The Global Architecture for the RoboCup Mixed Reality Sub-League proposed by the UFRN-POTI Team - version 2008

Adelardo A.D. Medeiros, Pablo J. Alsina & João Paulo F. Guimarães
Departement of Computing Engineering and Automation - DCA
Federal University of Rio Grande do Norte - UFRN
59078-900 – Natal RN – Brazil

***Abstract*— In this paper we present the concept of a global system to dispute the Mixed Reality Sub-League of RoboCup. The concept includes the specification of a proposed general system architecture, composed of software modules and their communication interfaces. We also write about some of these software modules and present the state of their development.**

I. Global architecture

In the Mixed Reality sub-league, the field (a horizontally mounted monitor) is overseen by a camera connected to a global vision system. Both the robots and entities projected on the screen (e.g. the soccer field) are seen and tracked by the vision system. Depending on the application, some elements may be virtual or physical in nature (e.g. a soccer ball, obstacles). The central server provides an interface relaying these perceptions to client programs developed by the teams, which in turn send commands to the central server. The central server forwards the commands to the robots themselves using infrared communication.

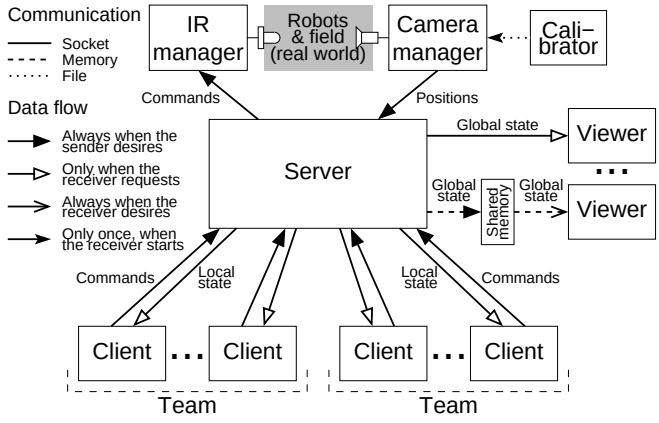

Fig. 1. Global architecture

We propose a global architecture for the Mixed Reality system represented in Figure 1. The software system is composed of 7 modules; each one can be executed at a different machine or at the same one. Communication between modules is described later (Section I-B). The modules are:

- **Client:** each instance of the Client module independently controls one robot of a team. Clients receive information from the Server module about the positions of the objects on the field. According to the rules of the specif application, information can be complete (global state) or partial (local state - for instance, when a player only sees objects into a certain field of view). Clients send the reference velocities of the robots (logical commands) to the Server module.
- **Viewer:** graphically represents the current state of the game. Several Viewers can be simultaneously running. Normally, at least one instance must be running and displayed on the screen on which the robots are playing. Viewers receive the global state of the game from the Server module.
- **IR Manager:** receives logical commands (reference velocities) from the Server module and sends them to the robots (physical commands).
- **Camera Manager:** processes images captured by the camera to extract positions of objects. These positions are converted to appropriate units (see Section I-A) and sent to the Server module.
- **Server:** this central software routes data from one module to another one and simulates the behaviour of the virtual elements of the game.
- **Calibrator:** before the game starts, it calculates the calibration parameters of the system and informs them to the Camera Manager module. These parameters model the characteristics of the camera (resolution, distortion, position) and of the field (size, position, luminosity, etc.).
- **Simulator:** this module, not shown in Figure 1, replaces the real world when we cannot or prefer not to play with the actual robots. In this situation, the Simulator module communicates with the Server module to replace both the IR Manager and the Camera Manager modules.

A. Position units

Some of the difficulties to use and to adapt the current version of the Mixed Reality server are caused by the choice of using a "physical" unit (pixels) to represent the position of the objects at the field. When the screen size or the camera resolution changes, all modules have to be adapted.

We propose to adopt a "virtual" unit (VU) to represent

the positions. The field size is fixed in VUs in the competition rules (for instance, 32000 × 18000 VU). The Camera Manager is the only module that knows about physical units and converts them to VUs, using information provided by the Calibrator module.

The actual size and velocities of the robots are not constant in VUs. The Calibrator module, besides calculating the equivalence between physical and virtual units, also converts the physical size of robots to VUs and informs the Server module of the virtual size. This piece of information is propagated to Clients, so they can adopt a control strategy based on the actual velocity capabilities of the robots, and to Viewers, to represent robots with correct dimensions.

B. Inter-module Communication

We propose adopting sockets as the communication medium between all modules, with two exceptions described later. Adopting sockets has some advantages:

- the modules can be executed in the same or in different machines;
- the modules are independent, in such a way they can be developed with different programming languages and executed in different operating systems.

We suggest not adopting sockets in two cases:

- To transfer information from the Calibrator to the Camera Manager modules. Communication between them only occurs when the Camera Manager module starts; in this case, it seems to be more appropriate to use a specific file, written by the Calibrator module and subsequently read by the Camera Manager module. Using the file, it is not necessary to recalibrate the system every time, if the conditions have not changed.
- In some situations, the refresh rate of the Viewer module can be high and data flow through the socket linking it to the Server module can considerable, possibly introducing a bottleneck. To prevent this situation, we advocate the existence of two communication ways between these two modules, so that both of them can or cannot be simultaneously used:
  - the usual communication by socket;
  - a shared memory, written by the Server module when it has new data and read by the Viewer module when it needs.

II. Software development

A. Calibrator

We implemented a Calibrator program for Linux that uses the Qt [1] library. Figure 2 showns two screens of the program.

To begin the calibration process, the user captures an image of the monitor with the robots and an instance of the Viewer being displayed on it. Then, several lines must be positioned with the mouse to delimit the borders of the game field. Using the virtual size of the field (32000 × 18000 VU) and the lines' intersection points, the program calculates the camera parameters (extrinsic and intrinsic parameters and radial distortion) [2] and the conversion from pixels to VUs [3].

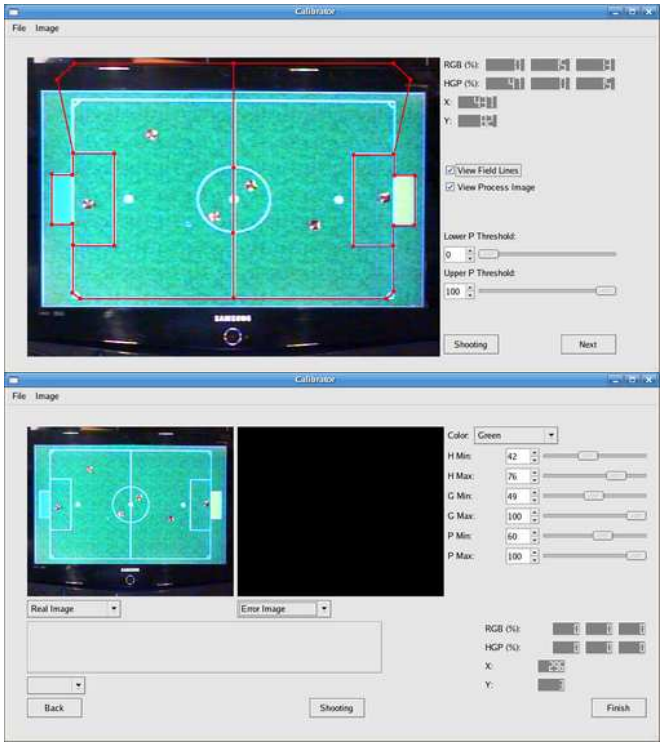

Fig. 2. The Calibrator program: calibrating field borders (up) and robot labels (down)

A second step concerns calibrating the recognition of labels, the marks on top of the robots. Without details, the idea is to use a color space suited to this purpose [4] where hue, saturation and luminance are calculated for each pixel. Intervals of these color parameters are adjusted by the user to correctly segment the image, separating the pixels belonging to robots from the other ones.

B. Camera Manager

The Camera Manager module processes images to obtain positions of objects in the field (robots). The program starts by reading information generated by the Calibrator module. After that, it continuously waits for the last image captured by the camera and process it to localize robots. These positions are sent to the Server module as soon as they are available.

To process an image, the program only searchs the field region, delimited by the Calibrator program. The search looks for pixels belonging to robots, using the previously calibrated intervals of color parameters. When such a pixel is found, a filling algorith is used to find all neighbour pixels belonging to the same robot. Knowing the form and color of the labels, this set of pixels allows calculating the position and orientation of the robot.

C. Viewer

We have developed a OpenGL-based [5] Viewer program for Linux. At the present time, only the shared-memory communication mechanism (see Section I-B) is available; sockets will be included soon.

Figure 3 shows a snapshot of the program. The drawn field maintains its width/height proportion always equal to the width/height proportion of the virtual field (see Section I-A).

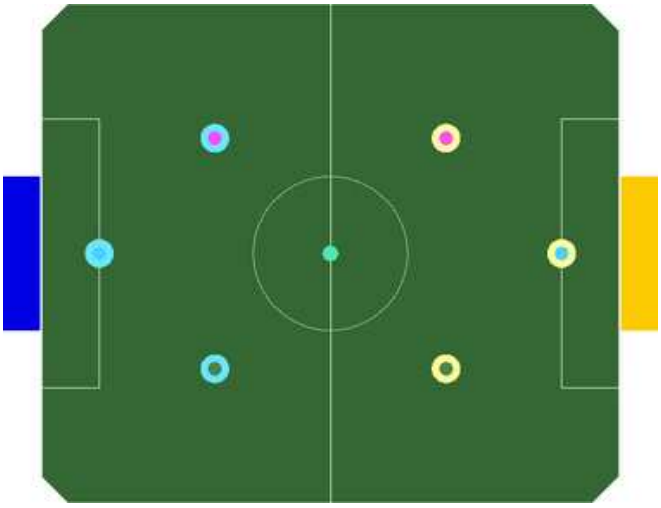

Fig. 3. The Viewer program

D. IR Manager

Our IR Manager uses the USB port to send data to the infrared transmitter [7]. It is composed of a single thread that continously waits for data in the socket, converts the received logical command into a physical one and sends data via USB to the appropriate robot.

E. Server

The Server module was implemented using several threads:

- Two threads manage communication with Client modules (one for each team). When data arrives, there are two possibilities:
  - The Client has sent a command. In this case, the local robot ID used by the Client is converted to the corresponding global (physical) one and the command is sent to the IR Manager module.
  - The Client has requested the game state. To fulfil the request, the last calculated global state is eventually manipulated to eliminate or to transform information that should not be included in the local state and the result is sent to the corresponding Client module.
- One thread carries out requests from Viewer modules when they arrive by socket, sending them the current global state of the game.
- One thread continuously waits for data from the Camera Manager module to export it to a shared memory, where the available most recent position of robots can be read by other threads.
- One thread periodically simulates and updates the position of all virtual objects in the field.

The program takes into account all collisions between virtual objects and other (virtual and real) entities to simulate the virtual objects. One funcionality not yet implemented but planned is to filter and to generate commands to simulate effects of virtual entities on the robots' behaviour: for instance, not allowing the real robots to go out of the field.

F. Simulator

We had already developed a dynamic simulator for another type of mini-robots [6]. This program was adapted to simulate the Eco-Be! robots and to replace the modules IR Manager and Camera Manager when the actual robots, infra-red transmitter and camera are not used.

References

[1] Trolltech. Qt documentation. `http://doc.trolltech.com/`. Accessed 01/2008.

[2] Forsyth, David A.; Ponce, Jean. Computer Vision: A Modern Approach. Prentice Hall, 2002.

[3] Aires, Kelson R.T.; Alsina, Pablo J.; Medeiros, Adelardo A.D. A Global Vision System for Mobile Mini-Robots. SBAI 2001 - Simpósio Brasileiro de Automação Inteligente. Gramado, RS, Brazil, 11/2001.

[4] Medeiros, Adelardo A.D.; Mendes, Ellon P. The HGP (Hue-Grayness-Purity) Color Space to Segment Colored Structured Environments. To be published.

[5] OpenGL Working Group.OpenGL documentation. `http://www.opengl.org/`. Accessed 01/2008.

[6] Yamamoto, Marcelo M.; Pedrosa, Diogo P.F.; Medeiros, Adelardo A.D.; Alsina, Pablo J. Um Simulador Dinâmico para Mini-Robôs Móveis com Modelagem de Colisões (A Dynamic Simulator for Mobile Mini-Robots with Modelling of Collisions). SBAI 2003 - Simpósio Brasileiro de Automação Inteligente, pp. 852-857. Bauru, SP, Brazil, 09/2003.

[7] IRTrans. IRTrans USB documentation. `http://www.irtrans.de/en/`. Accessed 01/2008. '