# Robocup

José M. Vidal

Tue Oct 14 11:32:18 EDT 2003

This talk summarizes material from

- Soccerserver Manual[1]
- Robocup Simulator[2]
- Stacy Marsella, Milind Tambe, Jafar Adibi, Yaser Al-Onaiza, Gal A. Kaminka, and Ion Muslea. Experiences Acquired in the Design of RoboCup Teams: A Comparison of Two Fielded Teams.[3] *Journal of Autonomous Agents and Multi-Agent Systems,* (4)1/2:115–129, 2001.

## 1 Robocup

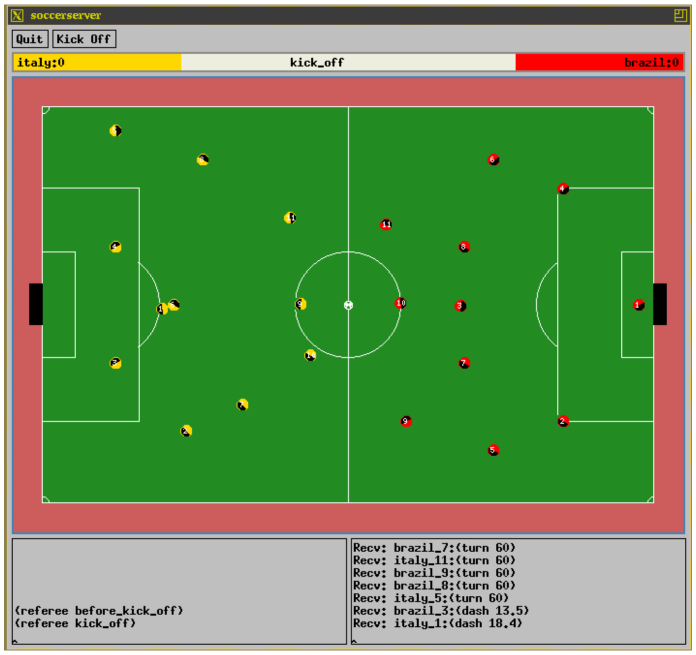
- An annual competition, held jointly with AI conferences.
- Small, medium, and large robotic leagues; legged league; simulation league; RoboCup rescue league.
- Goal: A robotic team that can beat worldcup champions.
- Participants are mostly graduate and undergraduate students. There are some "professional" teams (they lose). A junior league attracts high-school students.

[1] `http://sserver.sourceforge.net/downloads.html#manual`
[2] `http://sserver.sourceforge.net/`
[3] `http://jmvidal.cse.sc.edu/library/marsella-jaamas01.pdf`

## 2 Soccerserver

- The Soccerserver is used in the simulator league.
- The simulator implements a realistic world.
  - Actions do not always have their intended result.
  - The ball loses speed as it rolls.
  - There is noise in the inputs. The farther away something is the harder it is to see correctly, if at all. The farther away the ball is the harder it is to kick.
  - There is wind (although we will be turning it off).

– The players get tired. After a sprint, a player needs to recuperate before he can sprint again.

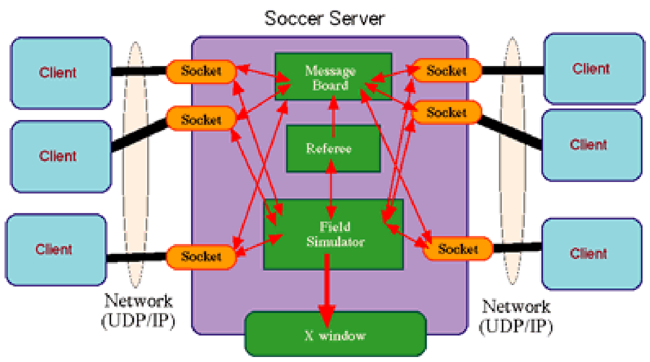


```
(referee before_kick_off)
(referee kick_off)
```

```
Recv: brazil_7:(turn 60)
Recv: italy_11:(turn 60)
Recv: brazil_9:(turn 60)
Recv: brazil_8:(turn 60)
Recv: italy_5:(turn 60)
Recv: brazil_3:(dash 13.5)
Recv: italy_1:(dash 18.4)
```

## 3 Architecture

- Uses UDP for communications. Messages can be lost (but they are almost never lost on a LAN).
- Every player connects independently to the sserver.
- The soccermonitor is the GUI. I also connects to the server using UDP.

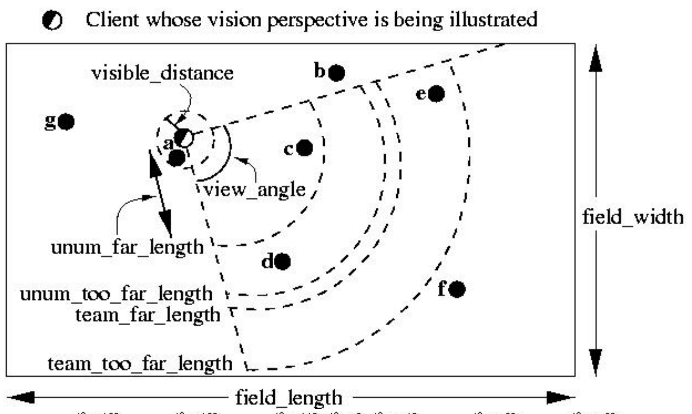


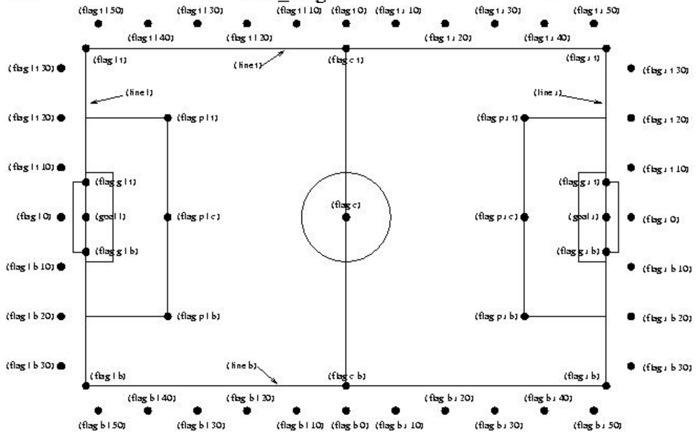
## 4 Physics

- The soccer field and all objects on it are 2-dimensional. There is no notion of height on any object.
- The players and the ball are treated as circles. All distances and angles used and reported are to the centers of the circles. The action model is discrete
- All objects move according to Newtonian physics. They have a decay. Move and Kick operations are implemented as vector additions to the current velocity vector.
- If at the end of the simulation cycle, two objects overlap, then the objects are moved back until they do not overlap.
- Noise is added to the movement of objects.

## 5 Player's Input

- The player receives information from the sserver which tells it:
  - What it `sees`, in polar coordinates relative to its current position and viewing direction.
  - This information includes the position of other players, the ball, goal posts, fixed flags, and lines.
  - The `sense-body` tells the player about its own body.





## 6 See

- Visual information arrives from the server in the following basic format: `(see Time ObjectInfo ObjectInfo ...)` where `ObjectInfo = (ObjName Distance Direction DistChng DirChng BodyDir HeadDir )` and

```
ObjName  ::=
(player Teamname UniformNumber)
      | (goal [l|r])
      | (ball)
      | (flag c)
      | (flag [l|c|r] [t|b])
      | (flag p [l|r] [t|c|b])
      | (flag g [l|r] [t|b])
      | (flag [l|r|t|b] 0)
      | (flag [t|b] [l|r] [10|20|30|40|50])
      | (flag [l|r] [t|b] [10|20|30])
      | (line [l|r|t|b])
```

## 7 Player Actions

- `(turn Moment)` Change the direction of the player according to `Moment`. `Moment` should be between `minmoment` and `maxmoment` (default is [-180, 180]). The actual change of direction is reduced when the player is moving quickly.
- `(turn-neck Angle)` Adds `Angle` to the clients neck angle (ie angle at which the view cone extends from the player) The neck angle must be between [-90,90]. The neck angle is always relative to the angle of the player, so if a `turn` command is issued, the view angle also changes.
- `(dash Power)` Increases the velocity of the player in the direction it is facing by `Power`*`dash-power-rate`. `Power` should be between `minpower` and `maxpower` (default: [-30, 100]). If power is negative, then the player is effectively dashing backwards.
- `(kick Power Direction)` Kick the ball with `Power` in `Direction` if the ball is near enough (the distance to the ball is less than `kickable-margin` + `ball-size` + `player-size`.). `Power` should be between `minpower` and `maxpower` (default is [-30,100). `Direction` should be between `minmoment` and `maxmoment` (default is [-180,180]).
- `(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, and for the goalie immediately after catching the ball (see the `catch`) command.
- `(catch Direction)` Tries to catch the ball in direction `Direction`. `Direction` should be in [-180, 180]. This command is permitted only for goalie clients. The player (goalie) can catch the ball when the ball is in the rectangle with width is `goalie-catchable-area-w` (default=1), length is `goalie-catchable-area-l` (default=2) and the direction is `Direction`. The probability the catch is successful if it is in the rectangle is given by the parameter `catch-probability`. Also note that the goalie can only do 1 catch every few cycles (specified by the `catch-ban-cycle` parameter). If the goalie tries to catch again during the ban cycle, the command is ignored. If the catch is successful, the server goes into free kick mode. Once it has caught the ball, the goalie can move within the penalty box with the `move` command. The ball moves with the agent. However, a catch does not immediately change the position or facing direction of the goalie.
- `(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 with length less than 512

characters, and consists of alphanumeric characters and the symbols "—+-*/-.() —". There is a maximum distance that messages can be heard. See the section on auditory information for specifics.

- `(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, no distance) 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 150 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 37.5 milli-sec.

- Discrete actions are received at any time. Every `simulator-step` (100ms) the simulator updates the world model using these inputs. Only one action per agent is accepted.

## 8 Sense Body

- In previous versions of the server, there was a `sense-body` command which returned some information about the state of the player. This command no longer exists in versions greater than 5.00. Instead, the following information is sent automatically to every client every `sense-body-step` milliseconds.

- 
```
(sense-body  TIME
  (view-mode QUALITY WIDTH)
  (stamina STAMINA EFFORT)
  (speed AMOUNT-OF-SPEED)
  (head-angle RELATIVE-HEAD-ANGLE)
  (kick KICK-COUNT)
  (dash DASH-COUNT)
  (turn TURN-COUNT)
  (say SAY-COUNT)
  (turn-neck TURN-NECK-COUNT))
```

## 9 Stamina

- Each player has its own stamina. There are three relevant parameters:
  - `stamina`: used up when dashing and replenished slightly. Stamina increases slightly every cycle. As `recovery` decreases, less stamina is recovered.
  - `effort`: determines how effective dashing is. The basic idea is that if `stamina` gets low, `effort` decreases (with a minimum value given by `effort-min`) and if `stamina` gets high enough, then `effort` increases with a maximum of 1.0.
  - `recovery`: controls how much stamina is recovered each cycle. This is similar to effort except that `recovery` never increases.
- The player can `dash` only with `Power` lower than the current stamina. The stamina decreases by the `Power`.

## 10 Tips

- Start with reliable low-level abilities: dribbling, passing (throwing and catching), shooting, guarding.
- Passing is key. The ball moves a lot faster than an agent can.
- Some of the most common lessons learned last year.
  1. *Testing with another team will immediately reveal many bugs in the program.* Many waited until the tournament to do this—bad idea.
  2. *Changing one line of code can change the emergent behavior of the whole team in unexpected way.* This is what makes multiagent system such fun (and challenging).
  3. *Placing a lot of `System.out.println()` in programs will slow it down a lot.* Of course.
  4. *The Java Virtual Machine's thread scheduler is not fair or round-robin* . I don't know why you would assume it was.
  5. *Always looking towards the ball to see where it is can make a player very slow moving* . Imagine that!
- Always think in terms of the team, even thought you are only programming one agent.
- Get started NOW.
- Code, test team, repeat.

## 11 Lessons from ISIS

- ISIS is USC's (west) team. The group has been led by Milind Tambe.
- There is a tradeoff in monitoring. You can either build a complex agent that keeps track of everything around it (control tower), or simpler agents that communicate the important information to each other (lookout person). The case with simpler agents relies on *joint commitments* .
- *Competition with collaboration* is good. Having an overlap in the players' zones allows for failure recovery (e.g., one player is tired, or can't see the ball, or is blocked by others....).
- Used C4.5 to learn what part of the goal to shoot for. Some decisions are too hard to do by hand. However, learning also fails because it often assumes a perfect opponent (worst-case scenario).'
- Social online learning must be undertaken with caution. That is, what one player learns might not be useful to others (because they face different challenges).