which is domain dependent, and the Deliberation Layer (DL), which is domain independent.

Field Abstraction Layer (FAL). The FAL is an abstraction layer whose purpose is to isolate the high-level AI from the low-level details of environment interaction. To achieve this abstraction, the FAL has two tasks. First, it uses the information gathered from the soccer field in order to provide domain-independent world-modeling services to the DL. The information given to the DL consists of positions, velocities and certainty values for the objects on the field. Second, the FAL implements a set of domain-independent low-level skills that the DL can choose from. These low-level skills usually require more than one action in order to be completed. For example, dribbling to a position on the field might require a sequence of alternating small "kicks" and small "run" commands.

Deliberation Layer (DL). The DL, which represents the high-level AI strategy of the player, uses the FAL's world modeling services in order to decide which one of the low-level skills it wishes to activate. In addition to choosing a new skill, the DL also has the option of letting the currently chosen skill run to completion.

*Interaction between the DL and the FAL***.** Each time the player gets a chance to act, the FAL takes over and arbitrates between the soccer field and the DL. First, the FAL updates the player's worldview with any new percepts from the soccer field. Then, if the FAL considers that enough new information has arrived for deliberation to be justified, it gives control to the DL, which chooses a new skill, or lets the current one continue. Finally, after the deliberation is done, the FAL sends to the soccer field the best action for the currently selected skill.

The Soccer-Swarm Observer Swarm

In the Swarm simulation system, the Observer Swarm object provides the developer with the graphical tools to visualize the strategies that are being developed. An observer swarm has two basic tasks: collect data from the model swarm and display at it to the screen. Data collection is achieved through a set of objects called Data Collection and Gathering (DCG) objects, and the data is displayed onto a raster window, which is represented as a Soccer Field Raster (SFR) object.

*DCG objects***.** The DCG objects extract data from the soccer agents or from the soccer field, and use this data to draw images on the Soccer Field Raster. The framework provides DCG objects as the building blocks to build customized views. As an example, the framework provides two fully implemented DCG's. One such DCG extracts exact positional information from the Soccer Field object, and draws a graphical representation of the ball and the players on the raster. Another DCG extracts information regarding an agent's perceived location of objects on the field, as well as corresponding certainty values. The less confident a player is on the position of an object, the darker the object appears. If a developer decides to use both these DCG's, then the raster will show a superposition of the perceived object location (from a given agent's perspective) and exact object location. Developers should design their own DCG's to display graphical information, which corresponds to the high-level strategy algorithms they are implementing. Figure nnn shows an example of a DCG that draws a vector force field, where opponent players (on the right side of the field) act as point charges.

*Soccer Field Raster***.** The SFR object is a graphical widget derived from one of the Swarm rasters, which serves as a graphical representation of the field. Multiple DCG's may draw on the SFR, which allows the developer to build a view by mixing and matching DCG's. The SFR also supports event handling in the form of mouse clicks, so the developer can allow the user to change the nature of the graphical feedback. For example, clicking on an agent will cause the SFR to show the state of the field from the perspective of the selected agent.

VALIDATION OF THE FRAMEWORK

The framework's effectiveness was evaluated by implementing a high-level strategy, which would be well suited to a graphical approach. For evaluation purposes the RoboCup Simulation league was used as the target domain.

Domain-specific issues

Before a strategy can be evaluated, the domain-specific components must be implemented. A complete system

requires that a Soccer Field object and a FAL be implemented corresponding to the domain in question.

*Soccer Field object***.** To have the agents interact in the simulation league requires that the program communicate with the RoboCup soccer server. The Soccer Field object handles all communications with the soccer server, serving as the interface between Swarm-modeled players and the server. The Soccer Field object enables the developer to view the exact position of objects on the field by retrieving this information from the soccer server. Furthermore, communication with the soccer server enables Swarm-modeled agents to interact with soccer agents that are implemented entirely outside of the framework provided these external agents can communicate with the soccer server.

*FAL object***.** To evaluate the strategy, only a simplified version of a FAL was implemented.

Planning a Sequence of Passes

The main validation of our framework was done by evaluating the performance of a planning algorithm. In order to simplify the planning problem, we have only considered one possible actions, namely passing. Since all other actions, such as dribbling or running, have been ignored, a plan for our purposes becomes a simple sequence of passes. Adding other actions into the planing problem is left as future work.

The problem we are looking at is to find the best plan, or pass sequence, to execute. Traditional planning doesn't seem particularly suited for this problem, since traditional planning solves a satisfiability problem, whereas we are looking at an optimization problem. Thus, we will do planning by optimization.

To start, we define a goodness function that assigns a numerical value to a given sequence of passes. The higher the goodness value, the better the pass sequence is. The goodness function should obviously depend on the probability that the pass sequence succeeds, which in turn depends on the success probability, p_l, of each individual pass in the sequence. In addition, the goodness function should also depend on some factors that indicate how advantageous the play will be if it actually succeeds. Thus, the goodness function can be expressed as

$$g(s) = p(s)a(s)$$

where s is a given pass sequence, a(s) indicates how advantageous the play would be if successful, and p(s) is given by

$$p(s) = \prod p_i$$

In our implementation, the success probabilities p_i are computed using an algorithm, but they can also be learnt by a neural network, in a manner similar to that employed by [6]. The function a(s) was chosen to return how much closer the ball is to the opponent's goal after the pass sequence is completed. Thus

$$a(s) = x_{last} - x_{first}$$

Once the goodness function is defined, we simply search for all pass sequences of length n or less, and find the one with the best goodness value. Although this method runs in exponential time with respect to n, it is tractable for several reasons. First, we limit the depth of the search to a reasonably small number, say n = 3. Second, passes are considered only if their probability is above a certain threshold. This prunes out the search considerably, as it will not consider very unlikely or even impossible passes, such as a pass from one end of the field to the other. (In our case, the threshold was 0, but we made the algorithm assign probabilities of 0 to very unlikely passes). Finally, we do not consider players that are already part of the pass sequence, which in essence means that we don't allow loops in the sequence. This constraint will be loosened once other actions, such as running and dribbling, are considered. Indeed, a player might pass to another, who will then pass back to the first, but at a different location.

Finally, once a player has decided on the best pass sequence, it will execute the first pass in the sequence. The next player will re-evaluate the best pass sequence, and again choose the first pass in this sequence. Thus, there is no explicit communication between the agents. However, our hypothesis is that if the world doesn't change much, the next player in the sequence will choose a pass sequence that is a continuation of the first. On the other hand, if the world does change considerably during the execution of the first pass, the next player will re-plan anyway.

To test our hypothesis, we used real-time soccer graphs to visualize the intentions of the players. Thus, we implemented a DCG which displays the pass sequence that a given player considers to be the best. The pass sequence is shown on the Soccer Field Raster by drawing lines between the locations of the players in the pass sequence. Figure 3 shows such an example, where the probed player is the bottom one. Although the current DCG changes the probed player only in response to mouse clicks, a more general approach would be for the DCG to decide itself which player to look at. For instance, the DCG might choose to probe the player that is closest to the ball.

By having the capability of switching between players, we were able to evaluate how our pass sequence selection created stable plans. For example, in figure 3 the pass sequence shown is that of the bottommost player. When the second player in the pass sequence is selected by the DCG, as shown in figure 4, we see that the new pass sequence is indeed a continuation of the first. Thus, we are able to see an emerging team plan from the behavior of individual agents.

RELATED WORK

Our split of the player into two parts has some similarities with the Reactive Deliberation architecture proposed by Sahota [4,5]. Indeed, the Reactive Deliberation architecture also splits the player in two: the executor, which implements a number of parameterized skills called action schemas, and the deliberator, which chooses from one of these action schemas. However, Sahota's main motivation for the split was to combine two approaches that run in different time scales: deliberation, which is computationally expensive, and reactive behavior, which requires constant interaction with the environment. Thus, to solve the gap in time scale, the executor continually interacts with the environment in order to provide highly reactive behavior, whereas the deliberator, running in parallel, can indulge in heavier computations. Our motivation, on the other hand, was to separate the domain dependent part of the player from the domain independent. Because of this, we have not yet concentrated on bridging the time gap that Sahota looks at, so that the current framework does not support running the DL and the FAL in parallel. However, this area is a possibility for future work, and might lead to a framework even more similar to the Reactive Deliberation architecture.

CONCLUSION

We have developed a visualization framework for the development of soccer robot behaviours. The advantages of our Soccer-Swarm framework are the following:

- A graphical interface to player design which provides the developer with a view of the behavior of individual agents as well as a global view of the system, reducing development time and motivating novel strategies.
- The ability to port high-level strategy from one domain to another.

A planning strategy has been developed to evaluate the practically of the framework. This strategy, which relies heavily on graphical feedback to the user to demonstrate its effectiveness, demonstrates the usefulness of being able to probe the minds of the agents.

FUTURE WORK

Improvements on the framework

There are several improvements on the framework that can be the source of future work. One of these has already been mentioned, namely allowing the DL and the FAL to run in parallel. Another improvement would be to add support for communication between agents, something currently not in the framework. The simulation league of RoboCup allows agents to communicate, and real robots also have methods of communicating. Thus, our framework should support some sort of abstract communication paradigm, which would be applicable to both the simulation league and to real robots.

Improvement on the planning of pass sequences

Adding other actions to the planning algorithm would allow for more elaborate plans to be created. We believe this avenue might lead to some very interesting results, although there are difficulties to overcome. The first problem is how to formulate actions such as dribbling and running. For passing, there are only a finite number of teammates that one can pass to, but for running or dribbling, there are infinitely many destinations. Discretizing the field into sections makes the number of destinations finite, but still too big for the search to be tractable. One solution to this problem might be to specify the actions of dribbling or running qualitatively instead of quantitatively. For instance, one might replace "run to a certain position on the field" with "run to a position so that you can receive a pass from this player".

ACKNOWLEDGEMENTS

We would like to thank Pascal Poupart for his helpful insight on methods for doing planning with optimization. This research was supported by a research grant from the IRIS National Centre of Excellence.

REFERENCES

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, “RoboCup: The Robot World Cup Initiative”. In *Proceedings of the First International Conference on Autonomous Agents*, 1997, 340-357.

[2] H. Matsubara, I. Noda, and K. Hiraki, “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*, pages 63-67, Menlo Park,CA, March 1996. AAAI Press. AAAI Technical Report SS-96-01.

[3] N. Minar, R. Burkhart, C. Langton, and M. Askenazi, “The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations”, Technical Report, Santa Fe Institute, Santa Fe, New Mexico, 1996

[4] M.K. Sahota, “Reactive Deliberation: An Architecture for Real-time Intelligent Control in Dynamic Environments”. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1303-1308. Seattle, 1996.

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

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

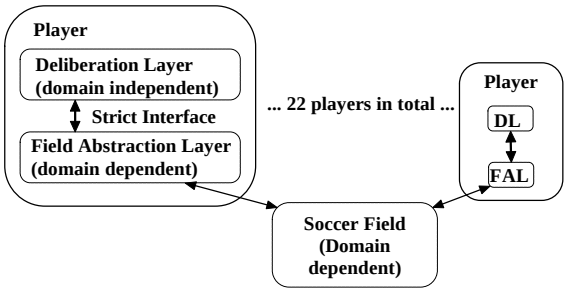

Figure 1. The Model Swarm Architecture

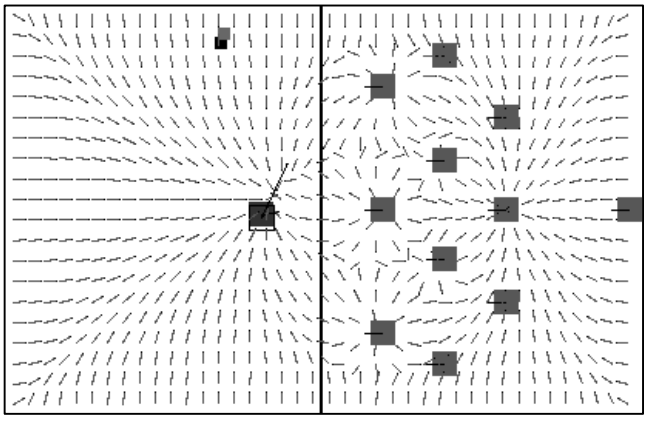

Figure 2. A Vector-field Data Collection and Gathering Object (DCG). Shown are the gradient vectors corresponding to an electrostatic force field arising from considering each player as having an electric charge.

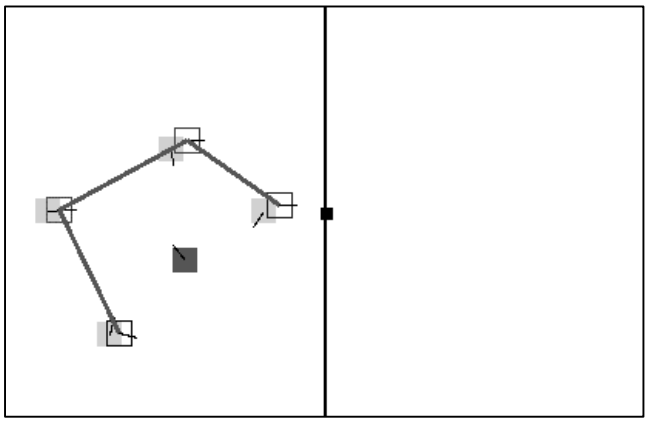

Figure 3. Pass planning. Shown is a DCG which displays the pass sequence that the bottom player considers to be the best.

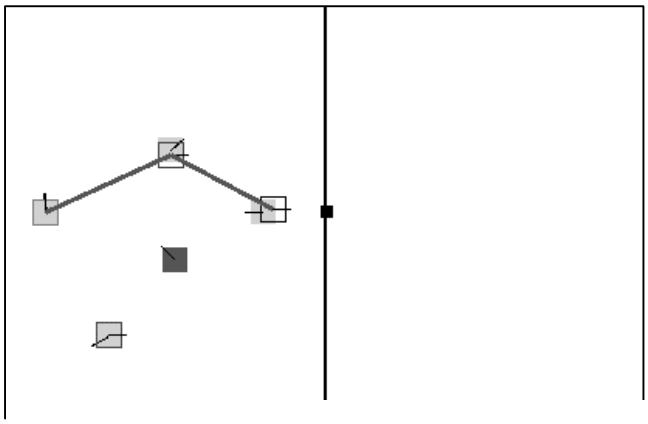

Figure 4. Pass planning from the perspective of the second player in the pass sequence.