JavaSoccer

Tucker Balch

Mobile Robot Laboratory
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-208 USA

Abstract. Hardware-only development of complex robot behavior is often slow because robot experiments are time-consuming and robotic hardware is subject to failure. The problem is exacerbated in multi-robot tasks like robot soccer where successful experiments require several robots to operate correctly simultaneously. Robotics research proceeds more quickly through a cycle of experimentation in simulation and implementation on hardware than through hardware-only implementation. To facilitate research in the RoboCup soccer domain we introduce JavaSoccer, a portable system supporting development of multi-robot control systems in simulation and, potentially, on soccer robots. In simulation, JavaSoccer emulates the dynamics and dimensions of a regulation RoboCup small size robot league game.

1 Introduction

Soccer is an increasingly active domain for mobile multiagent robotics research. The soccer task is simple and familiar to most people, yet provides opportunities for diversity and cooperation in the individual team members [5, 3, 2]. No matter the domain, multi-robot team design is challenging because success depends on coordination between complex machines. Since robot hardware is subject to frequent failure and experimentation is time consuming, hardware-only robot development is difficult.

To address these issues many researchers turn to a cycle of simulation and robot experiments to develop their systems. Behaviors are prototyped in simulation, then implemented on robots. Even though further refinement of the control system is often required for use on robots, simulation can rapidly accelerate control system development. To facilitate experimentation in RoboCup soccer, we introduce JavaSoccer, a dynamic simulation of RoboCup small size league play.

JavaSoccer is part of the JavaBots software distribution available from the Mobile Robot Laboratory at Georgia Tech. JavaBots is a new system for developing and executing control systems on mobile robots and in simulation. Individual and multi-robot simulations are supported, including multiple robot types simultaneously. The JavaBot system's modular and object-oriented design enables researchers to easily integrate sensors, machine learning and hardware with robot architectures. Additionally, since JavaBots is coded in Java, it is easily ported to many operating environments. To date it has run under Windows 95, Linux, SunOS, Solaris and Irix without modification.

The JavaBot system is being utilized in several ongoing research efforts at Georgia Tech, including RoboCup robot soccer, foraging robot teams and a "robotic pet" development effort. At present, control systems developed in JavaBots can run in simulation and on Nomad 150 robots. Hardware support will soon be provided for ISR Pebbles as well. In the future we hope to expand this capability to small soccer robots, so that behaviors tested in the JavaSoccer simulation will run directly on hardware.

Another important soccer simulator, the SoccerServer developed by Noda is used in simulation competition at RoboCup events [6]. While JavaSoccer is focused on simulation and control of existing, physical robots, SoccerServer is targeted at simulation of more futuristic, human-like players. The SoccerServer is coded in C++ and runs in most Unix/X environments. JavaSoccer also runs in Unix/X environments, but it benefits from the high degree of portability afforded by Java, allowing it to run in Windows 95 and NT machines as well.

The remainder of this chapter will describe JavaSoccer in detail, including the simulation kernel, JavaSoccer rules, robot capabilities and directions for future work.

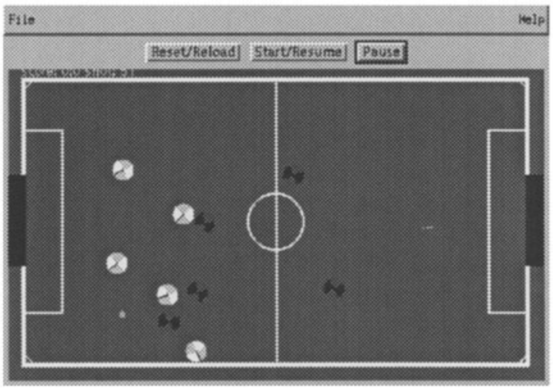

Fig. 1. A JavaSoccer game in progress.

2 The Simulation System

JavaSoccer runs in the JavaBotSim simulator distributed with JavaBots. At runtime, the simulator reads in a description file that describes which objects will be present in the simulation and where they should appear in the environment. Obstacles, small soccer robots, Nomad 150 robots, golf balls and "squiggle" balls have all been implemented to date. A description file appropriate to soccer simulation is included with JavaBots in the JavaSoccer directory (`robocup.dsc`). Simple modifications to the description file enable users to change the robot type and control systems used as well as adjust the color scheme of the robots.

Each object in the simulation includes two important components: a drawing method and a dynamics simulation method. The simulation kernel, or core, runs in a loop, calling the two methods for all the objects in turn at each step. The kernel also keeps track of time, and passes elapsed time to the dynamics methods so they know how far to move their associated simulated objects. The simulation can run in real-time or faster or slower than real-time according the `time` parameter set in the description file.

In addition to the drawing and dynamics methods, each robot has an associated control system object that considers sensor inputs and makes actuator outputs for the robot. A soccer robot control system is coded in Java by extending the `ControlSystemSS` class (`SS` stands for Small Soccer robot). Several example control systems are provided with the software. The intent is for these examples to be modified by researchers for the purpose of testing their own soccer strategies.

Since the API between the control system and a simulated robot is the same as for actual hardware, the same control algorithm can be tested in simulation and in real robot runs. The capability for running soccer robot control systems on small robot hardware is not available yet, but users can develop and run systems for Nomad 150 robots.

3 RoboCup Rules in JavaSoccer

Rules for JavaSoccer play are based on RoboCup's regulations for the small size robot league [4]. Some differences are necessary in JavaSoccer due to the simulation environment. Any differences are noted below:

Field dimensions The field is 152.5cm by 274cm. Each goal is 50cm wide. The defense zone, centered on the goal, is 100cm wide by 22.5cm deep.

Defense zone A defense zone surrounds the goal on each side. It is 22.5cm from the goal line and 100cm wide. Only one defense robot can enter this area. Note: this is not enforced by JavaSoccer, but it will be eventually.

Coordinate system The center of the field is $(0, 0)$ with $+x$ to the right (east) and $+y$ up (north). The team that starts on the left side is called the "west" team and the team on the right is called the "east" team. At the beginning of play, the ball is placed at $(0, 0)$.

Robots Robots are circular and 12cm in diameter. In RoboCup rules 15cm is the maximum size. The simulated robots can move at 0.3 meters/sec and turn at 360 degrees/sec.

Team A team consists of no more than 5 robots.

Ball The simulated ball represents an orange golf ball. It is 40mm in diameter and it can move at a maximum of 0.5 meters/sec when kicked. It decelerates at 0.1 meters/sec/sec. Ball collisions with walls and robots are perfectly elastic.

Field Coloring Coloring of the field doesn't really matter in the simulation, but the image drawn on the screen matches the RoboCup regulations.

Robot marking Robots are marked with a two-color checkerboard pattern. In RoboCup rules they are marked with two colored ping-pong balls.

Length of the game At present, once the game starts, it runs forever. This will be changed to conform with the 10 minute halves of RoboCup.

Wall A wall is placed around all the field, except the goals. Ball collisions with walls are perfectly elastic. Robots may slide along walls if they aren't headed directly into them.

Communication There is presently no facility for robot to robot communication, but this may be added if researchers are interested in the capability.

Goal keepers At present, the goal keeper has no means for grasping the ball. This feature may be added later.

Kick-off/Restart/Stop Kick-off, restart and stop of the game are managed automatically. The west team gets to kick-off fist, with subsequent kick-offs made by the scored-against team.

Robot positions at kick-off The robots are automatically positioned as shown in Figure 2 at kick-off. The numbers next to the players indicate values returned by the `getPlayerNumber()` method. Specific positions are not specified in the RoboCup rules, except that only the kick-off team can be within 15cm of the ball.

Fouls/Charging Currently no fouls are enforced by JavaSoccer. There is however a "shot clock." At the beginning of a point, the clock is set to count down from 60 seconds. If no score occurs in that time, the ball is returned to the center of the field. The ball is also returned to the center if it gets stuck for a period of time. Those types of fouls that can be evaluated objectively by the running program will eventually be added. RoboCup rules include several fouls and charging violations.

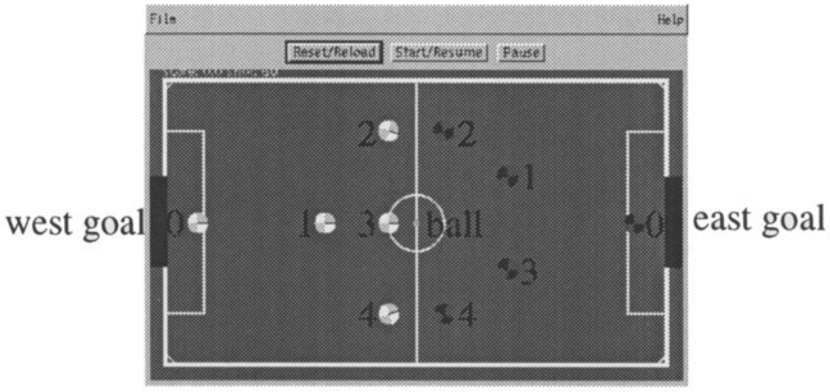

Fig. 2. Starting positions for a JavaSoccer kickoff. The west team is kicking off.

4 Capabilities of a JavaSoccer Robot

The interface between the robot's control system or "brain," and its sensors and actuators is provided by an API defined in the `SocSmall` Java class. The most complete description of the interface is in the javadoc-generated documentation for `SocSmall`, and `SocSmall`'s parent class, `Simple`, but a synopsis is provided here:

- Sensors
  - Detect whether the robot is in a position to kick the ball.
  - Get a vector pointing to the ball.
  - Get a vector pointing to the team's goal.
  - Get a vector pointing to the opponents' goal.
  - Get an array of vectors pointing to all the other players.
  - Get an array of vectors pointing just to the robot's teammates.
  - Get an array of vectors pointing just to the robot's opponents.
  - Get the player's number on the team (0 - 4), 0 is the goalie.
  - Get the player's position on the field.
  - Get the player's heading.
  - Get the time in milliseconds since the game started.
- Actuators
  - Kick the ball at 0.5 meters/sec.
  - Push the ball by driving over it.
  - Set desired heading. The robot can turn at 360 degrees/sec.
  - Set desired speed, up to 0.3 meters/sec.

Each of these capabilities is implemented as a method, or function call, in the `SocSmall` class.

5 Developing New Team Behaviors

New team behaviors are implemented by extending the `ControlSystemSS` class. The easiest way for a developer to accomplish this is by revising of one of the existing teams provided with JavaSoccer in the `JavaSoccer/teams` directory. Some of the example teams were developed using Clay, a behavior-based configuration system [1]. While developers may find Clay helpful in designing their strategies, it is not required.

Two methods must be implemented to complete a control system: `Configure()` and `TakeStep()`. `Configure` is called once at set-up time to provide for initialization and configuration of the control system. `TakeStep()` is called repeatedly by the simulation kernel to allow sensor polling and action selection. Sensors are read by calls to `abstract_robot.get*` methods, actuator commands are sent by calls to `abstract_robot.set*` methods.

Sensor example:

```
abstract_robot.getBall(curr_time)
```

reads the ball sensor and returns a 2-dimensional vector towards it. The `curr_time` parameter is a timestamp that helps prevent redundant sensor accesses (this is more important for use on real robots than in simulation). All other sensor methods are similar in form.

Actuator example:

```
abstract_robot.setSteerHeading(curr_time,0)
```

In this call, the second argument is the direction for the robot to head, in radians. So the robot will steer in the east, or $+x$, direction. Readers are invited to look at the example teams for more detail on sensor and actuator methods.

6 Simulating New Hardware

One of the advantages of the JavaBot system is the ease with which new objects and robot types may be added to the simulation system. It is a fairly simple matter to revise an existing class to emulate new robot hardware. For instance, to create a new, larger simulated robot with the same capabilities as a `SocSmall`, one need only extend the `SocSmall` class and revise the `RADIUS` variable. Other more complex changes are still straightforward, the developer need only ensure that all the methods defined by the `SimulatedObject` interface are implemented.

7 Installing and Running

The entire JavaBots distribution, including JavaSoccer is available as a "zip" file on the world-wide-web. Installation entails downloading the distribution and un-zipping it. Instructions are available at

```
http://www.cc.gatech.edu/~tucker/JavaBots
```

8 Future Directions

Our goal in building JavaBots and JavaSoccer is to accelerate the development of behaviors for physically-embodied robots by supporting identical control system code in simulation and on robot hardware. At present, for small soccer robots, the control systems are only supported in simulation. An important next step will be to implement the API for small mobile robot hardware. This has already been demonstrated in JavaBots for Nomad 150 robots.

Other potential improvements include: closer conformance to RoboCup regulations, simulation of several robot types actually used in RoboCup competition, and new simulation features such as robot position traces and player number display.

9 Acknowledgments

The author is indebted to Ron Arkin, director of the Mobile Robot Laboratory for access to computer and robot hardware used in the development of JavaBots. Itsuki Noda's SoccerServer provided an excellent example soccer simulation system [6]. I also wish to thank Peter Stone, Manuela Veloso, Hiroaki Kitano and Maria Hybinette for their advice in this endeavor.

References

1. T. Balch. Clay: Integrating motor schemas and reinforcement learning. College of Computing Technical Report GIT-CC-97-11, Georgia Institute of Technology, Atlanta, Georgia, March 1997.
2. T. Balch. Social entropy: a new metric for learning multi-robot teams. In *Proc. 10th International FLAIRS Conference (FLAIRS-97)*, May 1997.
3. Tucker Balch. Learning roles: Behavioral diversity in robot teams. In *AAAI-97 Workshop on Multiagent Learning*, Palo Alto, CA, 1997. AAAI.
4. RoboCup Committee. Robocup real robot league regulations. *http://www.csl.sony.co.jp/person/kitano/RoboCup/rule.html*, 1997.
5. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proc. Autonomous Agents 97*. ACM, 1997. Marina Del Rey, California.
6. Itsuki Noda. Soccer server: a simulator for robocup. In *JSAI AI-Symposium 95: Special Session on RoboCup*, 1995.