Robot Arena: An Augmented Reality Platform For Game Development

DANIEL CALIFE, JOÃO LUIZ BERNARDES, JR., and ROMERO TORI
Escola Politécnica da Universidade de São Paulo – USP

Nowadays electronic games are of great importance to the economic sector, to computing, and to academic research, and not are limited to entertainment applications only. Augmented reality is one of the new frontiers to be explored in the search for innovation in gameplay and in interactive interfaces of electronic games. This article presents the research, development, and testing of an infrastructure, called Robot Arena, for the development of innovative games using the spatial augmented reality and new interfaces that this technology brings. Based on horizontal interfaces with elements of hardware and software, Robot ARena has a flexible architecture that enables different ways of interaction and visualization. Another aspect planned for this project is the control and communication of robots, which can be used as avatars in games. A Robot ARena prototype was implemented, tested, and used in the development of two game prototypes, *FootBot ARena* and *TanSpace*, which bring new challenges to the infrastructure, such as exploration of the influence of virtual objects on real objects and the application of tangible interfaces.

Categories and Subject Descriptors: I.4.9 [**Image Processing and Computer Vision**]: Applications; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction Techniques*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Virtual Reality*

General Terms: Design, Experimentation

Additional Key Words and Phrases: Augmented reality, games, robots, tracking, virtual reality, human computer interfaces

ACM Reference Format:

Calife. D., Bernardes, J.L., Jr., and Tori, R. 2008. Robot Arena: An augmented reality platform for game development. ACM Comput. Entertain. 7, 1, Article 11 (February 2009), 26 pages. DOI = 10.1145/1486508.1486519 http://doi.acm.org/10.1145/1486508.1486519

1. INTRODUCTION

Although electronic game interfaces have evolved in time, enabling the creation of more graphically sophisticated environments and providing more elaborate controls, the way of providing input to games has not undergone any

Authors' address: INTERLAB – Interactive Technologies Laboratory Escola Politécnica da Universidade de São Paulo—USP, Av. Prof. Luciano Gualberto, travessa 3 no. 158 – CEP 05508-940-São Paulo, SP – Brazil; email: {daniel.calife,joao.bernardes,romero.tori}@poli.usp.br

major changes, with some exceptions still mostly using directional controls and push buttons, or mouse- and keyboard-based interaction. Augmented reality (AR), combining 3D real and virtual environments, interactively breaks this paradigm and makes new ways of interacting with games possible. It also allows an improved collaboration for this environment and stimulates a more direct interaction between players, even making it possible to eliminate avatars or text-only chat interfaces within the game, extending its social aspect [Magerkurth et al. [2004] affirms that in traditional electronic games, in computers and consoles, this type of interaction does not exist due to its technology-centered, limited interactions between players. With AR, the game may be brought to the real world, and people can have a more physical, personal, and emotional interaction as in traditional board or card games.

Aiming to analyze the new possibilities brought by the application of AR in games, Nilsen et al. [2004] characterize player experience in four aspects: physical, mental, social, and emotional. Based on these aspects, they compare the features of traditional and electronic games. No game uses only one of these aspects; the majority combines all of them. While electronic games show advantages in some aspects, traditional games are better in others. Computer games, for instance, shine in the application of artificial intelligence, attractive computer graphics, remote communication, and the automatic calculation of game rules. Traditional games often have more natural interfaces and a richer social interaction between players. The great advantage of the development of games with AR is the possibility of mixing the best features of both, making AR one of the important new frontiers to be explored in games.

Nowadays research projects in AR games can be classified into two broad categories: indoor and outdoor [Bernardes et al. 2005]. These classifications are based on the physical space demanded by the game: while outdoor games occupy large areas, indoor games usually require small, prepared spaces. This factor has a great impact on the technologies and implementation of the games, especially for registration and interaction. Outdoor games mostly use physical metaphors of interaction, such as colliding with virtual objects to collect them, touching other players, running or walking in some ample physical space to move within the game. Indoor games also use the physical metaphor, but to a lesser extent, often in the form of tangible user interfaces (TUI) [Ishii and Ullmer 1997], and combining it with other interaction metaphors such as player gesture or voice recognition.

1.1 Robot ARena's Objectives

This work presents an infrastructure, called Robot ARena [Calife et al. 2007a, 2007b], which can be used as a development platform for innovative indoor AR games based on spatial augmented reality (SAR) [Bimber and Raskar 2005] (which brings the visualization of the augmented content directly into the physical space) and a wireless controlled robot or tangible interfaces. This platform can be used to explore new ways of AR projection-based interactions with robots (real and virtual). The use of robots (which, for the purpose of this article, means any objects whose positions and other properties may be controlled by a computer) is interesting because it is one way to allow virtual objects to appear to actually affect real objects physically, instead of only the opposite happening, as with many tangible interfaces. Robot ARena may also be adapted for educational applications such as competitions for robots designed and programmed by students.

Based on horizontal interfaces [Shen et al. 2004], where the interactions are done on the surface of a table and composed of hardware and software elements, Robot ARena is an architecture that enables many ways of interaction and visualization. It can implement applications that, for example, use object tracking, gesture recognition, or multitouch interaction, and both front and back projection systems.

Another aspect of the Robot ARena project is the control of and communication with robots, which can be used, for instance, as avatars inside games. This control allows a two-way interaction between robots and the projected images in the arena, so that not only real elements affect virtual ones but also virtual elements can collide with remote-controlled real objects and "push" them or represent an obstacle to their movement.

As a proof of concept a Robot ARena prototype was implemented with the following characteristics: a retro-projection surface which allows the visualization of AR elements; an object tracking system; and the control and communication of a Lego robot [http://mindstorms.lego.com/].

Using this proof of concept configuration, the following game prototypes were developed, exploring the characteristics of this implementation:

- *FootBot Arena*: a game that explores the interactions between real and virtual elements and has focus on the visualization of virtual content;
- *TanSpace*: which implements the use of a tangible interface, where the manipulation of a physical element is used as a player input.

2. RELATED WORK

Since the end of the last decade, several applications using AR as an interface have been described in the literature. More recently several projects combining AR and games have also been presented. In the field of robotics, in particular, computer vision and augmented reality (and, of course, robots) have been used and combined even more extensively. This brief review is limited to more recent projects and those that are more closely related to the work described in this article. Some of the tools used in Robot Arena are also presented.

2.1 EnJine

EnJine is an open-source didactic game engine [http://www.enjine.org/; Nakamura, et al. 2006] that uses Java and the Java3D library. It is developed and supported by the Interactive Technologies Laboratory (Interlab) at University of São Paulo, and available as free software under the GPL license.

EnJine's main purpose is to serve as a teaching support tool in computer science courses, especially for computer graphics [Tori et al. 2006]. It is therefore didactic and relatively simple to learn and use, and has an architecture that

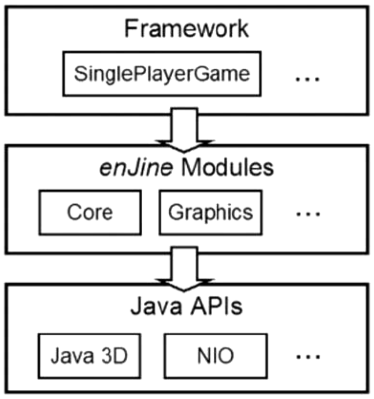

Fig. 1. EnJine's architecture [Nakamura et al. 2006].

facilitates good software engineering practices. Another purpose for enJine is to serve as a framework for implementing and testing new technologies, mainly applied to games.

EnJine provides a set of services for game development, as well as a structure for the creation of these games. Its main features areas follows:

- A set of Java classes to allow flexible representation of game elements such as characters, objects, and scenarios;
- Basic infrastructure for a game application, including the capability for complex interactions between game objects through a message exchange mechanism;
- 3D graphics rendering through the Java 3D API;
- Multistage collision detection and ray-casting collision detection;
- Skin-and-Bones animation;
- Support for basic sound output; and
- An input devices abstraction layer.

EnJine's architecture is divided into three layers. The lowest layer represents the external libraries used to communicate with the operating system and devices, and some other services such as Java3d rendering. The middle layer contains enJine's main modules, which implement the games' services such as video rendering, sound output, collision detection, and others. The top layer implements enJine's framework, which is a set of classes intended to provide a more complete and easier solution for the implementation of specific types of games, as opposed to enJine's more general nature. The framework includes concrete implementations of many of enJine's classes suited to specific uses. For instance, the SinglePlayerGame class implements the base code to aggregate enJine service providers needed for a computer game that runs in a single machine. These abstraction layers are shown in Figure 1.

The software infrastructure developed in this work extends enJine's middle layer, which is composed by the packages shown in Figure 2.

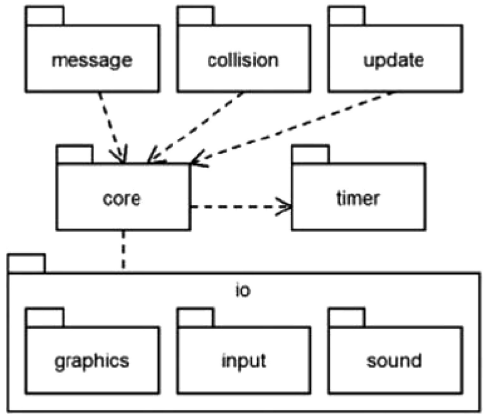

Fig. 2. EnJine's main packages [Nakamura, et al. 2006].

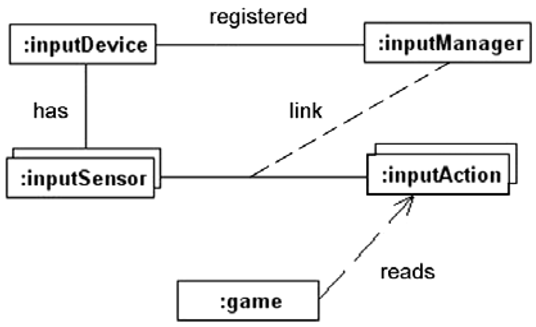

Fig. 3. EnJine's input layer architecture [Nakamura, et al. 2006].

In the context of this work, the main package is Core, which has the classes that represent the objects of the game and the class GameState.

The GameState is used in enJine to represent the games' different operational states such as title screens, levels, or game stages, and "game over" screens, and to represent operational modes inside the game as an inventory in an RPG. During the execution of a game there is always an active GameState, representing the state that the game is operating at that time [Nakamura, et al. 2006]. A simple game can be implemented using only a single GameState.

Another important enJine package is Input, which implements enJine's input abstraction layer. This input implementation simplifies the use of new input devices, such as Robot Arenas's Registration and Tracking system. Therefore, the behavior of enJine's input layer should be understood. Figure 3 illustrates its architecture.

The different input devices are defined by the concrete implementations of the abstract InputDevice class, which has a collection of InputSensor objects, that describe the device's capabilities (e.g., for a keyboard, each key should be represented as a sensor). Each InputSensor has a numeric identifier, a description and a floating-point value, which can either store discrete values to act as a button or a key, or continuous values to handle inputs such as mouse movement, analog controllers, object tracking and so on. The values of these sensors are updated by InputDevice.

The InputManager is responsible for maintaining the registry of input devices and carry out the connection between each InputSensor with an InputAction. InputActions are objects that represent commands with specific objectives in the game. They may be user commands like “jump” or system-triggered commands like “update the x position” in the case of a tracked object.

2.2 EnJine + JARToolKit

Tsuda et al. [2007] present an integration of JARToolKit [Geiger et al. 2002] and enJine, whose objective is the simplification of the development of low-cost AR games, increasing the potential of enJine as a didactic tool, and a testbed for new game technologies.

This integration with the JARToolkit gives enJine the capacity to recognize commands given with the use of fiducial markers and a video camera, besides reproducing the video stream obtained from the camera and tracking these markers. These characteristics, combined with enJine, facilitate the development of monitor-based AR games without the need for complex (and expensive) systems for tracking or stereoscopic visualization.

2.3 DWARF

The DWARF (distributed wearable augmented reality framework) [MacWilliams et al. 2003] is a framework based on distributed services (position tracking, three-dimensional rendering, input and output, and task modeling) for augmented reality applications. Each service is a piece of software that can run on separated hardware components, with its own processor, memory, I/O devices, and network access.

The different services are dynamically connected with other services communicating their needs and abilities through the network. Distributed middleware is responsible for controlling the communications between these services.

This framework is used to develop complete augmented reality applications, projected to be flexible and guarantee to users a high degree of freedom. These characteristics are attained because the framework’s services can be distributed over independent machines, which can be added or removed from the environment without causing a great impact on the whole system.

2.4 Augmented Coliseum

Augmented Coliseum [Kojima et al. 2006] is an application similar to the game prototype described here. It is a game in which augmented reality is used to enrich a competition between real robots with effects such as rays and explosions in a virtual environment. Robots in the virtual environment can also collide with obstacles that can block their movements. Unlike the work described in this article, however, there is no need for visual tracking. The position of the robots is always known because they actively follow a certain pattern projected on them along with the virtual environment they are in. Each robot has five light sensors in known positions that detect variations in the light pattern and can identify where it is moving to and then follow it.

2.5 Kobito

The Kobito project [Aoki et al. 2005] is an interesting one, it makes use of augmented reality and virtual and real objects interacting "physically". In this application, a real object, a tea caddy on a table set for a tea break, is moved around by "invisible brownies," virtual agents that can only be seen through a special display, the "Kobito Window," which is nothing more than an AR interface for visualization. The caddy is actually moved around by a magnet under the table, which, in turn, is manipulated by the SPIDAR system. The virtual agents not only manage to push the real caddy around, but can also be pushed by it (in fact, that is the only way to interact with the brownies, i.e., through a real object) if a user manipulates it. The caddy even offers force feedback, representing resistance by the brownies. The system makes use of physical simulation, but unlike its classical applications where all simulated elements are virtual, there is a real element that must be tracked precisely (this task is done with a camera) and manipulated (with the SPIDAR system). Unlike the manipulation system used in the Kobito project, so far the robot used in this Robot ARena configuration can only turn and move back and forward (it cannot be pushed sideways, for instance), so the degree of physical interaction with virtual objects will necessarily be limited, and must be designed to hide this problem as much as possible from the user.

2.6 Considerations

The related work mentioned in this section share several features with Robot ARena, such as the interaction between real and virtual objects, tracking using computer vision, and a focus on gaming applications. This is sometimes due to the utilization of, or at least the inspiration from, these works in the infrastructure described in this article.

EnJine has been the basic technology used in several projects at Interlab, where Robot ARena was developed. That fact, the know-how due to this, and especially enJine's excellent stability and flexibility were the main deciding factors for its adoption as the fundamental structure for the development of Robot ARena.

The integration of enJine and jARToolKit inspired this work, not only because it was the first application of AR using enJine but also in the way it made use of this game engine's input layer.

The DWARF project is an example of an infra-structure that allows several different configurations, with different forms of visualization and the use of several different input devices attached to the system through a flexible architecture, much like Robot ARena does.

The Kobito and Augmented Coliseum applications show an area rarely explored in AR, the influence of virtual objects on real ones, and in the Augmented Coliseum this interaction is realized through the use of tele-operated mobile robots, as in Robot ARena.

3. OTHER TOOLS AND TECHNOLOGIES

Besides enJine (described previously), which was adopted as a fundamental core for the development of the infrastructure's software described in this article, this section presents the tools and technologies used to develop both the logical and physical components of Robot ARena.

3.1 OpenCV

OpenCV [http://www.intel.com/technology/computing/opencv/] (Open Source Computer Vision Library) is a free library of C/C++ functions mostly for computer vision and image processing, with a focus on real-time applications, which makes it interesting for augmented reality.

Because two of the main concerns of augmented reality are registration and tracking, and one approach to obtain them is through computer vision, OpenCV is a robust and flexible tool for such applications.

3.2 Lego Mindstorms

Lego Mindstorms [http://mindstorms.lego.com/], or Lego robotics invention system (RIS), is a set of blocks, motors, sensors, and other components such as gears and axles, created and marketed by the Lego group. It can be used to build many types of robots and other automated or interactive systems. In the implemented configuration of Robot ARena, it was used to build the robot.

The version used, Mindstorms NXT, has an Atmel 32-bit ARM microprocessor and both USB and wireless communication interfaces; the wireless interface uses Bluetooth.

3.3 ICommand

ICommand [http://lejos.sourceforge.net/] is a JAVA API to control Lego NXT through a Bluetooth connection. To implement the Bluetooth communication, the BlueCove API [http://bluecove.sourceforge.net/], which implements JSR-82, along with RXTX [http://www.rxtx.org/] is used for communication through the serial ports emulated by Bluetooth.

3.4 Camera

A C3 LT-359A camera was used, with a 55° view angle, which allows the capture of the entirety of Robot ARena's surface, a VGA CMOS sensor with 640×480 resolution, and USB 2.0 communication, allowing a rate of communication of 30 frames per second.

The use of a few other cameras was tested, such as LG WebPro2 and Logitech QuickCam 4000 and 5000, mostly to compare the susceptibility of these cameras to infrared lighting. All the webcams tested, however, have filters to block infrared; but their removal is difficult, risking damage to the camera lens. For lack of a camera able to capture infrared well, an infrared-tracking device, which was planned originally, was not implemented or used in this configuration of Robot Arena. Testing the use of such a tracking device (especially to reduce interference from the projected virtual image with tracking) remains possible future work.

3.5 Projector

The projector was a Hitachi CP = X255, with the following characteristics: 3 LCDs, XGA 1024 × 768 resolution, 2000 ANSI lumens, and 400:1 contrast. An important feature of this projector is its throw distance, which allows, using zoom, a projected area of 120 cm × 90 cm at a distance of 180 cm. This facilitates the positioning of the projector in Robot Arena in both the front- and back-projection configurations.

4. ROBOT ARENA DEVELOPMENT

Robot ARena is a hardware and software infrastructure for the development of games or other interactive applications using an interface based on horizontal surfaces, tele operated mobile robots, tangible interfaces and spatial augmented reality projections. This infrastructure is based on enJine [http://www.enjine.org/], a game engine that provides a software framework for the creation of virtual worlds, games, and other interactive environments.

The objective of the robot is to be a real element of interaction that can be affected by virtual objects in the augmented environment. To reach this objective the robot is controlled remotely. As an example of implementation, a simple robot was built using Lego Mindstorms [http://mindstorms.lego.com/] and a control and communication module was developed for this robot (any other tele-operated robot may substitute in the implementation described here).

Working as a horizontal interface, Robot ARena was built on a table, which allowed the exhibition of virtual content both by front- and back- projections and using the concept of spatial projection-based devices [Bimber and Raskar 2005]. Cameras may be added to this structure with either a top view of the surface, or an underside view, allowing the tracking of devices (or fingertips, hands etc.). With this flexibility in configuration, Robot ARena allows several forms of SAR-based visualization and many possibilities of interaction such as gesture recognition and object-tracking with a top-view camera or a multitouch surface with an underside camera, which tracked the fingertip shadows or using the technique described in Han [2005], for instance.

EnJine's input abstraction layer was an important component in reaching this flexibility. One of the implementations of an input device, for instance, is the object-tracking component, which may be used to register the position of real objects in the augmented environment. This allows the tracking of a marker that may be attached to the robot or to a tangible interface device in Robot ARena.

4.1 Architecture

Robot ARena was implemented as a layer above enJine's three original layers, but still as part of enJine, an extension that brings new possibilities of coding, interaction and visualization in games made with the engine. Figure 4 shows Robot ARena inserted into the layered architecture of a game using enJine.

Figure 5 shows a more detailed view of the Robot ARena layer in Figure 4, illustrating the software architecture of the developed infrastructure.

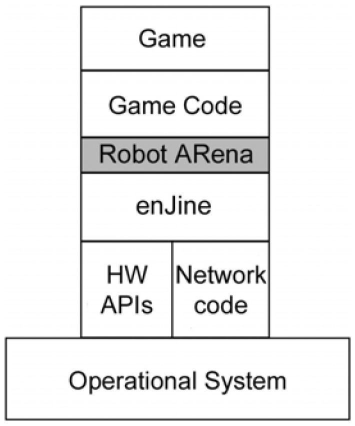

Fig. 4. Robot ARena in a game structure, based on Lewis and Jacobson [2005].

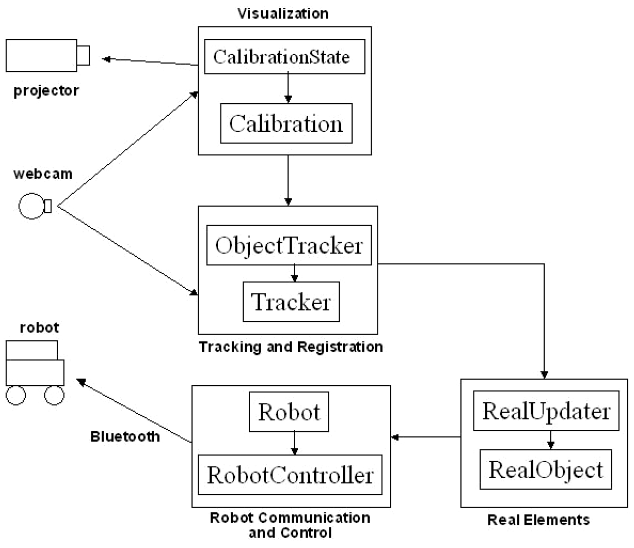

Fig. 5. Robot ARena architecture.

Robot ARena's hardware is composed of the table or projection surface, a mirror tilted at 45°, one or more cameras and projectors and the robot or any other tracked or tele-operated object, aside from a computer.

The infrastructure's software was divided in four subsystems, some of which access elements from the physical hardware which, for simplicity, are considered black boxes that simply send and receive signals. The four software subsystems are

Visualization. The visualization subsystem provides a class that computes the parameters of the camera within the virtual environment, which, in turn, determines how this environment will be rendered so that the projection can be correctly calibrated onto the desired surface. This calibration stage as implemented currently can either be resolved automatically using computer vision or manually by the infrastructure's user. This subsystem was implemented in a GameState, allowing it to be attached to any game at any time, or run independently.

Tracking and registration. This subsystem translates to a component that provides an interface with methods to realize the registration between the real and virtual elements of the augmented environment as well as the tracking of real objects. For future extensions of and implementations using Robot ARena, this subsystem serves as a standard, exemplifying the use of enJine's input layer and allowing new forms of interaction to be easily integrated to Robot ARena.

Real elements. This is a set of classes that inherits from enJine's GameObject and specifies the characteristics of objects in the application that have a physical component. The position of these real elements is computed and updated by the tracking and registration subsystem.

Robot communication and control. This set of classes represents the real robot and provides an interface for its communication and control, allowing movement commands to be sent to it either by a player or by the application, for instance when virtual elements affect the robot position independently from the player's control. This subsystem sends signals to the robot according to the updates provided by the RealUpdater class. According to the technology used to build the robot, it may be necessary to integrate other subsystems similar to this to the infrastructure.

These subsystems may be used separately, depending on each application; other subsystems may easily be added to insure Robot ARena's flexibility.

4.2 Hardware Implementation

This section describes the implementation of Robot ARena's physical infrastructure in its current configuration.

4.2.1 *The ARena.* The ARena's current version was developed to offer more flexibility allowing different configurations of projectors and cameras (the first version, for instance, assumed frontal projection only). Figure 6 shows a representation of the ARena with a back projection and top-view camera configuration.

The ARena built to test prototypes of the system is 140 cm-wide by 110 cm-long and is set at 100 cm of height. The top is composed of crystal acrylic covered with a screen that allows back projection and the usable projection area, excluding the structural frame, has dimensions of 120 cm $\times$ 90 cm, keeping a 4:3 ratio. In the ARena's inferior portion is a mirror at 45° that allows the deflection of the light from the projector, allowing back projection on the entire surface without the need for a very high table or a very wide angle of projection. This mirror also allows the visualization of the underside of the projection surface by a camera.

The decision to use this back projection and top-view camera in this proof of concept and prototypes is mainly due to the simplicity and practicality of assembling the system, as Wilson [2005] states (and as mentioned before), other configurations were also tested.

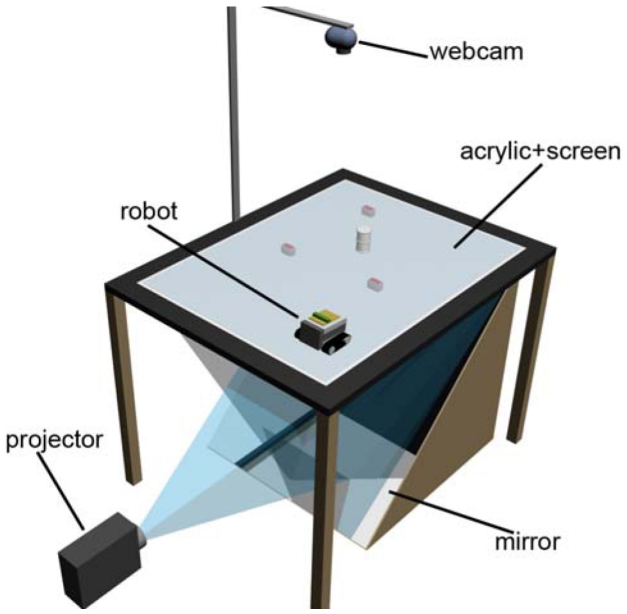

Fig. 6. Robot Arena hardware setup.

4.2.2 *The Robot.* The Robot, assembled using Lego NXT, allows communication through a Bluetooth connection, and, with the use of ICommand, a better control of the robot's characteristics (e.g., controlling the speed of each of its motors separately and the use of sound effects and other NXT features) is possible. This means that all the robot's movements and responses need not be programmed a priori, but the robot's sensory inputs should simply be responded to. In general terms, the current version of the robot uses caterpillar treads and differential steering that allows the robot to rotate around its own central vertical axis without altering its position. This is an important feature that simplifies collision detection with virtual objects. Figure 7 shows the robot assembled with NXT.

4.2.3 *Tracked Devices.* At first, Robot ARena tracked two distinct types of devices. One is a color marker assembled with Lego bricks and the second a device with two infrared LEDs. The color marker was created specifically to attach to and track the robot and consisted of two Lego bricks forming a "T," as shown in Figure 8.

The infrared tracking device is a small box with two LEDs that can be easily manipulated by users or attached to some other object that must be tracked. The two LEDs have a similar function to the colors in the Lego marker: easily determining the marker's orientation. To differentiate between both LEDs, a simple circuit regulates their intensity so one of them is considerably brighter than the other, allowing their individual identification. A representation of this device can be seen in Figure 9.

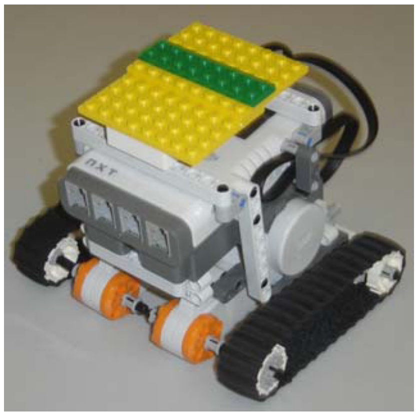

Fig. 7. The final version of the Robot, assembled with Lego NXT.

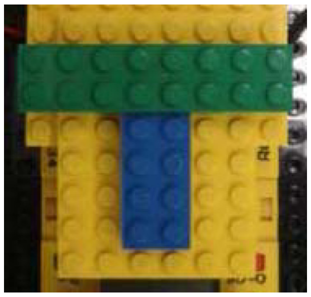

Fig. 8. Color marker made of Lego bricks.

In the current configuration of Robot ARena, a different Lego marker is used and the infrared device was discarded due to the presence of IR blockers in most webcams, making their sensibility to this spectrum vary too much. The new marker, shown in Figure 10, can be tracked using the algorithm presented in Van den Bergh and Lalioti [1999] which segments a single color channel in the image (in this case, yellow).

4.3 Software Implementation

This section presents and discusses the implementation of Robot ARenas software by subsystem.

4.3.1 *Visualization*. In this subsystem, CalibrationState represents the visualization (i.e., projection) calibration mode, usually a state that precedes the game's beginning. In the case of a fixed ARena configuration, this state may be run a single time and have its parameters recorded and updated again only if there is some change in the ARena's physical configuration. The class diagram shown in Figure 11 represents this subsystem.

The CalibrationState class has an instance of camera, which represents the virtual camera inside the game in enJine. This means that the angle of view and position of this camera determine how the virtual environment will be

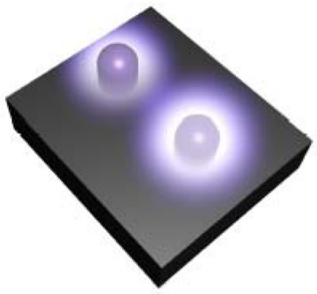

Fig. 9. Infrared-tracking device.

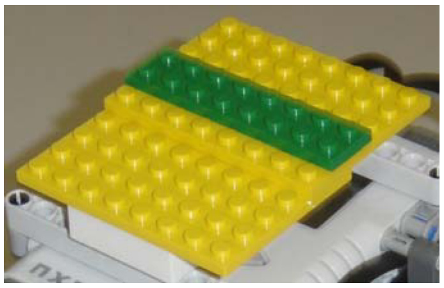

Fig. 10. Another Lego marker.

projected on the ARena. It is necessary, then, to calibrate this camera to insure the alignment of the projected image with the surface of projection.

CalibrationState basically shows a standard image consisting of two white squares of different sizes that must then be aligned with the surface of projection. This alignment may be performed manually using keyboard commands to rotate and translate the camera until the pattern is properly aligned and centralized on the ARena, or it can be performed automatically using the Calibration class, which makes the calibration using computer vision to segment these squares and find their position, and thus determine the correct position for projection. In both cases, the calibration parameters that are found may be saved to a XML file e, loaded later to dispense with any unnecessary calibration stages.

The calibration follows some steps which must occur in the following order, even when done manually:

- Determine and correct the virtual camera's orientation in getOrientation();
- Determine a scale factor from pixels to generic 3D units in getPixelto3d();
- Correct the X and Y camera positions; Z is fixed because the camera has a top view of the game and the ARena. This happens in getXTrans() and getYTrans();
- Find the size of the real robot, if necessary, with getRobotSize();
- Save the parameters that are found with saveParams().

Registration and tracking. The tracking and registration subsystem was divided into two classes: the ObjectTracker class, which calculates the posture,

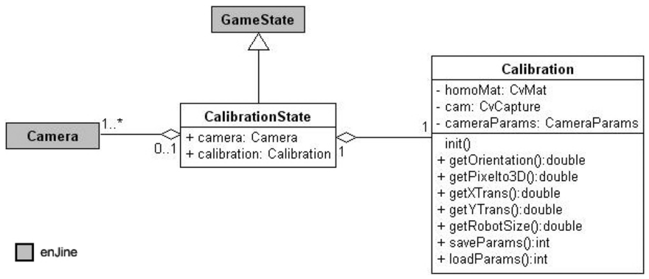

Fig. 11. Visualization subsystem class diagram.

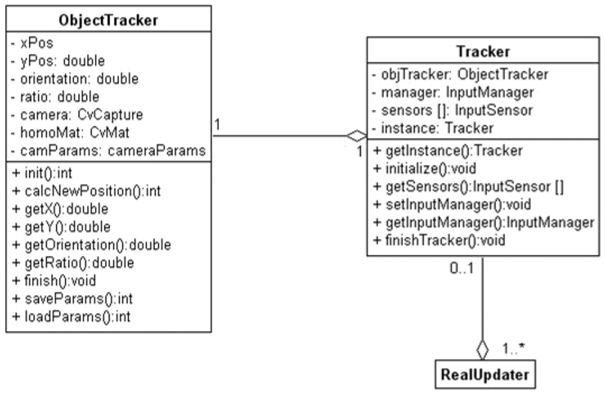

Fig. 12. Tracking and registration class diagram.

position, and orientation of the tracked object and is implemented in C++ using OpenCV; and the Tracker class, in Java and using enJine the input device through its sensors, updates the logical state of the virtual representation of the tracked object. These two classes are integrated using JNI, as in the visualization subsystem.

EnJine's input layer, through an input device, was used to implement this subsystem. Likewise, other devices may be created, allowing other forms of interaction to be easily integrated to Robot ARena.

Using this layer, the tracker input device, a concrete implementation of the abstract InputDevice class, is responsible for updating the logical position of the tracked real object. The class diagram for the tracking and registration subsystem is shown in Figure 12.

The use of the ObjectTracker class is divided in two stages, initialization and tracking. In the initialization stage, the init() function, the camera is started and its intrinsic parameters are determined to remove possible lens distortions. An homography matrix is calculated to correct camera perspective, so the four corners of the projection surface are aligned with the corners of the captured image, allowing the camera to be positioned freely as long as all four corners of the projection surface fit in the captured image. The pixels to millimeters ratio is also calculated in this stage to determine the real position of the tracked object on the surface of the ARena.

(a) (b)

Fig. 13. (a) Captured frame; (b) frame after corrections.

To calculate the homography matrix, the four corners of the ARena must be found. To make this search easier, in the current configuration of the ARena, four red stickers were positioned on the corners and are segmented using a threshold in the red channel. These points represent the origin for the calculation of the matrix. The destinations are the corners of the frame captured by the camera and depend on its resolution. These points are also used to determine the pixels to millimeters ratio, since the real distance between them is known. Figure 13 (a) shows the original image captured by the camera, and in (b) is the image after the lens and homography corrections.

Assuming the camera is in a fixed position, these parameters do not change. Therefore, the ObjectTracker class offers a method, saveParams(), which saves these parameters and can later be passed to the ObjectTracker's constructor.

In the tracking stage, the orientation and the X and Y positions of the tracked object are determined using the calcNewPosition() method and then are read using the getX(), getY() and getOrientation() methods. The approach adopted for tracking is color segmentation, thus the use of the Lego color markers. In the technique described by Van den Bergh and Lalioti [1999], the blue channel in a RGB image is usually segmented. The Lego bricks used, however, do not show a satisfactorily pure blue, so the yellow bricks were used and the image was converted to the CMY system so that the yellow channel could be easily segmented.

After the marker colors are segmented, two connect components are found and, from their centers, the position and orientation are determined. The position is simply the average of the centers of the connect components, while the orientation is obtained from the angle of the vector that goes from the center of the smaller Lego brick to the center of the larger one. These relative sizes can be easily determined by counting the number of pixels in each connect component.

The ObjectTracker class is instanced once in the Tracker class and runs the tracking calculations, as well as the update of the values of its sensors in a separate thread.

Real elements. The real elements subsystem consists of classes that, inheriting from enJine's GameObject or Updater core classes, specify the characteristics of game objects that have a physical representation. Figure 14 shows the class diagram for this subsystem.

The RealObject class inherits from GameObject and basically implements the attributes and methods needed to define and manipulate a real object,

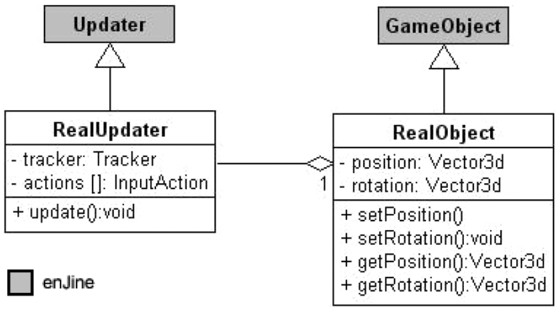

Fig. 14. Class diagram for the real elements subsystem.

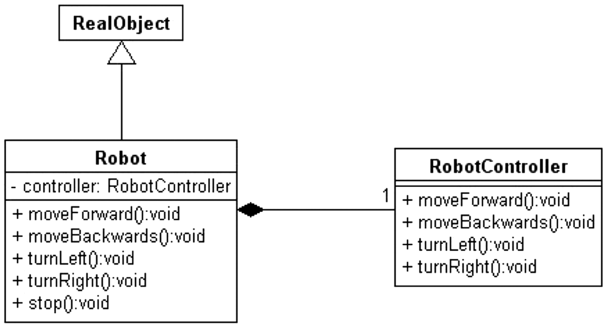

Fig. 15. Class diagram for robot control and communication.

whether it is being tracked or not. It has, therefore, attributes such as position and rotation that define its posture e methods to read and update these attributes.

The RealUpdater class is a specialization of enJine's Updater class and has as an attribute a tracker input device and the InputActions that, during the execution of the updater() method, are updated by tracker's InputSensors. The InputActions, in turn, update the position and orientation of RealObject. Other tracking input devices may be used to update RealObject's position, not only the tracker class described here.

Robot control and communication. The robot control and communication subsystem was implemented in the RobotController class which is part of the Robot class. The Robot class is a specialization of RealObject and has methods to control the robot's movements. RobotController is an abstract class and must be implemented according to the technology used to build the robot.

For the robot created with Lego NXT, the implementation of this class uses the ICommand API, which initiates and controls the communication with the robot through a Bluetooth connection using the BlueCove and RXTX APIs. Figure 15 shows the class diagram for this subsystem.

In this case, the concrete implementation of RobotController allows, through ICommand classes, the control of the robot's basic movements, starting or stopping its engines. It is possible to control the speed of each engine, the number of revolutions, play sound clips, and show images in NXT's screen. Due to a

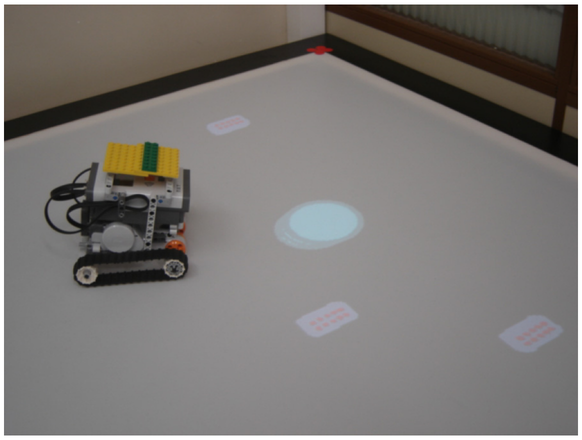

Fig. 16. Robot ARena proof of concept.

limitation of the Lego NXT system, it is only possible to communicate with one robot at a time.

This subsystem allows the player, as well as virtual elements of the augmented environment, to send movement commands to the robot, interacting with it.

4.4 Proof of Concept and Tests

As a test to this infrastructure, a game prototype was proposed, serving as a proof of concept to explore and tune Robot ARena's capabilities. While this first prototype has very simple game rules, it offers the necessary environment for the robot to interact with virtual elements present in the game.

The basic game mechanic is to move the real robot so it can collect some projected virtual batteries. As obstacles there are some barrels that block the robot's passage. If the robot collides with a barrel, it may not move forward and has to backtrack and go around the obstacle.

The virtual models for the game were created with a 3D modeling software and are simple polygonal objects with textures applied to them. All of the interactions between these virtual elements and the tracked real robot were handled by the collision detection and handling system provided by enJine.

Figure 16 shows an image of the game running, with the real robot moving towards the obstacle.

This proof of concept and the tests with it were implemented and run on a computer with the following configuration: Intel Core 2 Quad Q6600 2.4 GHz with 4 GB of RAM and a NVidia GeForce 7300LE card with 256MB of dedicated memory.

The tracking system allowed the determination of the position and orientation of a real object with an update rate that allows real time interaction. Using a resolution of 640 × 480 pixels, the tracking algorithm running in isolation ran at a rate of 10 times per second (i.e., the total execution time was approximately 100 ms). Out of this total time, 40 ms were spent on correcting the image obtained from the camera, the homography and lens corrections. The precision obtained with this resolution is approximately 4mm2.

Using the color segmentation algorithm based on the techniques described in Van den Bergh and Lalioti [1999] instead, and using only one color, resulted in a noticeable reduction in the jittering when using the marker with two colors [Calife et al. 2006] and a reduction in the influence of ambient light. Thus eliminating the main reason for using an infrared marker, which, as discussed previously, proved to be difficult to implement given the differences in infrared sensibility in the tested webcams, whose infrared blockers are difficult to remove and could cause damage to the camera lens.

The performance of the entire system, including the 3D rendering process, collision detection, the tracking system, and the communication with the robot remained stable at 60 frames per second. In fact, this frame rate was limited by the projector's refresh rate of 60 Hz or it would have been higher. This frame rate was not limited by the refresh rate of the slower tracking system because they were implemented in separated threads.

4.5 Analysis of Results and Considerations

Considering the objectives set for this infrastructure, the results were satisfactory and promising regarding the techniques implemented in this configuration of Robot ARena.

The projection calibration for this proof of concept was done automatically and made the positioning of the projector by the user when assembling the system simpler, since even if the projection was not perfectly aligned with the surface, the automatic calibration process would correct it. Because a fixed configuration was used for this proof of concept, CalibrationState was implemented as an independent application, run only once until the ARena configuration was changed. The calibration parameters were saved to and loaded from an XML file.

OpenCV was an important tool, supplying a layer of abstraction for communication with the camera and fast functions for image processing and computer vision, such as those used in the homography calculation.

The robot built using Lego Mindstorms NXT, along with the ICommand API, has flexibility and controllability that allow a deeper level of interaction with the user. With a fast Bluetooth connection, the response time between sending a command and the robot's movement was imperceptible, and thus difficult to measure exactly.

EnJine proved an adequate choice for the core of this infrastructure, simplifying the creation if the virtual environment, the integration between the different subsystems, and the development of the game mechanics.

Based on the successful implementation of this configuration of Robot ARena and the results described here, two more game prototypes were developed.

5. OTHER GAME PROTOTYPES

As further implementation tests for the platform, two game prototypes were developed, both exploring the characteristics of the Robot ARena:

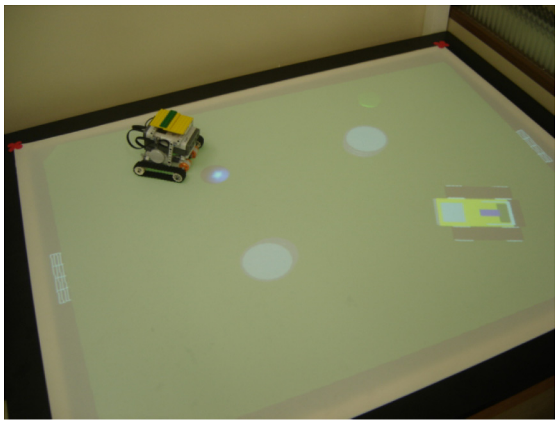

Fig. 17. *FootBot ARena* prototype running.

The first one, *FootBot ARena*, explores the interactions between real and virtual elements. The second game, *TanSpace*, implements the use of a tangible interface, where the manipulation of a physical element is used as a player input

5.1 FootBot ARena

The goal of the development of *FootBot ARena*'s prototype was to explore the interaction between real and virtual elements.

The game consists of a "soccer match" between two robots, one real and one virtual, playing with a virtual ball on a field with virtual obstacles. The first robot to score three goals wins the match. While there are similarities with real robot soccer competitions [Vicentim et al. 2007], this game did not aim to simulate or duplicate their characteristics.

As incremental features of the playability for this match between robots, some items are provided during the matches that may be collected by a robot, giving it abilities to help win the game. These items are the following:

- *Bomb*: this item allows the robot to launch a bomb, which bounces all over the field until it hits a robot (it may even hit the robot who launched it), and when that happens the robot that is hit is paralyzed for 5 seconds.
- *Ray*: allows the robot that captures it to shoot a ray in the direction it faces to hit an opposing robot. If, by the way, a robot is hit three times, it is paralyzed for 10 seconds.
- *Shield*: protects the robot against rays and bombs for 20 seconds.

The obstacles, also present on the playing field, not only block the robots' passage, but also make the ball bounce if it hits the obstacles, making "kicking" them into the opponent's goal more difficult. The ball also bounces against the edges of the playing field.

Figure 17 shows *FootBot ARena* running and the representation of the virtual robot; the figure shows the real robot carrying the virtual ball, the obstacles, and the goals.

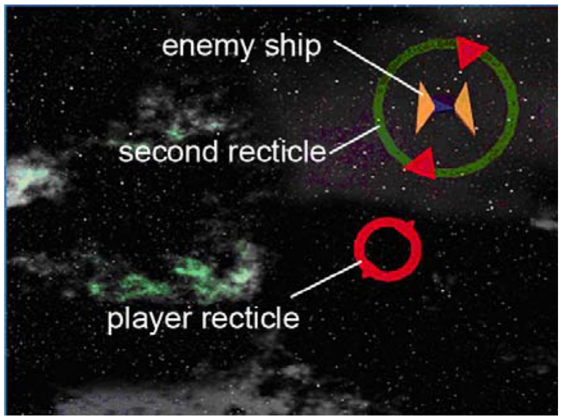

Fig. 18. Screen shot of a *TanSpace* prototype.

As seen in the results of the proof of concept, the interaction between real and virtual elements was done successfully, giving the impression that both coexist. In *FootBot ARena,* the fact that the behavior of both robots, real and virtual, is the same, especially in their interaction with items, obstacles, and with the ball, enhance this impression.

Because the main objective of this prototype was to test the interactions between virtual and real elements, further tests were done to make the interaction more active. For instance, the virtual robot should be able to push the real robot around the field. However, the results of these tests were not satisfactory, mainly due to the Lego robot's configuration, which only allowed back and forward movements and rotations around its central vertical axis. So if the virtual robot happened to push it in a sideways direction, it was impossible for the Lego Robot to move there. However, this limitation is intrinsic to this robot, and if another robot that could move in any direction (such as the Kobito tea caddy) had been used, the system would be well prepared to detect and handle any collisions between the real and virtual, and the real object could move in any direction, using enJine's collision detection and treatment to send the proper commands to the robot.

5.2 TanSpace

The main goal of this game was to explore a TUI (tangible user interface) by means of a tracked object in Robot ARena that interacts with projected content.

The game is set in space and the player controls a spaceship that must shoot down enemy ships. To do so he controls the "aiming recticle of his ship's main gun" by dragging it (physically dragging a real object) on the "combat window" projected on the ARena. He must position the recticle on the enemy ship and align it properly with a second recticle automatically placed on the enemy (by aligning a couple of triangles in each recticle). Only then will the gun shoot and destroy the enemy ship. Figure 18 shows the game's space environment, the recticle controlled by the player, the enemy ship that must be shot down, and the second recticle placed around it.

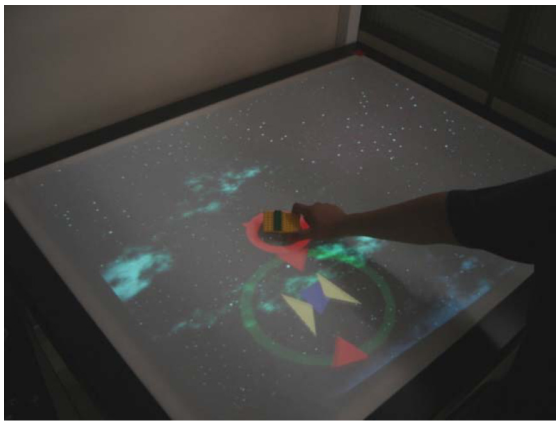

Fig. 19. *FootBot ARena* prototype running in Robot ARena.

The TUI device allows the positioning of the recticle by the player in a rather natural way, enabling him or her to move the recticle freely to match and follow the position of the enemy ship. But since one of the main ideas in this game is to free the user from interacting only through mouse and keyboard, we chose to avoid the use of an active element on the tangible device (such as a button) but to align the two recticles to fire a shot. This had the added benefit of turning a game that would otherwise be somewhat trivial to play into a more challenging one, since coordinating both movements to follow the enemy ship and rotating the tangible device to align the recticles (each a trivial task by itself, even though the ship's movements and the second recticle's rotation appear random) becomes more challenging than it appears at first, making the game more fun. This device also frees the player from having to use a mouse and keyboard, allowing a more natural and simple interaction. Figure 19 shows the game being executed in Robot ARena.

During the tests with *TanSpace,* the tracked device which functions as the tangible element in this game, was often occluded to the top- view camera by the players own hand, at least until the players learned to avoid doing so. This was partly due to the shape of the device, but caused errors in the calculation of its position and orientation. In most cases, this obstruction occurred while the players were rotating the device to align the recticles and destroy the enemy ship.

5.3 Results Analysis and Considerations

Both of these later prototypes met with problems. In *FootBot ARena*, a possible solution to the problem of the virtual robot not being able to push the real one in certain directions is to have a more elaborate system to treat collisions, so that virtual forces applied to the real robot's sides enable it to rotate instead of making only lateral movements.

For *TanSpace*, a possible solution to the marker's occlusion is to change the shape of the tangible device by placing the marker on a taller base, which would lead the player to manipulate the base instead of the marker itself, greatly reducing the possibility of the marker being occluded.

FootBot ARena could run exactly as it does now if it used two virtual robots instead of a real one and a virtual one. The player's experience, however, would be considerably richer if he could build or modify his own robot and see it functioning in a more ample space with real elements, instead of simply inside the computer monitor.

T*anSpace*, in turn, has a unique playability, due to the combination of two kinds of interfaces: horizontal and tangible, allowing the player to interact with the game directly on the table, and recalling the interaction in traditional table games.

Returning to Nilsen et al. [2004], it may be verified that several aspects of traditional and electronic games were profitably combined in both game prototypes presented here, with the calculations, rules, resolution, and presentation of an environment augmented by computer graphics all provided by the computer, while still maintaining the more physical and social aspects of traditional games.

The social aspect could be explored further in *TanSpace* if the tracking and registration subsystem is used to track two tangible devices simultaneously, allowing cooperative and competitive modes in the game.

The main result derived from the development of these prototypes is the conclusion that the Robot ARena infrastructure provides the functionality needed for the development and testing of different kinds of games, exploring new interaction technologies based on spatial augmented reality.

6. CONCLUSIONS

The main contribution of this work is Robot ARena itself, an infrastructure for the development of games and other interactive applications on a horizontal interface, using spatial augmented reality for visualizing the augmented environment, thus freeing the user from any devices that attach to his body.

Another important contribution is provision for the use of tele-operated objects, a robot in the cases described here, ensuring that not only real elements will affect virtual ones, but also the opposite. This feature reinforces the sensation that the virtual and real elements coexist.

The use of enJine as the core of the software system insured flexibility and expansibility to Robot Arena; these features are prominent in this game engine and guide its development. Robot ARena, in turn, expands on the didactic aspects of the enJine when it allows, for instance, the practical hands-on study of subjects such as computer vision and robotics. Finally, Robot ARena confirms enJine's vallue as an adequate platform to test new interactive technologies.

Not only may Robot ARena and enJine be used as teaching tools in and of themselves, they may also be used to develop games with educational objectives that make use of the innovations brought by augmented reality. Other possible applications in the area of "serious games" include its use in marketing, research or as an expressive tool for artists.

An important result obtained through the analysis of the game prototypes with Robot ARena is that the mere integration of elements of augmented reality in a game does not necessarily bring innovation to gameplay, but that the new forms of interface made possible by AR do allow the use of the best aspects of both electronic games and traditional games, together.

Finally, the work described here represents only the beginning of the development and exploration of Robot Arena. It provides a flexible infrastructure for many other experiments and research on using horizontal interfaces, spatial augmented reality, robots, tangible interfaces, and games.

6.1 Future Work

An interesting possibility raised by *FootBot ARena* is the development of applications for remote users connected over a network in a match where each user controls his own (possibly customized) real robot that interacts with a virtual representation of its remote adversary robot. This virtual representation may be a 3D model, but may also be a video avatar.

The development and exploration of other input devices in Robot ARena is another fertile field for future work. Two examples in particular can be described: The first example is the development of a multitouch interaction system, a rather common feature in horizontal interfaces that involves research both in the detection and recognition of the user's touch and in the treatment of this input by applications or by the operating system. In Robot ARena, this mutitouch system may be developed through the recognition of shadows on the projection surface or via the technique of internal reflection of infrared light in acrylic, as in Han [2005]. The second example is to use the recognition of user hand gestures and hand tracking as a form of interface, as in the work using enJine to implement a 3D desktop with gesture-based interaction [Vicentim et al. 2007].

To increase realism or immersion, it is possible to use stereoscopic projections to increase the tridimensional perception of the scene by users. But this effect depends on the user's point of view, which must be tracked while the user is free to move around. This complicates the simultaneous use of the system by multiple users.

Robot ARena still allows the use of projections to apply textures directly onto real objects, using the geometric projection techniques presented by Bimber and Raskar [2005], thus altering the object's appearance in real time. Examples of this application include changing a real robot's color and appearance to reflect the team it belongs to or its internal status or to add special effects such as blinking lights, moving flames, and so on. Another possibility is directing the illumination to specific real objects, allowing them to appear as bright in a dark environment as their virtual counterparts.

Finally, there is important future work in the development of games and applications with educational objectives and the analysis of these applications to verify their efficiency as a teaching tool, as well as the efficiency of Robot ARena in facilitating the development of such applications. The authors believe Robot ARena might have potential in this area, given that some of its elements, namely enJine and Lego, have proven themselves as tools for this purpose.

ACKNOWLEDGMENTS

The authors would like to thank all the Interlab students, researchers, and professors. Daniel Calife also thanks CNPq and ItauCultural's Rumos Arte Cibernética for their financial support.

REFERENCES

AOKI, T., ICHIKAWA, H., ASANO, K., MITAKE, H., IIO, Y., AYUKAWA, R., KURIYAMA, T., KAWA, T., MATSUMURA, I., MATSUSHITA, T., TOYAMA, T., HASEGAWA, S., AND SATO, M. 2005. Kobito—Virtual Brownie—Virtual creatures interact with real objects and real people. *SIGGRAPH'2005 Emerging Technologies.*

BERNARDES, J.L., JR., DIAS, J.S., AND TORI, R. 2005. Exploring mixed reality user interfaces for electronic games. In *Anais do IV Workshop Brasileiro de Jogos e Entretenimento Digital* (WJOGOS'2005), 353–358.

BIMBER, O. AND RASKAR, R. 2005. *Spatial Augmented Reality: Merging Real And Virtual Worlds*. A K Peters.

CALIFE, D., BERNARDES, J.L., JR., AND TORI, R. 2007a. Robot ARena: An augmented reality platform for game development. In *Proceedings of the VI Brazilian Symposium on Computer Games and Digital Entertainment* (SBGAMES'2007), 77–86.

CALIFE, D., TOMOYOSE, A., SPINOLA, D., BERNARDES, J.L. JR., AND TORI, R. 2007b. Robot ARena: Infraestructure for applications involving spatial augmented reality and robots. In *Proceedings of the IX Symposium on Virtual and Augmented Reality* (SVR'2007).

CALIFE, D., TOMOYOSE, A., SPINOLA, D.. AND TORI, R. 2006. Controle e Rastreamento de um Robô Real para um Ambiente de Realidade Misturada. In *Anais do II Workshop de Aplicações de Realidade Virtual* (WARV'2006).

GEIGER, C., REIMANN, C., STICKLEIN, J., AND PAELKE, E.V. 2002. JARTOOLKIT—A Java binding for ARToolKit. In *Proceedings of the First IEEE International Workshop on Augmented Reality Toolkit*.

HAN, J.Y. 2005. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology* (UIST'05), ACM, New York, 115–118.

ISHII, H. AND ULLMER, B. 1997. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, 234–241.

KOJIMA, M., SUGIMOTO, M., NAKAMURA, A., TOMITA, M., NII, H. AND INAMI, M. 2006. Augmented Coliseum: An augmented game environment with small vehicles. In the *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*.

LEWIS, M., JACOBSON, F. 2002. Game Engines in Scientific Research. *Communications of the ACM 45*, 1, 27–31.

MACWILLIAMS, A., SANDOR, C., WAGNER, M., BAUER, M., KLINKER, G., AND BRUEGGE, B. 2003. Herding sheep: Live system development for distributed augmented reality. In *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality* (ISMAR'2003), 123–132.

MAGERKUTH, C., ENGELKE, T., AND MEMISOGLU, M. 2004. Augmenting the virtual domain with physical and social elements: Towards a paradigm shift in computer entertainment technology. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACM, New York, 163–172.

NAKAMURA, R., BERNARDES J.L., JR., AND TORI, R. 2006. Using a didactic game engine to teach computer science. In *Proceedings of the Digital V Brazilian Symposium on Computer Games and Digital Entertainment* (SBGAMES'2006).

NILSEN, T., LINTON, S., AND LOOSER, J. 2004. Motivations for augmented reality gaming. In *Proceedings of the New Zealand Game Developers Conference* (FUSE'04), 86–93.

SHEN, C., VERNIER, F.D., FORLINES, C., AND RINGEL, M. 2004. DiamondSpin: An extensible toolkit for around the table interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI'2004), ACM, New York, 167–174.

TORI, R., BERNARDES, J.L., JR., AND NAKAMURA, R. 2006. Teaching introductory computer graphics using Java 3D, games and customized software: A Brazilian experience. *International Conference on Computer Graphics and Interactive Techniques*, ACM SIGGRAPH'2006 Educators Program.

TSUDA, F., HOKAMA, P.M., RODRIGUES, T.M., AND BERNARDES, J.L., JR. 2007. Integration of jARToolKit and enJine: Extending with AR the potential use of a didactic game engine. In *Proceedings of the IX Symposium on Virtual and Augmented Reality* (SVR'2007).

VAN DEN BERGH, F. AND LALIOTI, V. 1999. Software chroma keying in an immersive virtual environment. *South African Computer J. 24*, 155–162.

VICENTIM, A.P., TOKUNAGA, D.M., AND MUTO, M.M. 2007. Desenvolvimento e Estudo de um Desktop3D com o Uso de Gestos. Escola Politécnica, Universidade de São Paulo, São Paulo.

WILSON, A.D. 2005. Play Anywhere: A compact interactive tabletop projection-vision system. In *Proceedings of the 18th Annual ACM Symposium on User interface Software and Technology* (UIST'05), ACM, New York, 83–92.

Received November 2007; revised October 2008; accepted October 2008