HYPERPRESENCE – AN APPLICATION ENVIRONMENT FOR CONTROL OF MULTI-USER AGENTS IN MIXED REALITY SPACES

Douglas Tavares[2], Aquiles Burlamaqui[1] , Anfranserai Dias[2], Meika Monteiro[2], Viviane Antunes[2], George Thó[2] , Tatiana Tavares[1], Carlos Lima[3], Luiz Gonçalves[2], Guido Lemos[1], Pablo Alsina[2], Adelardo Medeiros[2]

[1] NatalNet Laboratory, [2] Robotics Laboratory, [3] Virtual Reality Laboratory

Federal University of Rio Grande do Norte, Campus Universitário Lagoa Nova, 59076-970, Natal. RN, Brasil

{tati, aquiles, guido}@natalnet.br, {douglas, xamd, lmarcos, meika, tho,adelardo,pablo,magno}@dca.ufrn.br

Abstract

The HYPERPRESENCE system proposed in this work is a mix between hardware and software platforms developed for control of multi-user agents in a mixed reality environment. The hardware basically composed by robot systems that manipulate objects and move in a closed, real environment and a video camera, imaging system. The environment can be any place that provides or needs interactions with virtual environments for showing results or else to allow manipulation on it via a virtual reality interface.The software part is composed by three main systems: an acquiring module for position control, a tower (hardware) communication module for manipulating the robots and a multi user VRML server for managing virtual spaces and to provide synchronism between the real places and the virtual ones. In this article we present the design and implementation solution adopted for the HYPERPRESENCE architecture. We analyze problems with communication protocols, precision on positioning, how to relate physical and virtual objetcs and show our solutions for these problems.

1. INTRODUCTION

At start user interfaces were based in commands. Later the GUIs (Graphical User Interface) brought color and forms to computer systems by using images for user interface implementation. So, the WYSIWYG[1] approach was real to users when they could delete and archive by putting it on the garbage. Today it is possible to extend the effects WYSIWYG proposal by using Virtual Reality resources. VR interfaces provide to users other ways to interact with the application. Immersed Virtual Reality – IVR [01] is a high-end user interface that involves the user into a 3D model application so that human users will feel immersed in a virtual world. Also, some VR techniques bring real worlds to users as the Mars exploration by using a UV (Umnanned Vehicle) called the Mars Exploration Rover [02].

Also, we can use robot devices integrated with VRML worlds. It can offer to users a real, efficient and secure interaction mode with real places. This feature can be especially useful in hostile environments where just machines could be safely, as nuclear power stations, submarine exploration, and others. 3D interfaces can help users during simulations that use tangible devices between the real and virtual models increasing the realism of them. Clearly, we have many solutions for implementating this kind of system. Our work differs from others in the same subject cited in the following by introducing the mixed reality ideas essentially to quickly provide feedback on seamless applications among virtual and real environments.

2. RELATED WORKS

According to the CSCW aproach Moran and Anderson [03] discuss three fundamental ideas: Shared Workspaces, Coordinated Communication and Informal Interaction. The Shared Spaces can represent CSCW applications in terms of spatiality, transportation and artificiality [04, 11]. Unlike usual applications in Virtual Reality that replaces the physical world, augmented reality explores seamlessly applications or applications that mix virtual and real features. Ressier [06] presents a environment for collaborative engineering. This environment integrates a multiuser synthetic environment with physical robotic devices. The goal is to provide a physical representation of the 3D environment enabling the user to perform direct, tangible manipulations of the devices. The implementation used Lego robots, VRML anda Java technologies. This application also uses a vision processing system for computing LEGO orientation and position in the real environment.

Using another approach, the Augmented Reality (AR) paradigm [03] enhances physical reality by integrating virtual objects into the physical world, which becomes in a sense an equal part of our natural environment. The

[1] "What you see is what you get?"

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

Collaborative AR Games (Figure 1) project is a good example of these. The work has been designed for demonstrating the use of AR technologies in a collaborative entertainment application. This is a multi player game simulation where the users play with cards. The goal of the game is to match object that logically belong together, such as an alien and a flying saucer. When cards containing logical matches were placed side by side, an animation is triggered involving the objects on the card.

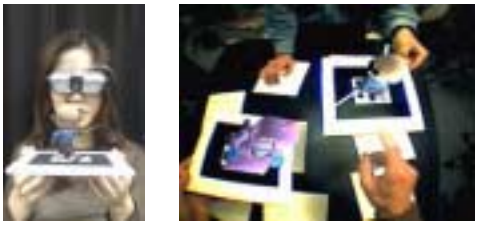


Figure 1 –Collaborative AR Games Project.

The **Augmented Groove** is a musical interface that explores use of augmented reality, three-dimensional (3D) interfaces and physical, tangible interaction for conducting multimedia musical performance. The goal is to improve for visitors a physical world with virtual objects where several visitors can easily join around the mixing table and "jam" together, passing musical sequences to each other in the same manner in which we would pass everyday objects. The music in a sense becomes a physical, tangible object, something we can touch and see as part of our physical environment. Besides, musical control is deliberately imprecise because the performer manipulates short musical sequences, or phrases, rather then individual notes.

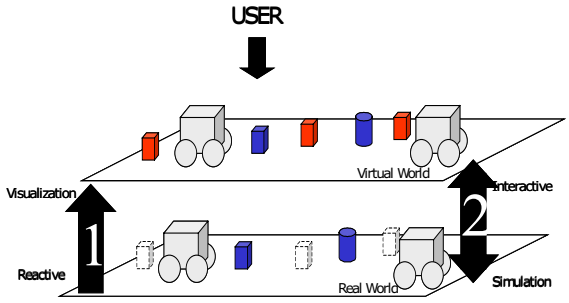


Figure 2 – HYPERPRESENCE system overview.

3. THE PROPOSED HIPERPRESENCE SYSTEM

Figure 2 illustrates the conceptual model of the HYPERPRESENCE system, proposed in this work. It follows the basic idea of having the real and virtual worlds representing the same situation. In phase 1 (arrow 1 in the left of figure) we have a unidirectional flow for virtual from real environment. The virtual interface is used for visualization of reactive actions performed by the real objects. The second step is to promote interaction between these two worlds. Arrow 2 (in the right) illustrate this phase where we have interactive options for users that can create and manipulate virtual objects that will be perceived in the real world and influencing the agents actions. In this case, we have a platform for simulating the behavior of these agents in the real world.

3.1 Real World Overview

As stated above, the robots operate in an environment that could be any place providing interactions with its virtual version for showing results or else to allow manipulation on it via a virtual reality interface. To show the applicability of the system, we adopted in this work two such environments shown in Figure 3 and Figure 3. These environments are an arena and a robot soccer game, that is, both are closed places in the real world used for autonomous robots competitions. The surfaces on which the robots operate are limited by black walls and, in the arena case, have just one entrance. Inside this environment we can have real objects as robots and obstacles. The robots are autonomous vehicles that were implemented using Lego techniques [14, 16, 13, 12] for fast assembly. The obstacles are colored cubes or cylinders. Besides, we can have more than one robot in the environment so a robot could be a dynamic obstacle for another one. With this infrastructure, we can simulate many kinds of tasks as: (1) Moving around and deviating from obstacles; (2) Looking for special objects (as color cubes or cylinders); (3) Identifying and catching special objects.

Figure 3 – The arena environment.

3.2 Virtual World Overview

For representing the real world and the objects, we implemented a very simplified virtual version of the above real environments using VRML. Each real object is represented in this 3D interface by using a corresponding avatar. Other versions of avatars as people in the Internet accessing this virtual version could have place in the virtual environment. In this case, we may let the real robot know or not its presence depending on the application. This issue (augmented reality for robots) will be discussed later in this work.

3.3 The architectural model

Figure 5 illustrates the architecture design of the HYPERPRESENCE system relating the main architectural components and connectors. Briefly, the **3D Interface Component** represents the 3D virtual interface and the 3D

objects. The major function of 3D_Interface component is the visualization of the real environment events. The **VRML Multi-User Client** is the observer of events for the 3D Interface. For each event generated for an avatar in the virtual environment the *VRML Mult-iUser Client* sends a message to the *VRML Multi-User Server* component.

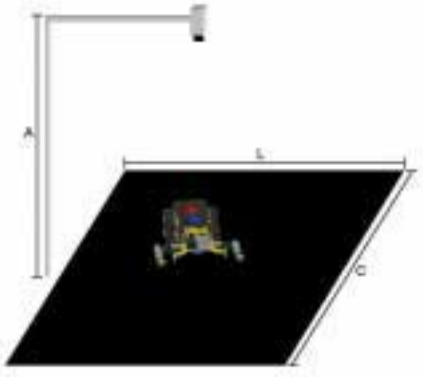


Figure 4 – The soccer game environment.

The **VRML Multi-User Server** component controls the VRML_MultiUser_Client components. For each real object (robot or obtacles) it is provided its geometric 3D representation – an avatar. Each avatar is treated by the VRML Multi-User Server as a client connected in this server. The VRML Multi-User Server feeds each client with the correspondent information (position and orientation) from the Visual Server and Hardware Server components (seen next), so it can keep the syncronization between both environments. Also, if a virtual avatar enters scene, its position and orientation is constantly updated.

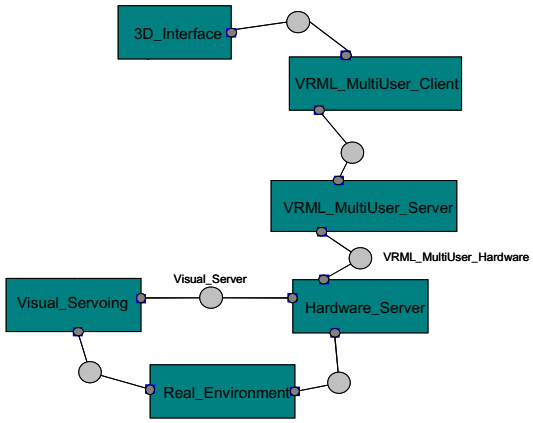


Figure 5 –System Architecture.

The **Hardware Server Component** is responsible for getting and sending information for the autonomous agents, that is, the robots in the closed environment. Although, the robots are implemented as reactive units the Hardware Server can send messages for each robot and update their inputs according with application needs. This component sends messages for the VRML Multi-User Server, too. The function of the **Visual Server Component** is to acquire/determine position and orientation of the real objects and to send them to the other components. To do that, the Visual Server component has an embedded video camera system, which uses image processing and recognition algorithms (discussed later) for mapping colors marks contained in each object in position and orientation, for each real object. The **Real Environment Component** is the physical environment and the real objects system represented by robots (Lego kits), cubes, cylinder and walls. Each robot has its own implementation code and this depends on the task goal to be achieved. Also, we can run different tasks for each robot, for example one robot can move and/or pick up a can.

4. SYSTEM COMPONENTS AND CONNECTORS

The functionality of each of the above, briefly described components and connectors as well as their integration in the proposed system are detailed in this section. Also, a description on how we have implemented each one of the components is given in the following.

4.1. The 3D interface component

This component was implemented by using VRML (Virtual Reality Model Language). VRML is a language for modelling and implementation virtual environments. VRML code is simple, supports modularization and objects reuse. We used VRML mainly because of it has the Internet support and Java integration facilities.

4.2. VRML Multi-User Client

This component is part of a multi-user server architecture for supporting representation of avatars in VRML environments. Each agent or obstacle in real mode is seen as an avatar and so it has a specific *VRML Multi-User Client* component for treating its events and actions. This component is a kind of an observer one. Each new event generated in the 3D world can be sent to the server by the *client thread* component and the same component can receive instructions from the server and execute them.

4.3. VRML Multi-User Server

This component represents the server side of the VRML multi user architecture. This component can be executed in two modes:

Syncronization Mode. *VRML Multi-User Server* is able to control the client events and actions according to the information of the real environment. In this way, the *VRML Multi-User Server* component takes input information from *VRML Server Hardware* connector and updates the 3D version in VRML code.

Interaction Mode. *VRML Multi-User Server* can also send control messages to the VRML Server Hardware in this case operating in an interaction mode where the user control

the 3D interface objetcs which are mirrored in the real environment. Otherwise, the user can insert new objects in the virtual world, as obstacles or new walls. So the agents in the real environment can react to them increasing a seamlessly approach on this application.

The *VRML Multi-User Server* is implemented in Java (JDK 1.3.1). More details of this component implementation can be found in [08]. We can point out some important components of *VRML Multi User Server* (called Vixnu Server) and *VRML Multi User Client* solutions:

Vixnu Server – controls and manages the functions that support these services. This server has a main control table that contains a list of all the "virtual worlds" (Environment Server) supported by Vixnu Server.

Environment Server – This server has two control lists: Slave Servers and the Avatars one. The first list links the *Environment Server* to the *VisualUserClient* and the second contains the information about each avatar. The avatar may represent a human user or a virtual one as a robot or an obstacle.

Slave Server – This component is used for observing the user actions and for sending them to the *Environment Server* where the messages are received and passed to the respective component.

VirtualUserClient – It´s the *VRML Multi-User Client* component which is implemented using HTML, VRML 2.0 and Java Applets. The VirtualUserClient is divided into four modules: VRMLModule, ChatModule, VoiceModule and CommunicationModule:

VRML Module – responsible for avatars perception.

Chat Module – creates an interface for supporting a chat service using only text.

Voice Module – responsible for supporting a chat service using **synthetized** voice.

Communication Module – improves the communication betwen the modules related above with the respective SlaveServer by using messages. These messages are composed by one byte integers representing an object identification (ID) and by three more integers of four bytes each. These three integers represent the X and Y coordinates in tenth of millimeters of the robot positioning and its orientation θ in degrees (a value between 0 and 360). The above data is encapsulated in a string that is sent through the network by using sockets. On the other side, a component has to unpack the sent message and to cast it to the desired data type.

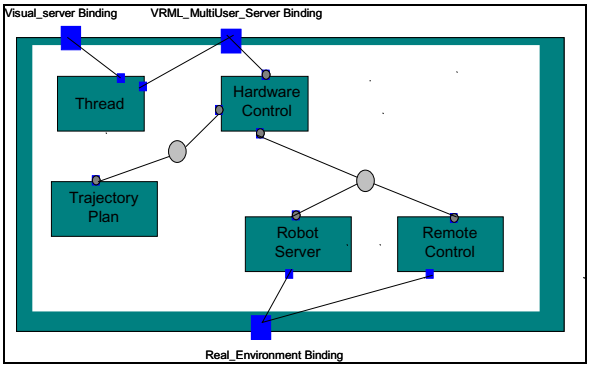


Figure 6 – Hardware Server Component Representation.

Connection between the Hardware Server (Figure 6) and Real Server is full-duplex. The Hardware Server sends position and orientation information to the Real Server which in its turn pass this information to the respective virtual agents that will move in the virtual world.

Also, the positioning and orientation information about the objects in the virtual world that comes from the Real Server are sent to the Hardware Server in order for this one to take actions. These actions will eventually move the robot in the real world updating its position and orientation. We note that, besides information about virtual objects, other information about obstacles that exist only in the virtual world could also be provided to the Hardware Server. In the same way, the above data is encapsulated in a string which is sent through the network via sockets.

4.4.1 Robot Server

This server controls three modules that have been implemented by using treads. These modules are: Remote Control, Trajectory Control, Virtual Control.

4.4.1.1 Remote Control Module

The Remote Control Module uses an API implemented in "C" language. This API allows this module, executed in a simple PC, to control the robots without querying about the communication process itself. Some examples of the developed functions are: motor control, speed and direction, read sensors, emit beep, etc. The developed API uses the LNP (Legos Network Protocol) [18]. This protocol uses an infrared tower for providing communication between the PC and the RCX (Robot Control Explorer) [16]. It sends commands packed into a string. The operational system chosen to operate inside the RCX was the BricOS, the current version of old named LegOS. [17]. We developed a shell (a command interpreter) that is executed in RCX mode (inside RCX) and that was compiled for BricOS. This shell receives all commands from the Remote Control Module (API), unpacks and executes them.

4.4.1.2 Control of trajectory

This component maps the real world into an internal geometric representation and is responsible for generating trajectories for the robots. That is, given an actual position and orientation and a goal (position and orientation), it generates all necessary commands for the robot to move from the actual to the goal positioning with the desired orientation. Issues as obstacle avoidance and trajectory planning are not addressed here since this is not the subject of the current work.

4.4.1.3 Virtual Control

This module passes information that exists only in the virtual world to the robots. An example of this kind of information could be position of an obstacle in the virtual world. We have made two implementation of this module.

We first implemented this module using a policy based on priorities. In this fashion, it could assume the control in situations. In this case, this implementation of the Virtual Control module assumes the control of the robot and makes the real robot to turn around the virtual obstacle as if this obstacle was present in the real world. We call this augmented reality for robots and another work is been prepared in this matter. After, the Virtual Control module returns the control of the robot to the Robot Server. This implementation was not finished, besides we have made another one to solve the Virtual Control problem, described as follows. In the second implementation carried out the value of sensory variables are constantly updated in the Remote Control API by the real sensors. So they have values that agree with the current environment values. In situations where we have virtual obstacles, we just change these values so they now agree with the virtual environment. This updating is done by the Virtual Control module directly in the API so the Remote Control does not matter about what is going on and the Virtual Control module does not need to take the robot under its control. That is, the Remote Control module does not distinguish between real or virtual information.

4.4.2 Thread Server

Its function is to serve as a bridge between the Visual Server, the Hardware Server, and the Real Server. Information provided by the visual server is sent to the above layer through the thread server. In its turn, it updates information in the sensors if necessary.

4.5 Visual Servoing Component

The Visual Server component is responsible for acquisition of images from the real environment. This component is composed of a colorful CCD (Charged Coupled Device) camera. The hardware used in the visual system is shown in Figure 7. We use a PCI 1411 board for image acquisition, this board supplies the processing system with an image in matrix representation, in two models of colors: RGB or HSL. These matrices are stored in a personal computer that is responsible for the image processing, part of the visual servoing system.

Figure 7 – Visual Servoing Hardware.

Figure 8 – Robots ColorFul Marks.

Processing the acquired image involves two phases. The first one is the calibration of the system in order to establish ideal parameters that allow the necessary classification of pixels, considering the illumination conditions and variations of each environment where the system is being used. This feature guarantees robustness for the system. With this phase done, the system can calculate, in the second phase, the objects localization and orientation in the scene. Basically, circular, colorful marks, called labels, are made on the mobile robot, as the examples that can be seen in Figure 8. Currently, we use the labels in a different positioning on the robot body. These labels are used in the image processing, in both calibrating and localizing process. Figure 09 shows an sketch of the working environment in an application characterized in this case by a black field where robots move (a robot soccer game). The color model to be used is defined based on the characteristics of the environment. The digitizer used in this work allows the use of RGB and HSL models. As the working environment in this application has the black color as the predominant one, what leads to a more complex picture processing when using the HSL, we have adopted the model RGB [09].

4.5.1 System Calibration

Calibration is an automatic process carried off-line, that is, when the system is not in operation. The calibration consists on the adjustment of parameters as resolution, contrast, definition of the colors, determination of the limits of the working space. The calibration process is done in two phases. The first one is carried out manually, where the user informs to the system some parameters as diameter and size of the labels, limits of the area covered by the camera, in

pixels and centimeters, and height of the camera in relation to the work plan. Some adjustments in the camera/digitizer system are also necessary, as focusing, brightness, saturation, contrast, frequency of the plate for image acquisition. As the calibration consists of the adjustment and classification of the colors, this phase has the purpose of increasing the robustness of the system, supporting small existing variations of luminosity in the image. For classifying the colors it makes use of techniques of grouping, what can use standard techniques or clustering techniques. One of these techniques is the K-means algorithm, adopted here. This is an algorithm based on heuristic, efficient and of fast convergence, a necessary requirement for a system that works in time real [07,10].

4.5.2 Finding position

An image is initially captured while the control system is not operating in order to locate the initial position of the mobile robots. Manually, through mouse or coordinates given by the user, a small region of the image is demarcated that contains the robot. A pixel with the color of the robot label is then found inside this region. From this pixel, another one is found that defines the position of the robot, given by the center of the label. Then, at run-time, the process of localizing a label consists of two phases: sweeping the image for finding a label and finding its center. The sweeping procedure aims to find one pixel with same color of the robot label inside the image. Since we have an estimated position for the robot, this is done by analyzing a fraction of pixels chosen inside the probable image area in which one has great probability of finding the label. From this pixel found in the sweeping phase, a small number of pixels neighboring will be analyzed in order to determine the position of the center of the label. The algorithm for the calculation of the center proceeds as: (1) Given a pixel *P* found in the sweeping, trace a horizontal segment until the boundaries say *P1* and *P2* of the label are found; (2) Determine the point *Pc* in the center of this segment; (3) From this point *P*c, trace a vertical segment, until the boundaries *P3* and *P4* of the label are found; (4) Calculate the central point in this segment (that also defines the center of the label). We remember the center is one pixel wide that may be close or (eventually) in the center of the image of the label. This introduces a small error in positioning that does not interfere in the whole process.

4.5.3 Finding orientation

At this time, a new search for finding the secondary label is performed. The center of this second label certainly is in a circle of radius D whose center lies in the first label center, where D is the distance (previously calculated) between the labels centers. Once this second label is found, the segment between the two labels can then be used to determine the orientation of the robot, given by

$$\theta = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

where points (x1, y1) and (x2, y2) are main and secondary labels centers. The visual server operates at every 33 miliseconds, that is, at a 30 frames per second rate.

4.6 Real Server Component

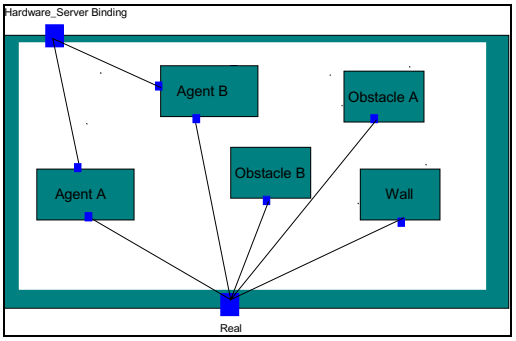


Figure 9 – Real Environment Representation.

This is a special component made of real objects. In Figure 9 we have an example of a possible configuration. In this case, we have two agents (Agent A and Agent B), the unique entity capable of communicating with the Hardware *Server* component, obstables (Obstacle A and Obstacle B) and a Wall component. The linking of each component with the *Visual Server* component is implemented with a visual mark on each component that is used for the visual processing for localization.

4.7 VRML Hardware Connector

This connector creates a virtual channel beetwen *VRML Multi-User Server* and *Hardware Server* components. Thought this channel flows the localization information in terms of orientation and positions of the objects. This is implemented by using sockets. This communication protocol is illustrated in Figure 10 where we can observe the messages flow between the components and how our approach identifies the objects and the position and orientation (angular information). This data is implemented as a string with the following format: *int ID, int x, int y, int θ*. We also define values for differentiating robots from the other objects.

4.8 Hardware Vision Connector

This connector (Figure 10) is responsable for linking *Hardware Server* with *Visual Server* components. This connector is sensitive for changes of the object and transmits messages each time that an object changes its orientation or position in the real scene. These messages are used for updating the virtual interface.

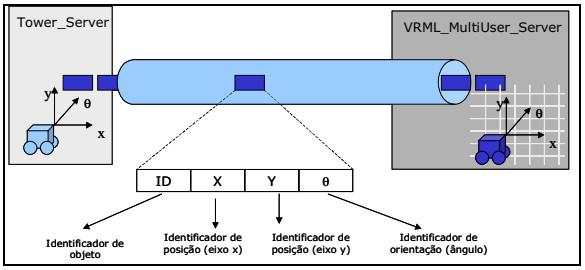


Figure 10 – VRML Hardware Connector.

5. EXPERIMENTS AND RESULTS

As we have not seen similar systems in the literature, it was not possible to design testing procedures against existing systems. So, the best practice that we have chosen to test our system was to try to put all sub-systems in operation and to test each one individually. Robot (hardware) server components are running with its control being made in a PC (in the DCA Robotics Lab) and a command interpreter running inside the Lego RCX. Visual servoing is running in another PC that communicates with the hardware server through a TCP/IP Internet protocol (a 10 Mb shared network connection). Visual information (position and orientation) has to be sent to the robot control server in order to make it follow a given trajectory planned by the trajectory-planning server. The Real Server is running in another PC (far a two hundred meters away, in the NatalNet Lab) communicating with first ones through another shared Internet link. 3D interfaces using VRML are running in other PCs (in our case, we started a virtual interface 500 hundreds meters away in the Virtual Reality Lab). We have encountered some operating problems, stressed and solved during these initial tests. One such example was how to unpack messages using Java.

In the first implementation tests, we observed times that each system spent to send a message and to actually show virtual avatars. Sending a message is not currently a problem, since its composition is very simple, as seen above. Actually, all servers could do this in real-time, respected some Internet constraints. That is, the system as a whole could keep tracking of positions and orientations of all objects (virtual and real versions), respected some constraints. One such a constraint must be imposed for example if one wants for a virtual avatar to move in its environment and the robot version of it to follow the same position and orientation in the real world. In this case, we just can limit the virtual avatar velocity so the robot control module can have time to receive and process its position and orientation. The visualization of he virtual world in the 3D interface using VRML was the slowest part, mainly when we put more than one robot in the real world. In order to test the communication between robot control server and the Real Server, we have initiated the system operation and sent a planned path that a robot could follow. This path was following in –45 degrees and turning 90 degrees right (to +45 direction). The left of Figure Figure 11 shows the sent numbers and the right of the figure shows the path performed by the virtual version of the robot. In this case, the real robot was not operating.

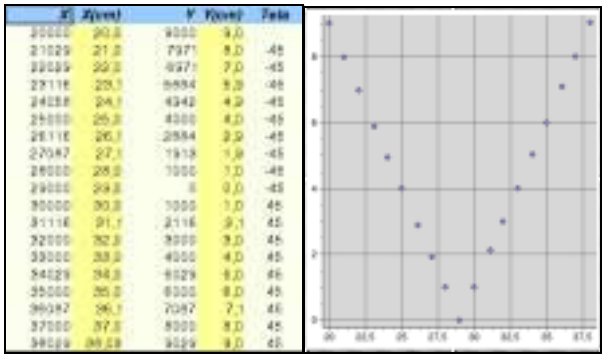
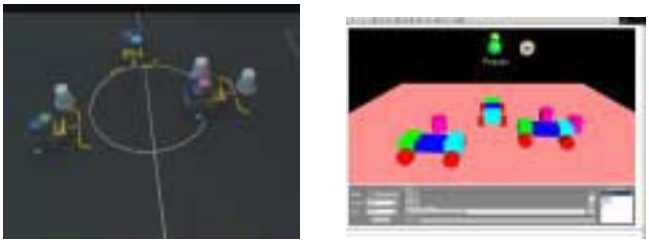


Figure 11 – Path planned simulated and received in VRML.

Figure 12 (a) shows a similar experiment with three robots inside the game field. In this case, the positioning coordinates and orientations of the robots are taken by the Visual Server and sent to the VRML that pass them in order for the virtual versions of the robots (Figure 12 (b)) to be visualized. In this case, if the robots move through the robot server controller, the virtual version of it can be updated.

6. CONCLUSIONS AND PERSPECTIVES

By using the proposed architecture, we could implement a mixed reality environment, in which several servers operate in a distributed and asynchronously way. In the current work, were mainly discussed results showing position and orientation of robots on the remote (virtual) site, but the opposite can currently be done too, without changes.



(A) (B)

Figure 12 – Robots configuration. (a) real and (b) virtual worlds.

The main drawback of our system is the need of a geometrical (virtual) copy of the environment in order to allow the proposed kind of interactions. In a constructive fashion, we are also working on a system in which several robots can together provide and send a map of the environment. In this case, one could send the robots

previously to interactions for constructing such a virtual map (VRML based). This is another work, of course.

A first problem that appears with our system is related to the transfer rate between the servers (virtual/hardware and real). While the real system could be receiving information, the visual server could be calculating and sending the next one so it may write over the previous one. This is a problem that may happen with asynchronous communication protocols in this case the reception is done at a lower rate than the visual system can operate (30 Hertz). A way to avoid such inconsistencies is to put constraints in the motion and information exchange so the several systems knows the rate each other operate, in this case, a synchronous protocol could also be implemented here. Besides this problem, we could visualize the virtual environment changing its contents according to these coordinates sent by the visual server in a well precise fashion.

There are several types of applications that can use a system like the one shown. For example, the robots can serve in a museum as (guide) real-avatars for people in the Internet, which could be looking the museum through a virtual implementation of a geometric model of it. In this case, the robot can emulate the movement of the person through the museum. Also, one can have several instances of the virtual environment running, each one attached to a person in a different site. In this way, someone in the real site can keep control on how many people are interested in the museum. Yet, the same architecture can also be used in several other situations where people could not be present in the environment (hostile environment, under radiation, closed).

We are currently working on the interface to provide the robots with augmented reality. Imagine a virtual agent enters the environment, but there is no available robot for representing it in the other side. But, the available robots have to know this person is present in the environment. At some time, the robot must be provided with augmented sensing information so it can avoid colliding with this virtual agent. Yet, a robot could pick up a virtual can that only exists in the virtual environment. So a next step is to test sending commands as pick up a can, move close to a place in the environment. This will be very useful to test working capabilities of a full-duplex communication and commands for all agents (real and virtual) in the environment.

7. REFERENCES

[1] S. Ellis. *What are Virtual Environments?* IEEE Comp. Graphics & Applications, 14(1):17-22, Jan 1994

[2] A.Paris and C. Adonis. Exploring Mars Using Intelligent Robots. (White paper).

[3] Moran, T.P. and Anderson, R.J. The Workaday World as a Paradigm for CSCW Design. In Proceedings of the Conf. on Computer-Supported Cooperative Work, CSCW '90. (Los Angeles, CA), pages 381–393. ACM Press, New York, 1990.

[4] P. Milgram and F. Kishino. A Taxonomy of Mixed Reality Visual Display, IEEE Transactions. on Information & Systems.,Vol. E77-D, No.12, pp.1321-1329, 1994.

[5] B. Koleva, H. Schnadelbach, S. Benford and C. GreenHalgh. Traversable Interfaces Between Real and Virtual Worlds. In: CHI2000 Proceedings.

[6] S. Ressler, B. Antonishek, Q. Wang, and A. Godil. Integrating Active Tangible Devices with a Synthetic Environment for Collaborative Engineering. National Institute of Standards and Technology.

[7] Anil K. Jain. Fundamentals of Digital Image Processing. Prentice Hall, 1 edn, 1989.

[8] A. Burlamarqui, T. Tavares, and G. L. S. Filho. Vixnu - Um Servidor Multi-usuário com Suporte a Comunicação em Ambientes Virtuais Colaborativos. In Proc. of VIII SBMIDIA - Brazilian Symposium on Multimidia and Hipermidia Systems. Conf. held at Fortaleza, CE, Brasil, October, 2002.

[9] K. R. T. Aires. Desenvolvimento de um sistema de visão global para uma frota de mini-robôs móveis, Master's thesis, Federal University of Rio Grande do Norte, Natal, RN, Brasil.

[10] Haykin, S. S. Neural Networks, Principles and Practices, 2 edn, Bookman Company ED.

[11] Shared Spaces: Transportation, Artificiality, and Spaciality. Steve Benford, Chris Brown, Gail Reynard and Chris Greenhalgh. Department of Computer Science, University of Nottingham. NG7 2RD, UK. E-mail: {sdb, ccb, gtr, cmg}@cs.nott.ac.uk

[12] www.lego.com/dacta/robolab/default.htm ROBOLAB Homepage - (20/07/2002)

[13] NQC Homepage - www.baumfamily.org/nqc/

[14] www.natalnet.br/~tati/kalango_arquivos/frame.htm Kalango Homepage - (20/08/2002)

[15] Billinghurst, M., Poupyrev, I., Shared Space: Mixed Reality Interface for Collaborative Computing. Imagina'2000 Official Guide: Innovation Village Exhibition. 2000. INA. pp. 52. (PDF)

[16] LEGO Dacta. LEGO MindStorms. http://mindstorms.lego.com.

[17] BRICOS. BricOS Operating System and Compiler "C" and "C++" for RCX. brickos.sourceforge.net.

[18] LNP: LegOS Network Protocol. legos.sourceforge.net/HOWTO/x405.html