Soccer Server Manual

Rev. 2.00

(for **Soccer Server** Ver.3.00 and later)

NODA Itsuki
noda@etl.go.jp

February 9, 1998

Contents

1 Overview **3**

2 User's Manual **4**
2.1 Getting Started . . . 4
2.1.1 Installation . . . 4
2.1.2 How to Start **Soccer Server** . . . 5
2.1.3 How to Stop **Soccer Server** . . . 5
2.1.4 `sserver` Script . . . 5
2.1.5 Supported Platforms, Required Facilities and More Information . . . 5
2.2 Whole Process of a Match . . . 6
2.3 Parameters . . . 6
2.3.1 Parameters of `soccerserver` . . . 6
2.3.2 Commandline Options of `soccerserver` . . . 7
2.3.3 Configulation File of `soccerserver` . . . 8
2.3.4 Parameters of `soccermonitor` . . . 8
2.3.5 Commandline Options of `soccermonitor` . . . 8
2.3.6 Configulation File of `soccermonitor` . . . 8
2.4 Protocol . . . 8
2.5 Rules . . . 8
2.5.1 Rules Judged by the Server . . . 8
2.5.2 Rules Judged by Human . . . 9
2.6 Coach . . . 9
2.6.1 Coach Mode . . . 9
2.6.2 Coach with Referee Mode . . . 10

3 Inside of Soccer Server **11**
3.1 Field and Objects . . . 11
3.1.1 Field . . . 11
3.1.2 Players and Ball . . . 11
3.1.3 Movements of Objects . . . 11
3.1.4 Player's Action . . . 12
3.2 Sensor Information . . . 12
3.2.1 Visual Information . . . 12
3.2.2 Auditorial Information . . . 15
3.3 `soccermonitor` . . . 16
3.3.1 from server to monitor . . . 16

A Table of Protocols **18**
A.1 Open Connection . . . 18
A.2 Control Commands . . . 18
A.3 Sensor Information . . . 19
A.4 Coach Commands . . . 19
A.5 Object Names . . . 19
A.6 Play Modes . . . 20
A.7 Restriction of Message . . . 20

B Know How **21**

C FAQ **22**
C.1 Start up **Soccer Server** 22
C.2 Making Clients 23
C.3 Visual Information 25
C.4 Inside of **Soccer Server** 26

Chapter 1

Overview

Soccer Server is a system that enables various types of programming languages to play a match of soccer (association football) against each other.

A match is carried out in a server-client style: A server, **Soccer Server** , provides a virtual field and simulate all movements of a ball and players. Clients become brains of players that control movements of players. Communication between a server and each client is done via UDP/IP sockets. Therefore users can use any kinds of programing systems that have UDP/IP facilities.

Soccer Server consists of 2 programs, `soccerserver` and `soccermonitor`. `soccerserver` is a server program that simulates movements of a ball and players, communicates with clients, and controls a game according to rules. `soccermonitor` is a program that displays a virtual field in the server using the X window system. A number of `soccermonitor` can connects with a `soccerserver`, so we can display field-windows on multiple displays.

A client connects with `soccerserver` by a UDP socket. Therefore users can create client programs by any kinds of programing systems that has UDP/IP interfaces. Using the socket, the client sends commands to control a player of the client and receives information from sensors of the player. In other words, a client program is a brain of the player: The client receives visual and auditory sensor information from the server, and sends control-commands to the server.

Each client can control only one player [1] [2]. So, a user must invoke the same number of clients as players. Communications between the clients must be done via `soccerserver` using `say` and `hear` protocols. (See Section **??**.) One of purposes of **Soccer Server** is evaluation of multi-agent systems, in which efficiency of communication between agents is one of criterions. Users must realize control of multiple clients by such restricted communication.

About This Manual

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

[2] In order to test various kinds of systems, we may permit a client controls multiple players if the different control modules of players are separated logically from each other in the client.

Chapter 2

User's Manual

2.1 Getting Started

2.1.1 Installation

1. Get source files from the following FTP cite:

   ```
   ftp://ci.etl.go.jp/pub/soccer/server/sserver-*.tar.gz
   ```

   where * is the version number.

2. Extract source files like the following operation:

   ```
   % gzip -dc sserver-*.tar.gz | tar xvf -
   ```

3. Change working directory to "`sserver-*`".

   ```
   % cd sserver-*
   ```

4. Do "`configure`" and answer questions about directories.

   ```
   % configure
   Enter g++-include directory.
     [default=/usr/lib/g++-include]:/usr/lib/g++-include
   Enter bsd directory.
     [default=/usr/include/bsd]:/usr/include/bsd
   Do you use X11R6?[y or n]:y
   Enter X11R6 include directory.
     [default=/usr/X11R6/include]:/usr/X11R6/include
   Enter X11R6 libraries directory.
     [default=/usr/X11R6/lib]:/usr/X11R6/lib
   Do you use dynamic linking?[y or n]:y
   Configuration Summary:
     OS type = Linux
     g++-include PATH = /usr/lib/g++-include
     bsd PATH = /usr/lib/g++-include
     X11 revision = 6
     X11 include PATH = /usr/X11R6/include
     X11 libraries PATH = /usr/X11R6/lib
     Link style = Dynamic
   Creating Makefile...[server][monitor][sampleclient][recfile_change][logplayer]. Done.
   ```

5. Do "`make`".

   ```
   % make
   ```

2.1.2 How to Start Soccer Server

1. Start `soccerserver`.
2. Start `soccermonitor`.
3. Start clients, which open a connection with `soccerserver`.
4. When all clients are ready, click the `kick-off` button on a window of `soccermonitor`.

2.1.3 How to Stop Soccer Server

1. Stop all clients.
2. Stop all `soccermonitor`s by clicking the `quit` buttons.
3. Hit CNTL-C to `soccerserver`, or kill `soccerserver`.

2.1.4 `sserver` Script

In order to make simple to start and stop **Soccer Server** , the distribution includes script file "`sserver`". This script does the following things:

1. Setup environment variables.
2. Fork "`soccerserver`" with standard configuration file.
3. Start"`soccermonitor`" with standard configuration file, and wait it ends.
4. Kill "`soccerserver`".

This script uses "`tcsh`". You may modify the path of `tcsh` in the 1st line of the script.

2.1.5 Supported Platforms, Required Facilities and More Information

Currently, **Soccer Server** supports the following platforms.

- DEC Alpha with OSF/1
- Sun Sparc with SunOS 4.1.x and 5.x
- SGI Indy/Indigo with IRIX
- i386 with Linux
- Sony WS with NewsOS

Soccer Server requires the following facilities.

- gcc verions 2.7.x or later
- libg++ version 2.7.x or later
- X11R6 and Athena widgets

Soccer Server is freely distributed 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 to 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`.

2.2 Whole 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 [1]. 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.
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.
8. If the match is a draw, an extra half starts. The extra half ends immediately when a team gets a new goal (Victory-goal style).

2.3 Parameters

2.3.1 Parameters of `soccerserver`

`soccerserver` has the following modifiable parameters.

goal_width (default = 14.02)
: Width of the goal.

player_size (default = 0.8)
: Radius of a player.

player_decay (default = 0.4)
: Decay rate of movements of a player. If this is 1.0, a player keeps its movements, and if this is 0.0, a player loses its speed in a simulation cycle.

player_rand (default = 0.1)
: Amount of noise added in player's movements and turns.

player_weight (default = 60.0)
: Weight of a player. This parameter concerns with the wind factor.

ball_size (default = 0.085)
: Radius of the ball.

ball_decay (default = 0.96)
: Decay rate of movements of the ball.

ball_rand (default = 0.05)
: Amount of noise added in ball's movements.

ball_weight (default = 60.0)
: Weight of the ball. This parameter concerns with the wind factor.

stamina_max (default = 2000.0)
: Maximum stamina of a player.

stamina_inc (default = 20.0)
: Amount of stamina that a player gains in a simulation cycle.

hear_max (default = 2)
hear_inc (default = 1)
hear_decay (default = 2)
: These three parameters control the capacity of hearing of player. Roughly speeking, a player can listen $N(=$ **hear_inc**$)$ messages in $M(=$ **hear_decay**$)$ simulation cycles.

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

dash_power_rate (default = 0.010)
Rate by which *Power* argument in `dash` command is multiplied.

kick_power_rate (default = 0.013)
Rate by which *Power* argument in `kick` command is multiplied.

visible_angle (default = 90)
Angle of view cone of a player in the standard view mode.

audio_cut_off (default = 50.0)
Maximum distance a message said by a player can reach.

maxpower (default = 100)
minpower (default = -30)
Maximum and minimum value of *Power* in `dash` and `kick` commands.

maxmoment (default = 180)
minmoment (default = -180)
Maximum and minimum value of *Moment* and *Direction* in `turn` and `kick` commands.

wind_force (default = 0.0)
The amount of force of wind.

wind_dir (default = 0.0)
The direction of wind.

wind_rand (default = 0.0)
The amount of change of direction of wind.

quantize_step (default = 0.1)
The step of quantiazation of distance for moving objects (players and ball).

quantize_step_l (default = 0.01)
The step of quantiazation of distance for landmarks (flags, lines and goals).

simulator_step (default = 100)
Length of period of simulation cycle. Unit is milli-second.

send_step (default = 300)
Length of period of sending visual information to a player in the standard view mode. Unit is milli-second.

recv_step (default = 10)
Length of period of server's polling sockets to clients. Unit is milli-second.

half_time (default = 300)
The length of a half time of a match. Unit is simulation cycle. In other words, actual length of a match is 2 × half_time × simulator_step [ms].

say_msg_size (default = 512)
Maximum length of a massage a player can `say`.

port (default = 6000)
The port number of UDP/IP that connects with player-clients and `soccermonitor`.

coach_port (default = 6001)
The port number of UDP/IP that connects with coach-clients.

2.3.2 Commandline Options of `soccerserver`

A user can modify parameters described in Section 2.3.1 specifying in the command line as follows.

`% soccerserver [-`*ParameterName Value*`]*`

`soccerserver` also accept the following options.

- `-f` *ConfFineName*
  See Section 2.3.3.
- `-coach`
  See Section 2.6.1.
- `-coach_with_referee`
  See Section 2.6.2.

2.3.3 Configulation File of `soccerserver`

A user also can modify parameters specifying by a configulation file as follows.

`% soccerserver -f` *ConfFileName*

In the configulation file, each line consists a pair of name and value of a parameter as follows.

ParameterName : *Value*

In the file, lines that start '`#`' are ignored as comments.

2.3.4 Parameters of `soccermonitor`

`soccermonitor` has the following modifiable parameters.

print_log (default = on)
: Flag for display log of communication. The value must be `on` or `off`.

host (default = localhost)
: The host name on which `soccerserver` is running.

port (default = 6000)
: The port number of UDP/IP of `soccerserver`.

length_magnify (default = 6.0)
: Rate of magnification of size of the field.

player_widget_size (default = 12.0)
: Size of a player widget.

log_line (default = 6)
: Size of log window.

goal_width (default = 14.02)
: Width of the goal.

2.3.5 Commandline Options of `soccermonitor`

A user can modify parameters described in Section 2.3.4 specifying in the command line as follows.

`% soccermonitor [-`*ParameterName Value*`]*`

2.3.6 Configulation File of `soccermonitor`

A user also can modify parameters specifying by a configulation file as follows.

`% soccermonitor -f` *ConfFileName*

In the configulation file, each line consists a pair of name and value of a parameter as follows.

ParameterName : *Value*

In the file, lines that start '`#`' are ignored as comments.

2.4 Protocol

2.5 Rules

2.5.1 Rules Judged by the Server

Soccer Server 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, suspend the match 5 seconds, moves the ball to the center mark, and changes the play-mode to `kick_off`. While the referee suspends the match 5 seconds, clients must return to their own side. During this interval, clients can use "move" commands. If a player remains in the oppenet side, the referee moves the plaeyr to a random position in their own side (See "Kick-off" rule.)

- **Kick-off**
  At the kick off, all players must be in their own side. If players are in the opponent side, the referee moves them into their own side randomly.

  There is a period of 5 seconds for players to go back to their own side after a goal. During the interim, clients can use the move command to move their player to a certain position immediately. After the time expires, players in the opponent side are moved randomly by the server.

- **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 `kick_in`, `corner_kick` or `goal_kick`.

  In the case of a corner kick, the referee sets the ball at (1m,1m)-inside from the corner.

- **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 5 minutes (3000 simulation-cycles). If the match is draw after the second half, the match is extended until another goal (victory-goal style).

2.5.2 Rules Judged by Human

Fouls like 'obstruction' are difficult to judge automatically, because they concern players' intentions. Therefore, the server prepares a way for human to judge such fouls.

- **Un-gentlemanly Play**
  If a player is too un-gentle or interferes a match, the match-commissary (an organizer of the match) can suspend a match by clicking on the player in the field window and restarting the match by a free kick for the opposite team.

2.6 Coach

2.6.1 Coach Mode

Like human soccer, a match is played without interruptions from the outside, and nobody (and no program) can control the match. However, this is not useful when a programmer is developing and training client-programs. In order to make it easy to develop client-programs, the server has 'coach mode'.

If **Soccer Server** is invoked with the '`-coach`' option, the server runs in coach mode. In coach mode, a privileged client (called a 'coach client') takes the place of the referee module in the server. The coach client can do the following operations.

- Control play mode.
- move objects (players and a ball).
- Get information about positions of all objects on the field.
- Broadcast a message as a judgement of the referee.

The server also has another option '`-coach_w_referee`'. Using this option, a coach client can connect the server in which the referee module is still running. In this case, both of the referee module and the coach client can control a match.

Initialization

If the server is invoked with coach mode, it prepares a UDP socket bound to the coach port (default 6001). Then, the server receives all messages sent to this port as commands of a coach client described below.

Commands of Coach

- (`change_mode` *PLAY_MODE*)
  Change the play-mode to *PLAY_MODE*. *PLAY_MODE* should be one of `before_kick_off`, `kick_off`, `play_on`, and `kick_in`.

- (`move` *OBJECT X Y* [*VDir*])
  Move *OBJECT* (players or a ball) to (X, Y), and change the direction of *OBJECT* to *VDir*.

- (`look`)
  Get information about positions of all objects on the field. The server returns

  (`look` *TIME* (*OBJ1 X1 Y1*) (*OBJ2 X2 Y2*)...).

- (`say` *MESSAGE*)
  Broadcast a *MESSAGE* to all clients.

- (`check_ball`)
  Check the position of the ball. The server returns

  (`ok check_ball` *TIME BALLPOSITION*).

  BALLPOSITION will be one of `in_field`, `goal_l`, `goal_r`, and `out_of_field`.

- (`ear` *MODE*)
  Turn on or off sending auditory informations. *MODE* must be one of `on` and `off`. If (`ear on`) is sent, the server sends all auditory information to the coach client in the same manner as to player-clients. If (`ear off`) is sent, the server stops sending auditory information to the coach client.

2.6.2 Coach with Referee Mode

Soccer Server has also the coach-with-referee mode. In this mode, a coach client can connect with the server in the same way as the coach mode described the previous section, and the referee module of **Soccer Server** also controls a match. In order to start the server in this mode, you start `soccerserver` with '`-coach_with_referee`'.

Using this mode, the coach client can focus to suggest player-clients without controling a match by the rules described in Section 2.5.1.

Chapter 3

Inside of Soccer Server

3.1 Field and Objects

3.1.1 Field

The soccer field and all objects on it are 2-dimensional. This means that only horizontal movements are taken into account and vertical movements are ignored. The size of the field is 105m × 68m [1].The width of the goals is 14.64m, which is double the size of ordinary goals, because 2-dimensional simulation makes it difficult to score goals.

3.1.2 Players and Ball

Players and ball are treated as circles. Their movements are simulated stepwise one by one.

3.1.3 Movements of Objects

In each simulation step, movement of each object is calculated as following manner:

$$\begin{aligned}
(u_x^{t+1}, u_y^{t+1}) &= (v_x^t, v_y^t) + (a_x^t, a_y^t) \\
(p_x^{t+1}, p_y^{t+1}) &= (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}) \\
(v_x^{t+1}, v_y^{t+1}) &= \text{decay} \times (u_x^t, u_y^t) \\
(a_x^{t+1}, a_y^{t+1}) &= (0, 0)
\end{aligned} \tag{3.1}$$

where, (p_x^t, p_y^t), and (v_x^t, v_y^t) are respectively position and velocity of the object in timestep t. decay is a decay parameter specified by `ball_decay` or `player_decay`. (a_x^t, a_y^t) is accelaration of object, which is derived from *Power* parameter in `dash` (in the case the object is a player) or `kick` (in the case of a ball) commands as the following manner:

$$(a_x^t, a_y^t) = Power \times (\cos(\theta^t), \sin(\theta^t))$$

where θ^t is the direction of the object in timestep t. In the case of a player, its direction is calculated as the following manner:

$$\theta^t = \theta^t + Moment$$

where *Moment* is a parameter of `turn` commands. In the case of a ball, its direction is given as the following manner:

$$\theta^t_{\text{ball}} = \theta^t_{\text{kicker}} + Direction$$

where θ^t_{ball} and θ^t_{kicker} are directions of ball and kicking player respectively, and *Direction* is the second parameter of a `kick` command.

[1] The unit is meaningless, because parameters used in the simulation do not be decided based on the physical parameters, but be tuned in order to make matches be meaningful as evaluation of multi-agent systems.

Collitions

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.

Noise and Winds

In order to reflect unexpected movements of objects in real world, **Soccer Server** add noise to objects' movements and parameters of commands.

Concerned with movements, noise is added into Eq. 3.1 as follows:

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) + (\tilde{r}_{\mathrm{rmax}}, \tilde{r}_{\mathrm{rmax}}) \tag{3.2}$$

where $\tilde{r}_{\max}$ is a random number whose distribution is uniformed between the range $[-\max, \max]$. rmax is a parameter that depends on amount of velocity of the object as follows:

$$\mathrm{rmax} = \mathrm{rand} \cdot |(v_x^t, v_y^t)|$$

where rand is a parameter specified by `player_rand` or `ball_rand`.

Noise is added also into a *Power* parameter in a `turn` command as follows:

$$Moment = (1 + \tilde{r}_{\mathrm{rand}}) \cdot Moment$$

3.1.4 Player's Action

Three Basic Action

`turn`

`dash`

`kick`

Stamina

Each player has its own stamina. The player can `dash` with *Power* lower than the stamina. The stamina decreases by the *Power*, and increases by the `stamina_inc`. Actually, these values are operated as follows:

$$\begin{aligned} Power &:= \min(Power, Stamina) \\ Stamina &:= Stamina - Power \end{aligned}$$

$$Stamina := \min(Stamina + SInc, SMax)$$

3.2 Sensor Information

3.2.1 Visual Information

Format

The format of 'see' information will be changed as follows:

(ObjName Distance Direction DistChng DirChng)

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

Table 3.1: Control Command

(`move` *X Y*) Move the player to the position (X, Y). The origin is the center mark, and the X-axis and Y-axis are toward the opponent's goal and the right touchline respectively. Thus, X is usually negative to locate a player in its own side of the field. This command is available only in the `before_kick_off` mode.
(`turn` *Moment*) Change the direction of the player according to *Moment*. *Moment* should be $-180 \sim 180$. The actual change of direction is reduced when the player is moving quickly.
(`dash` *Power*) Increase the velocity of the player in the direction it is facing by *Power*. *Power* should be $-30 \sim 100$. (The range is modifiable by the parameter file.)
(`kick` *Power Direction*) Kick the ball by *Power* to the *Direction* if the ball is near enough (the distance to the ball is less than 1.0+`ball_size`+`player_size`.). *Power* should be $-30 \sim 100$, and *Direction* should be $-180 \sim 180$. (The ranges are modifiable by the parameter file.)
(`say` *Message*) Broadcast *Message* to all players. *Message* is informed immediately to clients as sensor information in the (`hear ...`) format described below. *Message* must be a string whose length is less than 512, and consist of alphanumeric characters and the symbols "`+-*/_.()` ".
(`change_view` *ANGLE_WIDTH QUALITY*) Change angle of view cone and quality of visual information. *ANGLE_WIDTH* must be one of `wide` (=180 degrees), `normal` (= 90 degrees) and `narrow` (=45 degrees). *QUALITY* must be one of `high` and `low`. In the case of `high` quality, the server begins to send detailed information about positions of objects to the client. In the case of `low` quality, the server begins to send reduced information about positions (only directions) of objects to the client. Default values of angle and quality are `normal` and `high` respectively. On the other hand, the frequency of visual information sent by the server changes according to the angle and the quality: In the case that the angle is normal and the quality is high, the information is sent every 300 milli-sec. (The interval is modifiable by specifying `send_step` in the parameter file.) When the angle changes to wide the frequency is halved, and when the angle changes to narrow the frequency is doubled. When the quality is low, the frequency is doubled. For example, when the angle is narrow and the quality is low, the information is sent every 75 milli-sec.

Table 3.2: Sensor Information

(see *Time ObjInfo ObjInfo ...*)

Transmit visual information. *Time* indicates the current time. *ObjInfo* is information about a visible object in the following format:

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

where *Distance* and *Direction* are the relative distance and direction of the object respectively, and *DistChng* and *DirChng* are respectively the radius- and angle-components of change of the relative position of the object. *DistChng* and *DirChng* are not exact values but are sketchy values. For more detail, please see Section **??**. If the object is not movable, *DistChng* and *DirChng* are omitted.

The characters "l, c, r, t, b" in flags and lines mean left, center, right, top and bottom respectively, that is positions of flags. So, (flag l t) means the flag on upper-left corner, and (flag c b) means the flag on the lower end of the center line. (flag p ...) is virtual flags on penalty area. So, (flag p l t) means the position of the upper-right corner of the left penalty area.

In the case of (line ...), *Distance* is the distance to the point there the center line of the player's view crosses the line, and *Direction* is the direction of line.

As the distance to another player becomes larger and larger, more and more information about the player is lost. Specifically, *UNum* is lost in the case of players further than a certain distance, and *Teamname* is lost from players who are very far. *Vdirection* is also lost when the object is too far.

The quality of information is also changed by change_view commands. (See the description of change_view in Section 3.1.)

The frequency of sending this information depends on the quality of information transmitted. See the description of change_view.

(hear *Time Direction Message*)

Transform auditory information. *Direction* is the direction of the sender. In the case that the sender is the client itself, Direction is 'self'. *Time* indicates the current time. This message is sent immediately when a client sends a (say *Message*) command.

Judgments of the referee are also broadcast using this form. In this case, *Direction* is 'referee'. Possible judgments are as follows:

before_kick_off, kick_off_l, kick_off_r,
kick_in_l, kick_in_r, corner_kick_l, corner_kick_r, goal_kick_l,
goal_kick_r, free_kick_l, free_kick_r, play_on, half_time, time_up,
extend, foul_*Side_UNum*, goal_*Side_Point*,

Distance, *Direction*, *DistChng* and *DirChng* are calculated by the following way:

$$
\begin{aligned}
p_{rx} &= p_{xt} - p_{xo} && (3.3)\\
p_{ry} &= p_{yt} - p_{yo} && (3.4)\\
v_{rx} &= v_{xt} - p_{xo} && (3.5)\\
v_{ry} &= v_{yt} - p_{yo} && (3.6)\\
Distance &= \sqrt{p_{rx}^2 + p_{ry}^2} && (3.7)\\
Direction &= \arctan(p_{ry}/p_{rx}) - a_o && (3.8)\\
e_{rx} &= p_{rx}/Distance && (3.9)\\
e_{ry} &= p_{ry}/Distance && (3.10)\\
DistChng &= (v_{rx} * e_{rx}) + (v_{ry} * e_{ry}) && (3.11)\\
DirChng &= [(-(v_{rx} * e_{ry}) + (v_{ry} * e_{rx}))/Distance] * (180/\pi) && (3.12)\\
& && (3.13)
\end{aligned}
$$

where (p_{xt}, p_{yt}) is the absolute position of a target object, (p_{xo}, p_{yo}) is the absolute position of the sensing player, (v_{xt}, v_{yt}) is the absolute velocity of the target object, (v_{xo}, v_{yo}) is the absolute velocity of the sensing player, and a_o is the absolute direction the sensing player is facing to. In addition to it, (p_{rx}, p_{ry}) and (v_{rx}, v_{ry}) are respectively the relative position and the velocity of the target, and (e_{rx}, e_{ry}) is the unit vector that is parallel to the vector of the relative position.

Range of View

The player can 'see' objects in one of the following areas:

View Cone The relative direction of the object is within $[-45, 45]$ in the case of normal view. In the case of wide and narrow view, the direction is within $[-90, 90]$ and $[-22.5, 22.5]$ respectively.

Neighborhood The distance to the object is within 3m. When the object is in the neighborhood and out of the view cone, the player can know only the type of the object (ball, player, goal or flag), but does not know the exact name of the object. Moreover, the case of neighborhood, the capitalized name, that is "`Ball`", "`Player`", "`Goal`" and "`Flag`, is used as the name of the object rather than "`ball`", "`player`", "`goal`" and "`flag`,

Unreliability of Information about Far Objects

In the case that an object in sight is a ball or a player, The value of distance to the object is quantized in the following manner:

$$d' = \text{Quantize}(\exp(\text{Quantize}(\log(d), 0.1)), 0.1) \quad (3.14)$$

where d and d' are the exact distance and quantized distance respectively, and

$$\text{Quantize}(V, Q) = \text{rint}(V/Q) \cdot Q \quad (3.15)$$

This means that players can not know the exact positions of very far objects. For example, when distance is about 100.0, the maximum noise is about 10.0, while when distance is less than 10.0, the noise is less than 1.0.

In the case of flags and lines, the distance value is quantized in the following manner.

$$d' = \text{Quantize}(\exp(\text{Quantize}(\log(d), 0.01)), 0.1) \quad (3.16)$$

View of Lines

View Mode and Frequency

3.2.2 Auditorial Information

Format

Range of Communication

A message said by player-A is informed only to players within 50m from player-A. For example, a defender, who may be near of his own goal, can hear a message from his goal-keeper (a player in front of the own goal), but a striker who is near the opponent goal can not hear the message.

Capacity of Hear

A player can hear only 1 message from each team during 2 simulation cycles.
1 message is chosen according to the order of arrival.

3.3 soccermonitor

soccerserver and soccermonitor are connected via UDP/IP.

3.3.1 from server to monitor

soccerserver sends a dispinfo_t struct in the following list to soccermonitor.

```
typedef struct {
        short   enable ; /* alive or not */
        short   side ;   /* left team, right team or ball */
        short   unum ;   /* the uniform number */
        short   angle ;  /* direction player is facing */
        short   x ;      /* position X */
        short   y ;      /* position Y */
} pos_t ;

typedef struct {
        char    name[16] ;/* name of a team */
        short   score ;   /* the number of goals */
} team_t ;

typedef struct {
        char    pmode ;   /* playmode */
        team_t  team[2] ; /* information of both teams */
        pos_t   pos[23] ; /* positions of all objects */
        short   time ;    /* simulation cycle */
} showinfo_t ;

typedef struct {
        short   board ;         /* left window or right window */
        char    message[2048] ; /* contents of message */
} msginfo_t ;

typedef struct {
        short   mode ;
        union {
                showinfo_t      show ;
                msginfo_t       msg ;
        } body ;
} dispinfo_t ;
```

dispinfo_t contains one of showinfo_t or msginfo_t. showinfo_t is information about positions of all objects in the field. msginfo_t is information about messages shown in bottom windows. One showinfo_t is sent per 100ms (=simulation_cycle). msginfo_t is sent when it required. These data are transmited directly: soccerserver sends the data as follows:

```
dispinfo_t dint ;
    /* fill dint data */
    ...
sendto(socketfd,(char *)(&dint),sizeof(dispinfo_t),0,
       &mon_addr,sizeof(mon_addr)) ;
```

, while soccermonitor receives the data as follows:

```
dispinfo_t dint ;
recvfrom(socketfd,(char *)(&dint),sizeof(dispinfo_t),0,
       &serv_addr,sizeof(serv_addr)) ;
```

In order to keep compatibility between different platforms, values in `dispinfo_t` are represented by network byte order.

The details of these structures may be changed. See the source file "server/types.h".

Appendix A

Table of Protocols

A.1 Open Connection

from client to server	fron server to client
(`init` *TeamName*)	(`init` *Side UniformNumber PlayMode*) *Side* ::= `l` | `r` *UniformNumber* ::= `1` ∼ 11 *PlayMode* ::= one of play modes (`error no_more_team_or_player`)
(`reconnect` *TeamName UniformNumber*)	(`init` *Side UNum PlayMode*) *Side* ::= `l` | `r` *UNum* ::= `1` ∼ `11` *PlayMode* ::= one of play modes (`error no_more_team_or_player`) (`error reconnect`)

A.2 Control Commands

(from a client to a server)

(`move` *X Y*) *X* ::= `-54` ∼ `54` *Y* ::= `-32` ∼ `32`
(`turn` *Moment*) *Moment* ::= `-180` ∼ `180`
(`dash` *Power*) *Power* ::= `-30` ∼ `100`
(`kick` *Power Direction*) *Power* ::= `0` ∼ `100` *Direction* ::= `-180` ∼ `180`
(`say` *Message*) *Message* ::= a message (See Section A.7.)
(`change_view` *Width Quality*) *Width* ::= `narrow` | `normal` | `wide` *Quality* ::= `high` | `low`

A.3 Sensor Information

(from a server to a client)

(see *Time ObjInfo ObjInfo ...*) *Time* ::= simulation cycle of the soccerserver *ObjInfo* ::= (*ObjName Distance Direction DistChange DirChange*) | (*ObjName Distance Direction*) | (*ObjName Direction*) *ObjName* ::= one of object name (See Section A.5.) *Distance* ::= positive real number *Direction* ::= real number *DistChange* ::= real number *DirChange* ::= real number
(hear *Time Sender Message*) *Time* ::= simulation cycle of the soccerserver *Sender* ::= referee | self | *Direction* *Direction* ::= -180 ∼ 180 *Message* ::= a message sent by a say command
(error ...)

A.4 Coach Commands

from coach client to server	fron server to coach client
(move *ObjectName X Y* [*Angle*]) *ObjectName* ::= one of object names *X* ::= -54 ∼ 54 *Y* ::= -32 ∼ 32 *Angle* ::= -180 ∼ 180	(ok move) (error illegal_object_form) (error illegal_command_form)
(look)	(ok look *Time ObjInfo ObjInfo ...*) *Time* ::= simulation cycle of the soccerserver *ObjInfo* ::= (*ObjName X Y*) *ObjName* ::= one of object names *X* ::= real number *Y* ::= real number
(check_ball)	(ok check_ball *Time Status*) *Time* ::= simulation cycle of the soccerserver *Status* ::= in_field | goal_*Side* | out_of_field
(say *Message*) *Message* ::= a message (See Section A.7.)	(ok say) (error illegal_command_form)
(ear *OnOff*) *OnOff* ::= on | off	(ok ear *OnOff*) (error illegal_command_form)
(change_mode *NewPlayMode*) *NewPlayMode* ::= one of play modes	(ok change_mode) (error illegal_command_form)
	(hear *Time Sender Message*) *Time* ::= simulation cycle of the soccerserver *Sender* ::= name of the sender *Message* ::= a message

A.5 Object Names

(ball)
(player *TeamName UniformNumber*)
(player *Side UniformNumber*)
(goal *Side*)
(flag {l | c | r} {t | b})
(flag p {l | r} {t | c | b})
(line {l | r | t | b})

Object Names Sensed by Short Distance Sensor

(Ball)
(Player)
(Flag)
(Goal)

A.6 Play Modes

before_kick_off
time_over
play_on
kick_off_*Side*
kick_in_*Side*
free_kick_*Side*
corner_kick_*Side*
goal_kick_*Side*
goal_*Side*

A.7 Restriction of Message

A *Message* said by clients must be an ascii string less than 512 charactors [1], which consists of numbers, alphabets and symbols in "(). +-*/?<>_".

Message from Referee

goal_*Side_Point*
foul_*Side_UniformNumber*
half_time
time_up
time_up_without_a_team
time_extended
one of play modes (See Section A.6.)

[1] The length limitation is modifiable by parameters. See Section 2.3

Appendix B

Know How

How to build your clients

Appendix C

FAQ

C.1 Start up Soccer Server

Q. 1
When I attempt to run `soccermonitor`, I get the message

```
ld.so: libSM.so.6: not found
```

A. 1
In this case, please try the following instruction.

1. Change directory to "`monitor/`".
2. Edit setting of "XLIB" in "`Makefile`" as follows (adding "`-lSM -lICE`" options).

```
XLIB = -L/opt/X11R6/lib -lXaw -lSM -lICE -lXmu -lXt -lXext\
           $(XLINK_Dynamic) -lX11
```

3. Remake `soccermonitor`.

Q. 2
When I attempt to run `soccermonitor`, I get the message

```
ld.so: lib???.so.???: not found
```

A. 2
You must configure `LD_LIBRARY_PATH` in `sserver` script by hand. For example, you may change the 3rd lines of `sserver` as follows:

```
setenv LD_LIBRARY_PATH /usr/X11R6/lib:/usr/ucblib:/usr/lib:/lib
```

Q. 3
When I start the soccerserver, I get a message saying

```
can't bind local address.
```

Why?

A. 3
Can't bind local address usually happens when there is an old soccerserver process running. Try

```
ps -aux | grep soccerserver
```

If there are old processes running, kill them and try again.

C.2 Making Clients

Q. 4
The server does not execute "`(move X Y)`" after `init` command is established.

A. 4
There are two possibility:

- When the server accepts an init command from a client, it allocates new port for the client. The client must send commands to the new port. The client can know the number of new port by the socket information the returned "`init`" message from the server. Please see the definition of '`init_connection()`' function in "sampleclient/client.c" in the distribution.
- Move commands are avairable only before kick-off, so that you can not use them after you crick kick-off button.

Q. 5
Why does the sample client continuously receive

```
recv 2035 : (see 0 ((goal r) 73 -7) ((flag r t) 84.8 -31) ((flag r b)
76.7 18) ((flag p r t) 63.4 -28) ((flag p r c) 56.8 -10) ((flag p r b)
56.8 10) ((ball) 22.2 -26 0 0) ((line r) 72.2 -90)
```

is that just the automatic sensory responce from the server to the clients?

A. 5
Yes. This is visual sensor information sent from the server every 300ms. This means,

Now, TIME is 0 (may be before kick-off),
right goal (goal r) is 73m far away and 7 degrees left
flags of top-right corner is 84.8m far away and 31 degrees left
...
ball is 22.2m far away and 26 degrees left, and it's speed is (0,0)
...

The server also sends auditorial information that informs messages from referee and other players. These information is sent automatically, so you should make a client that always polls the socket that connects to the server.

Q. 6
How do I to set up a socket connection to the server as non-blocking?

A. 6
Here is a sample of C/C++ code to set up a socket connection as non-blocking.

```
    /*
     *  Open UDP socket.
     */
    if( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0){
        return sock ;   /* Can't open socket. */
    }

   val = fcntl(sockfd, F_GETFL, 0) ;
   /* 5/17/96 Dave found that O_NONBLOCK didn't work for setting up a*/
   /* non-blocking socket.  O_NDELAY seems to work, though!          */
   /*    val |= O_NONBLOCK ;*/
    val |= O_NDELAY;
    fcntl(sockfd, F_SETFL, val) ;
```

Q. 7

My clients can not get newest visual information, because a couple of message from the server is on the queue of the socket. How do clients get the newest message from the queue?

A. 7

Here is a sample of C/C++ code that clean up queue and get the newest "see" information. If there are multiple "see" messages on the socket, this code prints the old ones, but skips them. This makes sure all hear messages are received, though.

```
          if(FD_ISSET(sock.socketfd, &readfds)){
            /*n = receive_message(buf, MAXMESG, sock, &port) ; */
            /*keep receiving messages until the socket is empty*/
            while((n = receive_message(tmp, MAXMESG, sock, &port)) > 0){
              int copy = TRUE;          /* in general, copy tmp to buf  */

              if (MessageCounter > 0){  /* not first time through while */
                if ( initialized && tmp[1]=='h' ){/* New msg is a hear  */
                  ParseSensoryData(buf, Memry);   /* parse the sound    */
                  if ( Memry->HeardNewSound() ){  /* respond to it      */
                    Communicate(behavior, Memry, &sock);
                    Memry->ClearSound();          /* Undo the sound bit */
                  }
                  copy = FALSE;          /* Leave the last msg in buf   */
                }
#if PRINT_MISSED_MSGS
              else if ( initialized )
                printf(``Player %d-%c skipped a message %.40s ...\n
        before  %.40s ...\n'',
                       Memry->MyNumber,Memry->MySide,buf,tmp);
#endif
            }
            if (copy)
              strcpy(buf,tmp);
            if ( buf[1] == 'h' )  /* also stop if it's a ``hear'' */
              break;

            MessageCounter++;
          }

          MessageCounter = 0;
```

Q. 8
I'm trying to run Ogalets or Sekine's clients, but players does not work well. Why?

A. 8
Unfortunately, Ogalets and Sekine's clients are for `sserver-2.78`. Protocol of sserver has been changed in `sserver-3.xx`, so now, these clients can not analyze sensor information sent from the server.

In order to test these clients, please get `sserver-2.78`.

C.3 Visual Information

Q. 9
Are the following two values the same?

$$DirChng = -1 \quad , \quad DistChng = -.334$$
$$DirChng = 179 \quad , \quad DistChng = .334$$

A. 9
No. If the client receives

$$(ObjName\ \ Dist\ \ Dir\ \ DistChng\ \ DirChng)$$

the actual relative movement (V_x, V_y) is:

$$\begin{aligned} V_x &= DistChng \cdot \cos(Dir) + (\pi/180) \cdot DirChng \cdot Dist \cdot \sin(Dir) \\ V_y &= DistChng \cdot \sin(Dir) + (\pi/180) \cdot DirChng \cdot Dist \cdot \cos(Dir) \end{aligned}$$

where V_x is a component that is parallel to the facing direciton of the player, and V_y is a component that is vertical to the facing direciton.

Q. 10
Does it always keep $|DirChng| < 90$?

A. 10
No. Theoretically, *DirChng* may take any real value.

Q. 11
If the player is stationary and the ball moves from relative position $(Dist, Ang)$ $(10, 15)$ to position $(10, 17)$ in the last simulator step (.1 second), then *DirChng* would be the angle of the ball motion relative to the player and *DistChng* would be the ABSOLUTE distance the ball traveled. Even though the ball's distance to the player remained constant, *DistChng* would not be 0.

A. 11
No. In this case, *istChng* should be 0, or near of 0. *DirChng* may be near of 2.0.

Q. 12
If the player moves, the vector of the players's ABSOLUTE motion is subtracted from and the ball's ABSOLUTE

motion vector. *DistChng* is the magnitude of the resulting vector and *DirChng* is its angle relative to the player's direction.

A. 12
No.

Suppose that player's motion and position are (V_x, V_y) and (P_x, P_y) respectively, and ball's motion and position are (U_x, U_y) and (Q_x, Q_y) respectively. Then, relative vector of motion (W_x, W_y) is

$$(W_x, W_y) = (U_x, U_y) - (V_x, V_y)$$

The relative position vector of ball (R_x, R_y) and its unit vector (E_x, E_y) are

$$\begin{aligned}(R_x, R_y) &= (Q_x, Q_y) - (P_x, P_y)\\ R &= \sqrt{R_x^2 + R_y^2}\\ (E_x, E_y) &= (R_x/R, R_y/R)\end{aligned}$$

Then, radius component ($D_r = DistChng$) and angle component ($D_a = DirChng$) of the relative movement (W_x, W_y) are

$$\begin{aligned}D_r &= (W_x \cdot E_x) + (W_y \cdot E_y)\\ D_a &= (180/\pi) \cdot (W_x \cdot E_y - W_y \cdot E_x)/R\end{aligned}$$

The inverse translation is as follows.
Suppose that D_r, D_a, R_x, R_y, V_x and V_y are given. Then,

$$\begin{aligned}R &= \sqrt{R_x^2 + R_y^2}\\ (E_x, E_y) &= (R_x/R, R_y/R)\\ W_x &= D_r \cdot E_x - (D_a \cdot \pi/180) \cdot R \cdot E_y\\ W_y &= D_r \cdot E_y + (D_a \cdot \pi/180) \cdot R \cdot E_x\\ (U_x, U_y) &= (V_x, V_y) + (W_x, W_y)\end{aligned}$$

These equations are theoretical and noise-less environment. Actually, the server adds various noise and quantizes values, so that the inverse translation is not so simple.

C.4 Inside of Soccer Server

Q. 13
The **Soccer Server** manual says that the magnitude of the noise in the ball motion varies in proportion to the parameter `ball_rand`.

Does the magnitude of the noise mean the standard deviation of the gaussian determining its next position? Or is it some other meaning?

A. 13
Generally **Soccer Server** use unformed distribution of noise. This means that, if `ball_rand` parameter (= magnitude of noise) is R, then the noise value is in $[-R, R]$, and the density of probability is the same in the domain$[-R, R]$. In other words, the probability that the noise is in $[x, x + dx]$ is,

$$\begin{aligned}P(\text{noise is in } [x, x+dx]) &= (1/2R) \cdot dx \text{ ; if } x \text{ is in } [-R, R]\\ &= 0 \text{ ; otherwise}\end{aligned}$$

Standard deviation σ of this distribution is:

$$\sigma^2 = \int_{-R}^{R} (1/2R) \cdot x^2 \cdot dx$$

$$
\begin{aligned}
&= (1/2R) * \left| \frac{x^3}{3} \right|_{-R}^{R} \\
&= \frac{R^2}{3}
\end{aligned}
$$

Q. 14
The scores on the `soccermonitor` wrap very soon, going into negative numbers if the score gets too big. This is a problem when using the same `soccerserver` for a large number of evaluations. Can I suggest using `int`s or `long`s for the scores instead of `char`s or `short`s?

A. 14
Internally, `soccerserver` uses `int` for score. So you need not care about it. The referee reports correct score when a team gets a goal.

We use `char` for communication between `soccerserver` and `soccermonitor` in order to reduce the amount of the communication.

Q. 15
What is the units of `player_size` and `ball_size`? Does these parameter specify radius? or diameter?

A. 15
The units is meters. They specify radius of the objects.

Q. 16
What is the units of `player_weight` and `ball_weight`? What is the units of `wind_force` and `ball_weight`?

A. 16
There are no correspondence between this unit and phisical unit systems. '`*_weight`' specifies the stiffness against wind effect. If weight is large, effect of force of the wind is small, and if weight is small, the effect is large. Actually, wind accelates the velocity of objects evety simulation cycle in the following manner:

$$
\begin{aligned}
\texttt{vel.x} &\mathrel{+}= \texttt{vel.r} \cdot \frac{\texttt{wind.x}}{\texttt{weight} \cdot \texttt{WIND_WEIDHT}} \\
\texttt{vel.y} &\mathrel{+}= \texttt{vel.r} \cdot \frac{\texttt{wind.y}}{\texttt{weight} \cdot \texttt{WIND_WEIDHT}}
\end{aligned}
$$

where (`vel.x`,`vel.y`) is object's velocity, `vel.r` is amount of movement of the object ($= |(\texttt{vel.x}, \texttt{vel.y})|$), (`wind.x`,`wind.y`) is a vector of strongness of the wind, and `WIND_WEIGHT` is a constant parameter ($= 10000$).

`wind_force` is amount of force of wind. This is equal to $|(\texttt{wind.x}, \texttt{wind.y})|$ (absolute value of the wind vector) in the above equation.

Q. 17
"`wind_dir` is 0" means that wind flows to north?

A. 17
No. Left-to-right on the display is 0 degree.

Q. 18
What is the units of `maxpower` and `minpower`?

A. 18
These do not correspond to the phisical parameters. It specifies the degree of powers. For example, if the client send `(dash 100)`, then the velocity of its player will change as follows:

$$\texttt{vel.x} += 100 * \texttt{dash_power_rate} * \cos(\texttt{dir})$$
$$\texttt{vel.y} += 100 * \texttt{dash_power_rate} * \sin(\texttt{dir})$$

where (`vel.x`,`vel.y`) is the velocity of the player, `dash_power_rate` is a parameter specified in file "`server.conf`", and `dir` is the direction the player is facing.

Q. 19
Is the units of `maxmoment` and `minmoment` degrees?

A. 19
Yes.