RoboCup Advanced 3D Monitor

Carla Penedo, João Pavão, Pedro Nunes, and Luis Custódio

Instituto de Sistemas e Robótica
Instituto Superior Técnico
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
{ccfp,jpp,pmgn}@rnl.ist.utl.pt, lmmc@isr.ist.utl.pt

Abstract. RoboCup Advanced 3D Monitor is a three-dimensional application for visualizing and debugging games of the RoboCup Soccer Simulation League. This paper discusses the issues pertaining the implementation of this monitor using OpenGL, a standard API for rendering high-performance 3D graphics. Our application provides a true 3D soccer game experience maintaining a healthy balance of realistic animation features and high-speed rendering achieved by the implementation of specific computer graphics techniques. Besides its main usefulness as a visualization tool, this monitor may be used as a supporting tool for the development of other robotics techniques. To illustrate this, two of such techniques are discussed here: sensor fusion and Markov localization methods.

Keywords: three-dimensional monitor, simulation league, levels of detail, markov localization, sensor fusion.

1 Introduction

The RoboCup *soccer server* simulates a 2D virtual field in which two agent teams play a soccer match. Although the environment is two-dimensional, the *soccer server* has proved to be an adequate platform for the development of realistic 3D visualization tools.

In this paper, the RoboCup Advanced 3D Monitor (RA3DM), a three-dimensional monitor for the RoboCup Simulation League is introduced. RA3DM aims to turn the visualization of simulated soccer games more realistic and entertaining. As we were developing RA3DM we realized that the RoboCup *soccer server* provides an interesting testbed for prototyping algorithms which could be used afterwards on real robots. A Markov localization algorithm and a sensor fusion method were successfully implemented in this environment and the latter has already been adapted to real robots of the middle-size league. Our monitor was extremely useful for developing/testing/debugging the implemented algorithms.

RA3DM is implemented on OpenGL, a low-level graphics library which is designed to be used with C and C++ programming languages. One of the main advantages that OpenGL presents is its independence from operating and windowing systems. This feature enabled the development of versions for the Linux, MacOS X and Windows platforms with only a little extra effort.

D. Polani et al. (Eds.): RoboCup 2003, LNAI 3020, pp. 637–644, 2004.

The remainder of this paper is organized as follows. In the next section the global architecture of the distributed system is briefly described. In Section 3 we present the main features of the implementation of RA3DM. Section 4 explains how RA3DM was used for the application of sensor fusion and Markov localization algorithms in the RoboCup Simulation League. Finally, in Section 5 conclusions are drawn and future work is discussed.

2 System Overview

The RoboCup simulation system is controlled by a central process — the *soccer server* — that runs the whole simulation, sending and receiving all the relevant information to and from all the client programs connected to it [1]. An extra connection was added to enable us to visualize the internal state of a soccer player. The RA3DM connects to a special port on the *soccer player* program to be monitored via a dedicated UDP socket. The global architecture employed in our implementation is illustrated in Fig. 1.

The *soccer server* program sends information to all its connected monitors every 0.1 s. However, the majority of hardware available today (both graphic cards and main processors) is capable of delivering full-screen image update rates (or frame rates) well above 10 Hz (i.e., 10 frames per second). This fact led us to an architecture that maximizes drawing performance by allocating the maximum possible CPU time updating the scene. To accomplish this we use two permanently running threads [2]. One of the threads is responsible for all transactions with the *soccer server* such as sending commands issued by the user and receiving data concerning the state of the simulation. The other thread will update the displayed 3D scene as fast as the underlying hardware allows. This thread is also responsible for managing all user interaction through the use of menus, mouse and keyboard.

3 Implementation

3.1 Player

RA3DM uses a skeletal animation system which allows for three main animations to be performed by the player: walk, run and kick (see Fig. 2). A skeleton is defined by a hierarchy of bones connected by joints with a position and an orientation, arranged in a tree structure. The model, usually a single mesh of

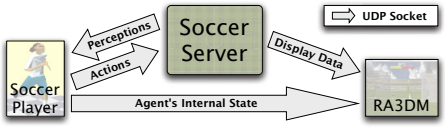

Fig. 1. Global architecture of the distributed system.

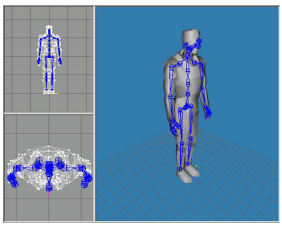

Fig. 2. Player model and its bone structure.

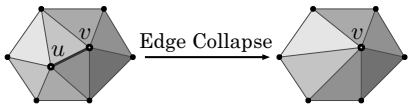

Fig. 3. Collapsing vertex u onto v.

vertices or polygons, is attached to the skeleton through a process known as *skinning*. When the latter is animated, the mesh is deformed accordingly, and therefore, the model is continually morphed to match the movements of the skeleton.

The animation process is simplified by keyframe interpolation techniques. A keyframe animation is created by interpolating frames (snapshots in time) between several successive keyframes that define specific pivot points in the action. Each keyframe contains the values of the rotations of each joint of the skeleton. The mesh is stored only once, along with the skeleton and the keyframes of the animation, resulting in a very compact representation. Each animation is created according to an uniform and constant speed that can be modulated directly in RA3DM to obtain different running or walking velocities.

3.2 Levels of Detail

The computation and storage requirements for complex scenes, typical in some computer graphics applications, far exceeds the capacity of modern hardware. In order to accelerate the rendering of such scenes, approximations of decreasing complexity (levels of detail) are produced similarly to the initial model. The simpler versions contain fewer details that can not be noticed when the object is farther away [3]. Most of the best techniques of polygonal simplification are based on Hughes Hoppe's progressive mesh work [4]. One possible simplification is achieved by continuously applying an *edge collapse* operator that merges two edge vertices into one, thus removing that edge.

Vertex Remove. This operation takes two vertices u and v (the edge $\overline{uv}$) and "moves" or "collapses" one of them onto the other [5]. Figure 3 illustrates a polygon before and after the application of the edge collapse operator. The following steps explain how this operation is implemented:

1. Remove any triangles that have both u and v as vertices (i.e., remove triangles on the edge $\overline{uv}$).
2. Update the remaining triangles that use u as a vertex to use v instead.
3. Remove vertex u.

The removal process is repeated until the target polygon count is reached.

Selection of the Next Vertex to Collapse. When selecting a vertex to collapse, the basic idea is to preserve (as far as possible) the global appearance of an object, trying to cause the smallest visual change to it. Despite the considerable number of algorithms that determine the "minimal cost" vertex to collapse at each step, they are, in general, too elaborate to implement. A simple approach for this selection process may be to just consider the cost of collapsing an edge defined as its length multiplied by a curvature term (the latter having half the weight of the former in this calculation). The curvature term for collapsing an edge $\overline{uv}$ is therefore determined by comparing dot products of face normals in order to find the triangle adjacent to u that faces furthest away from the other triangles that are along $\overline{uv}$. Equation (1) expresses this idea, where T_u is the set of triangles that contain u, $T_{\overline{uv}}$ refers the set of triangles that contain both u and v, and f_N and n_N are the face normals of triangles f and n.

$$\text{cost}(u, v) = \|u - v\| \times \max_{f \in T_u} \left\{ \min_{n \in T_{\overline{uv}}} \left\{ \frac{1 - f_{\text{N}} \cdot n_{\text{N}}}{2} \right\} \right\} \tag{1}$$

The algorithm described throughout this section can be summed up as follows: while the current polygon count is greater than the desired target number, select a candidate edge to collapse (according to its associated cost) and apply the edge collapse operator to it.

Player Simplification. In practice, the polygon simplification algorithm produces very reasonable results. Our soccer player originally contained 347 vertices and 639 triangles which, after the application of this algorithm, were reduced to 68 vertices and 130 triangles. These two versions correspond to the highest and the lowest level of detail used in RA3DM. The former is used when the camera is closer to the player while the other is selected if it is far away. In between we consider intermediate levels obtained by the continuous variation of the total number of vertices used to draw the player. In this way, it is possible to avoid an undesired effect, named *popping*, that can be seen as an abrupt change in the detail present in the object's shape caused by the considerable difference in the number of vertices.

3.3 Cameras

There are three distinct types of views into the field available to the end user: static, user controlled and automatic cameras. A view similar to the display of the official 2D soccer monitor (i.e., aerial view, pointing towards the center of the soccer field) is available, as well as a completely custom camera that the user controls using the mouse and the keyboard. Next we describe two of the most interesting automatic cameras:

TV camera has a fixed position slightly raised on one side of the field and constantly follows the ball with a small delay delivering image sequences similar to those captured by a human camera operator. This camera zooms

in and out as the subject being shot approaches and moves away from the camera;

Event-driven cinematic camera consists of a series of cameras placed in some strategic points of the 3D scene. Some of these cameras have a fixed position whereas others are capable of some dynamic motion. At any given point in time during the simulation of a soccer match, each one of these predefined cameras will be assigned a subjective value representing the quality and usefulness of the image sequence provided by it. The measurement of this value and its assignment to the cameras is taken care of by a *director* agent through the use of some heuristics. This agent also decides how to point the cameras to the scene and how much zoom to use. After this evaluation, an *editor* agent in charge of producing the sequence of images seen on screen will decide (based on the values produced by the *director*) which camera to choose and for how long. Different *director* and *editor* agents can be defined. By changing the way the *director* rates the footage and the way the *editor* chooses among them will effectively change the resulting cinematic style, thus giving a totally different experience to the user [6]. For example, if two players are fighting for the ball in the midfield, the director may choose to zoom in one of the closest cameras to make a close-up and give that camera a high priority value. The editor will then be able to use that footage, always taking into account the minimum and maximum time intervals allowed between cuts.

4 Applications

In this section we describe the implementation of sensor fusion and Markov localization algorithms in the framework of the RoboCup Simulation League. Our 3D monitor was a valuable resource allowing the visualization of the player's internal state and the gathering of experimental results. RA3DM includes a low quality viewing mode that was extensively used in order to facilitate the data observations.

4.1 Sensor Fusion

Sharing information among robots increases the effective instantaneous perception of the environment, allowing accurate modeling. Two known sensor fusion approaches were tested in the RoboCup Simulation League: the Stroupe and the Durrant-Whyte methods [7].

To perform sensor fusion the soccer server's noise model is approximated by a two-dimensional Gaussian distribution $N(\mu, C_L)$, where μ is a vector representing the calculated position of the object and C_L is a diagonal matrix that denotes the variance along both axis (see Fig. 4). The variance along the axis that points from the robot towards the observed object (σ_{maj}) is calculated based on the quantization made by the *soccer server*. Thus, σ_{min} represents the variance along the perpendicular axis and is based on the maximum error in angle that an observation can have.

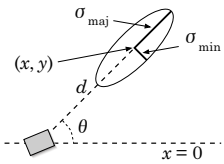

Fig. 4. Distribution parameter definitions: mean (x, y), angle of major axis (θ), major and minor standard deviations $(\sigma_{maj}, \sigma_{min})$ and distance to mean (d).

In order to use sensor fusion we must exchange sensor information between team members. This exchange provides a basis through which individual sensors can cooperate, resolve conflicts or disagreements, or complement each other's view of the environment. In this case, an agent communicates information (μ, C_L and θ) about seen objects to other agents.

Our goal was to compare the efficiency of the two mentioned sensor fusion methods. The first approach simply merges the Gaussian distributions of the observations made by the robot with the ones made by the others. The second method takes into account the last known position of the object and tests if the readings obtained from several sensors are close enough to make the fusion. When this test fails, no fusion is made and the sensor reading with less variance is chosen. The conditions in which this test fails and succeeds are presented in Fig. 5.

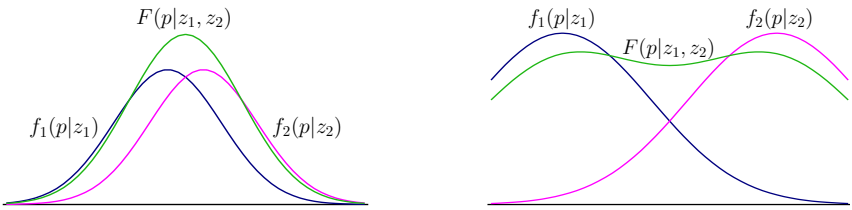

Fig. 5. Two Bayesian observers with joint posterior likelihood indicating agreement and disagreement.

The ball in Fig. 7 (a) with the black and white texture represents the real ball position communicated by the *soccer server*. The farthest ball from the real position represents the position where the goalkeeper, on the other side of the field, sees the ball. The position communicated by a team member to the goalkeeper is represented by the other dark ball. Finally, the white ball represents the fusion between heard and seen ball obtained with the Stroupe method. As can be noticed, the newly calculated position represents a better estimate than the observed value.

4.2 Markov Localization

For mobile robots, localization is the process of updating the pose of a robot, given information about its environment and the history of its sensor readings.

An implementation of the Markov localization method was applied to self-localize a player in the RoboCup Simulation League [8].

At any point in time, Markov localization maintains a position probability density (belief) over the entire configuration space of the robot based on an incoming stream of sensor data (observations) and an outcome of actions. This probability framework employs multi-modal distributions for the robot belief enabling the representation of ambiguous situations by considering multiple hypotheses in parallel.

This particular implementation of Markov localization uses a fine-grained geometric discretization to represent the position of the robot. A position probability grid is three-dimensional, as each possible location l is defined by a tuple $\langle x, y, \theta \rangle$ representing the robot position and orientation. The principle of the position probability grid approach ascribes to each cell of the grid the probability of the robot being located in that cell, denoted by $Bel(L_t = l)$. Figure 6 illustrates the structure of a position probability grid and the correspondence between grid and field positions in the RoboCup simulation system. Each layer of the grid assigns all possible poses of the robot with the same orientation.

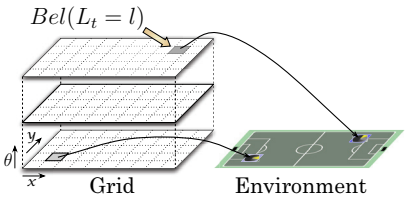

Fig. 6. Transformation of grid coordinates into field coordinates.

When used as a debugger, RA3DM enables us to inspect the internal state of a player (more precisely, its position probability grid) in real time. Figure 7 (b) shows a screen capture of RA3DM while it was monitoring the goalkeeper player. The cells with a belief value above a certain threshold are drawn in a shade of blue and the one with the highest belief at each time step is distinguished with a red color.

Experimental results show that, generally, the Markov localization method keeps the robot position error bound within very reasonable limits. For example, we obtained a medium error of 2.44 m for a grid of cells of 2.63×1.70 meters.

5 Conclusions and Future Work

Watching simulated soccer games on a realistic 3D monitor such as RA3DM is far more motivating than on the traditional 2D monitor. We showed that the RA3DM is a powerful tool that can be used as an aid for testing and debugging of prototype applications that may later be employed on real robots. Another possible application that we are considering consists of using RA3DM to simulate

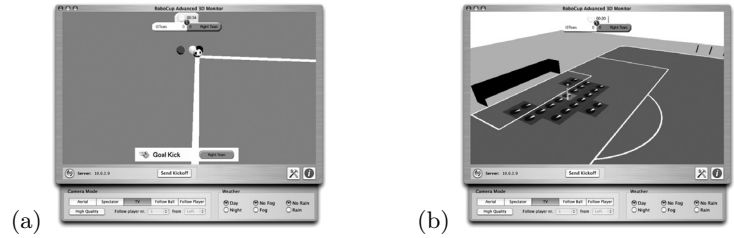

Fig. 7. (a) Ball localization using Stroupe's sensor fusion method. (b) Goalkeeper self localization using Markov estimation.

humanoid stereo vision. This way, the soccer playing agents could receive images supplied by RA3DM instead of the visual perceptions in the form of strings provided by the *soccer server*.

There are some features we would like to add in the future, such as sound effects, the recording of parts of a game to instant replay them later on and the addition of a commentary system.

References

1. Noda, I., *et al.*: RoboCup Soccer Server: Users Manual for Soccer Server Version 7.07 and later. (2001)
2. Barney, B.M.: POSIX threads programming (2001) Lawrence Livermore National Laboratory.
3. Krus, M., Bourdot, P., Guisnel, F., Thibault, G.: Levels of detail & polygonal simplification. ACM Crossroads **3.4** (1997) 13–19
4. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Mesh optimization. Computer Graphics **27** (1993) 19–26
5. Melax, S.: A simple, fast, and effective polygon reduction algorithm. Game Developer Magazine (1998) 44–49
6. Hawkins, B.: Creating an event-driven cinematic camera. Game Developer (2002)
7. Nunes, P., Marcelino, P., Lima, P., Ribeiro, M.I.: Improving object localization through sensor fusion applied to soccer robots. In: Proc. of the Robótica 2003 Portuguese Scientific Meeting, Lisbon (2003) 51–58
8. Penedo, C., Pavão, J., Lima, P., Ribeiro, M.I.: Markov localization in the RoboCup Simulation League. In: Proc. of the Robótica 2003 Portuguese Scientific Meeting, Lisbon (2003) 13–20