Soccer Server: a simulator of RoboCup

NODA Itsuki
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, 305 Japan
noda@etl.go.jp

Abstract

Soccer Server is a simulator of RoboCup. Soccer Server provides an environment to confront two teams of players that are controlled by various types of systems with each other. Each system connects to the server as a client, which controls a player on the soccer field via a computer network. Using this server, we can compare performances of multi-agent systems and multi-robot-control systems from various viewpoints.

1 Introduction

Recently, soccer is used as an example to test and demonstrate performances of various kind of AI systems [Uchibe *et al.*, 1995; Littman, 1994; Sahota and Mackworth, 1994; Sahota, 1993; Asada *et al.*, 1995; Kitano *et al.*, 1995]. The reason of this trend is that the soccer includes features of problems that the recent AI research is faced with, such as, real-time processing, robustness against noise, cooperations in multi-agent systems, and processing of incomplete information. However, the soccer is not established as a common problems, because we have no common platforms to compare the performance of such systems that use the soccer as an example. RoboCup [Kitano *et al.*, 1995] is proposed in order to provide such a common platform.

RoboCup consists of 3 sections of games: a real-robot section, a simulation section and a special-skill section. In this article, I describe the simulation section and Soccer Server, a simulator used in the section.

2 Soccer Server

2.1 Simulation Section of RoboCup

In the simulation section, two teams of players controlled by competitors' programs fight in a virtual soccer field simulated by a computer. In order to encourage competitors using various programing systems to join this section, we adopt the client-server style. In this style, a server provide the virtual soccer field. Each competitor's program connects to the server as a client, and sends commands to control a player in the field via computer networks. As the server we use Soccer Server described below.

Because abilities of players are controlled to be identical in the server, strategies of teams are compared mainly in this section. For example, cooperation among players like 'passes' becomes an important factor, because passing a ball between teammates is generally more effective than dribbling the ball by a player.

2.2 Overview of Soccer Server

Soccer Server provides a virtual soccer field, in which players controlled by clients run and kick a ball. Fig. 1 and Fig. 2 show its window images.

Soccer Server consists of 3 modules: a field simulator module, a referee module and a message-board module (Fig. 3). The field simulator module calculates movements of objects on the field and checks collisions among them. The referee module controls a game according to rules. The message-board module manages communication among clients.

A client connects with the server by a UDP socket. Using the socket, the client sends commands to control a player of the client and receives information from sensors of the player. Each client can control only one player [1] [2]. All communication between the server and each client is done using by ASCII strings.

2.3 Simulator

2.3.1 Field

The soccer field and all objects on it are 2-dimensional. The size of the field is decided according to the official size of rules of human soccer: The length is 105 and the width is 68. The width of goals is doubled, that is 14.64,

[1] Technically, it is easy to cheat the server. Therefore this is a gentleman's agreement.

[2] In order to test various kind of systems, we may permit a client controls multi-players when each control module of a player is separated logically from each other in the client.

because 2-dimensional simulation makes it difficult to get goals.

2.3.2 Object

There are the following objects on the soccer field:

- Movable objects: a ball and players.
- Visible objects: goals, flags and lines.

Each movable object is a circle that has the following parameters.

- Size
  The radius of the circle of the object. The sizes of players and a ball are 0.5 and 0.15 respectively.
- Position
  The 2-dimensional location of the center of the object in the field.
- Velocity
  The 2-dimensional velocity of the object in the field.
- Acceleration
  The acceleration of the velocity.
- Direction
  The direction of the object. In the case of a player, the object can accelerate along the direction. The direction of a player also means a center direction of view of the player.
- Decay
  The decay parameter of the velocity. If the decay is 1, the object continues to move. If the decay is 0, the object stops immediately.
- Noise Parameters
  The rate of uncertainty of the movement of the object. If this parameter is 0, the object moves exactly according to the Velocity. If this parameter is positive, noises are added to its movements. The magnitude of the noise is in proportion to this parameter and the velocity.

2.3.3 Movement

Movements of objects are simulated stepwise one by one in the following manner:

$$
\begin{aligned}
p_x^{t+1} &= p_x^t + v^t * \cos d^t \\
p_y^{t+1} &= p_y^t + v^t * \sin d^t \\
d^{t+1} &= d^t + m/(v_0 + v^t) \\
v^{t+1} &= (1-q) * v^t + f^t \\
m^{t+1} &= 0 \\
f^{t+1} &= 0
\end{aligned}
$$

If a object overlaps another object, that is, collides with another object after its movement, the object is moved back until it does not overlap other objects. Then its velocity is multiplied by -0.1.

2.4 Protocol

In this section, I describe the protocols of communication between the server and a client.

2.4.1 Open Connection

First of all, a client must open a connection with the server. A client can open a connection in two manners, 'initialization' and 're-connection'. 'Initialization' is a form used when a client wants to be assigned to a new player. 'Re-connection' is a form used when a client wants to be assigned to an existing player. This is used after a half time of a match.

Actual protocols are as follows.

Initialization

1. A client sends the following message to the key port (default 6000) of the server.

   `(init` *Teamname*`)`

2. If the server succeeds to assign a player to the client, the server sends the following message to the port of the client from which the above message is sent.

   `(init` *Side UNum PlayMode*`)`

   where *Side* and *UNum* indicate the side of the team ('`l`' or '`r`') and a uniform number assigned to the player. *PlayMode* is one of '`before_kick_off`', '`kick_off`', '`play_on`', '`throw_in`', '`corner_kick`' and '`goal_kick`'. Meanings of them are described in section 2.5.

   This message is sent from a new port assigned for the client. The client must send control messages described below to this port. In order to do this, the client should be keep the information about the port number described in the socket struct of C.

3. If the server fails to assign a new player, the server sends the following message to the client.

   `(error no_more_team_or_player)`

Re-Connection

1. A client sends the following message to the key port of the server.

   `(reconnect` *Teamname UNum*`)`

2. If the server succeeds to re-connect the player, the server sends the following message to the client.

   `(reconnect` *Side PlayMode*`)`

3. If the server fails to reconnect the player, the server send the following message to the client.

   `(error can_not_reconnect)`

2.4.2 Control Command

A client can send the following commands to control its player.

- (`move` *X Y*)
  Move the player to the position (X, Y). The origin is the center mark, and the X-axis and Y-axis is toward the opponent goal and the right side respectively. So, usually X is negative to locate the player in its own side of the field.
  This command is availably only in the `before_kick_off` mode.

- (`turn` *Moment*)
  Change the direction of the player according to *Moment*. *Moment* should be $-180 \sim 180$. Actual change of the direction is reduced when the player is moving fast.

- (`dash` *Power*)
  Increase the velocity of the player toward its direction according to *Power*. *Power* should be $-30 \sim 100$.

- (`say` *Message*)
  Broadcast *Message* to all players. *Message* is informed immediately to clients using a `(hear ...)` format described below. *Message* should consist of characters of alphabets, numbers and symbols in "`+-*/_.`".

2.4.3 Sensor Information

A client gets two kinds of sensor information about the field from the server, visual and auditory information. The visual and auditory information are informed by `see` and `hear` messages respectively.

- (`see` *Time ObjInfo ObjInfo ...*)
  Inform visual information. *Time* indicates the current time. *ObjInfo* is information about a visible object, whose format is:

  (*ObjName Distance Direction*)
  ObjName ::= (`player` *Teamname UNum*)
  | (`goal` *Side*)
  | (`ball`)
  | (`flag [l|c|r] [t|b]`)
  | (`line [l|c|r|t|b]`)

  As the distance to a player is larger and larger, more and more information about the player loses. Actually, *UNum* is lost in the case of farther than a certain distance, and *Teamname* is lost in the case of very far.
  This message is sent 2 times per second. (The frequency may be changed.)

- (`hear` *Time Direction Message*)
  Inform auditory information. This message is sent immediately when a client sends (`say` *Message*) command. *Direction* is the direction of the sender. *Time* indicates the current time.
  Judgements of the referee is also informed using this form. In this case, *Direction* is '`referee`'. Possible judgements are as follows:
  - `kick_off_l`, `kick_off_r`
  - `goal_l`, `goal_r`
  - `throw_in_l`, `throw_in_r`
  - `corner_kick_l`, `corner_kick_r`
  - `goal_kick_l`, `goal_kick_r`
  - `half_time`, `time_up`

2.5 Rules

The referee module controls a match according to the following rules.

- **Goal**
  When a ball is in a goal, the referee announces the goal (broadcasts a message to all clients), renews the score, moves the ball to the center mark, and changes the play-mode to `kick_off`.

- **Out of Field**
  When a ball is out of the field, the referee moves a ball to a proper position (a touch-line, corner or goal-area) and changes the play-mode to `throw_in`, `corner_kick` or `goal_kick`.

- **Clearance**
  When the play-mode is `kick_off`, `throw_in`, `corner_kick` or `goal_kick`, the referee removes defending players from an area in a circle whose center and radius are the ball and 9.15 respectively.

- **Play-mode Control**
  When the play-mode is `kick_off`, `throw_in`, `corner_kick` or `goal_kick`, the referee changes the play-mode to `play_on` after the ball starts by a `kick` command.

- **Halftime and Time Up**
  The referee suspends a match when the first or the second half finishes. The length of each half is about 10 minutes.

2.6 Process of a Match

A match organized by Soccer Server is carried out in the following steps:

1. Each client of each team connects with the server by an `init` command.

2. When all clients are ready to play, the match commissary (a person who invokes the server) starts the match by pressing the *kick-off* button of the server window. Then the first half starts.

3. The first half is about 10 minutes [3]. When the first half finishes, the server suspend the match.

4. The half-time is 5 minutes. During the half-time, competitors can change client programs.

[3] The length of a half may be changed.

5. Before the second half, each client re-connects with the server by a `reconnect` command.

6. When all clients are ready, the commissary start the second half by pressing the *kick-off* button. Then the second half starts.

7. The second half is also about 10 minutes. After the second half, the server stops the match.

2.7 Implementation

Soccer Server is implemented using g++ and X-window system with Athena widget-set. Currently, I support SunOS 4.1.x on Sparc machines. I am planning to support Solaris 2, DEC OSF/1, and other OSs and architectures.

3 Research Issues

In this section, I discuss several research issues involved in making strong teams for the simulation section of RoboCup.

3.1 Uncertainty

Each client must deal with various kind of uncertainty in controlling its player effectively. In order to make simulated environment similar to one of the real world, I introduced random factors in players' action and restricted its performance in sensing. Therefore, designers of clients must consider about how to process such low-reliable information. For example, the designers may deal with the following problems.

- **Unreliability in Action**
  Control commands sent by clients are not always executed. The commands may be lost in the network or may not be accepted by the server because of timing of sending the message [4]. Moreover, executions of the commands are not accurate. For example, the server adds noise to movements of objects. Therefore the ball may roll on toward an unexpected direction when a client sends a `kick` command. Movements of players are also unreliable. '`(turn 90)`' does not mean that the player turns right. It means that the player changes direction in a certain angle that is decided by the *Power* (= 90), his velocity and noise factor. Therefore clients should have mechanisms to confirm effects of commands sent by itself.

- **Limited Sensor Information**
  Each client can get limited visual information because of narrow view (only 60 degrees). Therefore, in order to get complete information of the field, the client rotates its player slowly [5] using the `turn` command and integrates visual informations sent from the server. In this case, uncertainty of actions becomes a problem. Because `turn` commands are not executed exactly, the client can not integrate visual informations before and after a `turn` command simply.

  Moreover, information about objects in sight is lost when distances to the objects are long. For example, the client gets the team-name and the uniform number of a player in sight if the player is near enough. If the player is distant the client can not get the uniform number, and if the player is very far, the client cannot get even the team-name. Therefore, in order to know such information about players that are very far, the client must infers it from a past history of information sent from the server.

[4] In the current version, the server accepts only the last message on the message cue of each socket at the timing when the server checks the socket.

[5] If the client rotates its player quickly, it can not get complete information because interval of sending visual information is set relatively long.

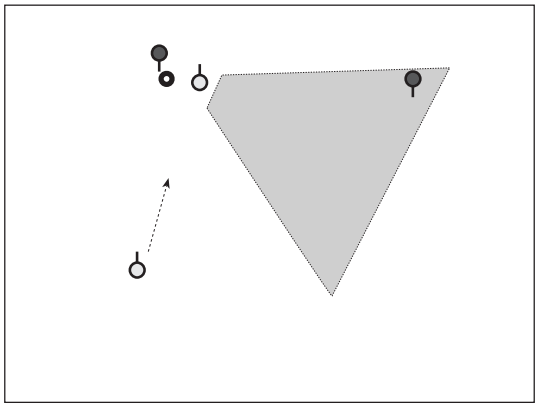

Figure 5: One Side Cut

3.2 Cooperation With Team-mates

Because soccer is a multi-agents game, team-plays, that is cooperations with team-mates, are important factors to decide the ability of teams. For example, the following team-plays become elementary problems.

- **Pass**
  A pass is the most basic team-play. In order to pass a ball among players, the players must have a consensus about pass-courses. Even in the case of pass between two players it is not easy to reach a consensus. For example, consider a situation like Fig. 4-(a). In this case, we can consider two pass-plans: 'one-two-pass'(Fig. 4-(b)) and 'through-pass'(Fig. 4-(c)). In order to break the defense, players of the offense must decide which of the two pass-plans they adopt.

- **One-Side Cut**
  'One-side cut' is a basic combination play in defense. In this play, a defender nearest to an offense keeping a ball chases the offense from left or right side, while other defenders cover another side (Fig. 5). This play involves the same problem as the 'pass', that is, how to reach a consensus on choosing a side to cut.

- **Coaching**
  As mentioned above, each client can get limited

Figure 1: Window Image of Soccer Server

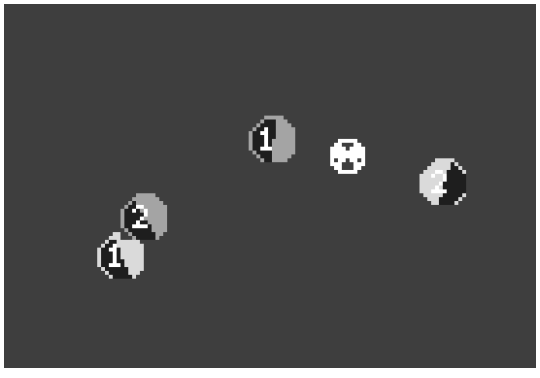

Figure 2: Close-up of Players and a Ball

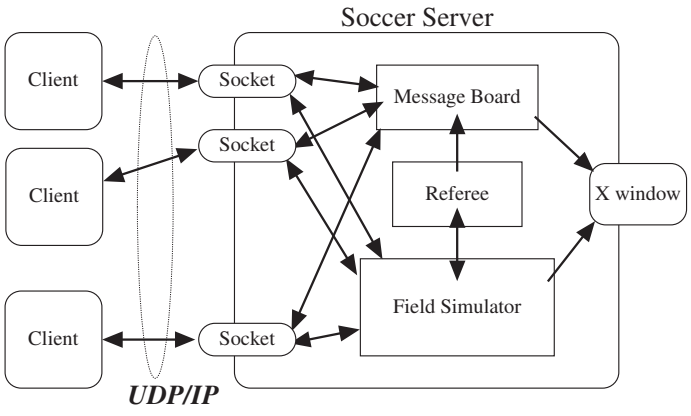

Figure 3: Overview of Soccer Server

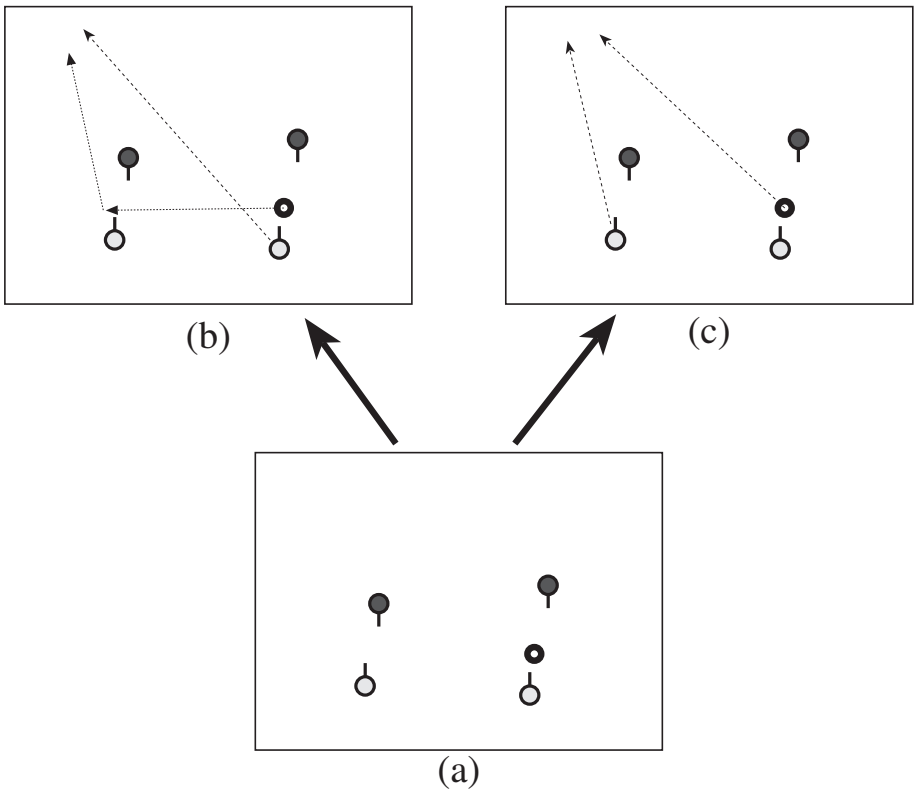

Figure 4: Pass

information about the field. Communicating and sharing such informations among clients is a solution to overcome the problem. In human soccer, such communication are called as 'coaching'. The coaching in the soccer server is not so simple. The server provides a way of communication, the `say` command and the `hear` information, but its capacity is restricted [6]. Therefore the communication should be effective.

4 Conclusion

Soccer Server is distributed freely from the following access points:

Web Home Page:
`http://ci.etl.go.jp/~noda/soccer/server.html`

FTP Service:
`ftp://ci.etl.go.jp/pub/soccer/server/`

Also for further information, please refer the above access points, or join the mailing list, `RoboCup@csl.sony.co.jp`. Any comments and suggestions are welcome to be sent to `noda@etl.go.jp`.

References

[Asada *et al.*, 1995] Minoru Asada, Shoichi Noda, and Koh Hosoda. Non-physical intervention in robot learning based on lfe method. In *Proc. of Machine Learning Conference Workshop on Learning from Examples vs. Programming by Demonstration*, 1995.

[Kitano *et al.*, 1995] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup intiative. In *Working Notes of IJCAI Workshop: Entertainment and AI/Alife*, pages 19–24, Aug. 1995.

[Littman, 1994] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *11th International Conference on Machine Learning*, pages 157–163, 1994.

[Sahota and Mackworth, 1994] Michael K. Sahota and Alan K. Mackworth. Can situated robots play soccer? In *Proc. of Canadian AI-94*, 1994.

[Sahota, 1993] Mickael K. Sahota. Real-time intelligent behaviour in dynamic environments: Soccer-playing robots. matster thesis, Department of Computer Science, The University of British Columbia, Aug. 1993.

[Uchibe *et al.*, 1995] Eiji Uchibe, Minoru Asada, Shoichi Noda, and Koh Hosoda. The coordination of multiple behaviors for a mobil robot acquired by vision-based reinforcement learning. Report SIG-FAI-9403-4, JSAI, Mar. 1995.

[6] The server restricts the frequency of acceptance of `say` commands.