Integrating Active Tangible Devices with a Synthetic Environment for Collaborative Engineering

Sandy Ressler Brian Antonishek
Qiming Wang Afzal Godil

{sressler | antonis | qwang | godil}@nist.gov Information Technology Laboratory
National Institute of Standards and Technology

Abstract

This paper describes the creation of an environment for collaborative engineering in which the goal is to improve the user interface by using haptic manipulation with synthetic environments. We have integrated a multiuser synthetic environment with physical robotic devices to create a work environment. These devices can move under computer control or may be manipulated directly by the user. The work environment represents objects from the application domain such as a building construction environment or manufacturing cell. Collaborating engineers can discuss object interactions, such as crane planning or building placement using this environment. A physical representation of a work environment enables the user to perform direct, tangible manipulations of the devices which are mirrored in the synthetic environment. The direct physical manipulation of robotic devices offers the users a natural and efficient method of interacting with the synthetic environment.

CR Categories and Subject Descriptors: I.3.2 [Computer Graphics]: Graphic Systems - Distributed/network graphics; I.3.6 [Computer Graphics]: Methodology and Techniques: Interaction techniques, standards; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Virtual Reality; J.6 [Computer Applications]: Computer Aided Engineering - Computer-aided design (CAD).

Additional Keywords and Phrases: virtual environments, user interfaces, device control, tangible reality, VRML

* National Institute of Standards and Technology
100 Bureau Drive STOP 8940
Gaithersburg MD 20899-8940

1 Introduction and Related Work

We have developed a collaborative engineering environment based on the integration of a multi-user synthetic environment with tangible devices. By tangible we refer to the ability to pick up and interact with actual physical objects present in the interface. In this work environment the objects a user can manipulate are active and can be used both for feedback and control of elements in a synthetic environment. The interface was implemented in a multi-user environment which was created with off-the-shelf commercial software. A Java based mediation system forms the central hub through which object position updates and device interface commands flow. The Java hub integrates the Virtual Reality Modeling Language (VRML) synthetic environment with the robotic devices in the actual physical environment.

The use of virtual environments (VE) for collaborative engineering has been a topic of research for quite a few years. A notable set of workshops on Collaborative Virtual Environments has been held since 1996 [3]. However most of the CVE research focused on the collaborative and social nature of interaction as opposed to engineering applications. Our immediate goal was to determine the feasibility of using a tangible interface with a multiuser VRML environment as applied to collaborative engineering. The use of VRML, the only ISO standard for 3D graphics on the Web allows us to provide software to the public. An additional goal was to use as much off-the-shelf software and hardware as possible in order to facilitate transfer of the technology to the commercial world. We have created a multi-user VE utilizing a commercial system, the blaxxun Community Platform, along with commercial off-the-shelf configurable robots, LEGO Mindstorms robotics invention system, illustrated in Figure 1 [13].

There has been significant research performed on the use of tangible devices and other relevant haptic interfaces in the literature.The concepts of Tangible Reality interfaces have been developed initially as a “graspable interface” [6] and continue to the present day in Ishii’s Tangible Bits project [9]. Our work differs from these systems by incorporating a multiuser system that includes active haptic devices which can be used both as control and for feedback. The addition of active devices enables us to create a “mirror world” in which the actions in the virtual world are mimiced in the real world and visa versa. The concept of a “mirror world” that can be used for industrial applications is a modification of Gelernter’s [7] concept of Mirror Worlds.

2 System Overview

The overall environment is conceptually simple. For example, two collaborating engineers in geographically separate areas wish to manipulate and discuss a construction project. Previous work by our team has demonstrated complex integration of a VRML in a manufacturing environment, generated from several sources including near real time dynamics[21]. Recent work at NIST [14] has demonstrated that VRML can be used to represent rich construction environments. However, manipulation of elements such as a virtual excavator is awkward. Control panels with many buttons and sliders are functional but difficult to manipulate. We "mirror" the physical environment with the synthetic environment in order to present the user with a consistent view of the work environment. We believe direct haptic manipulation can lead to more intuitive interaction. Users move the tangible excavator and adjust the rotatable arm and this causes the virtual equivalent to update.

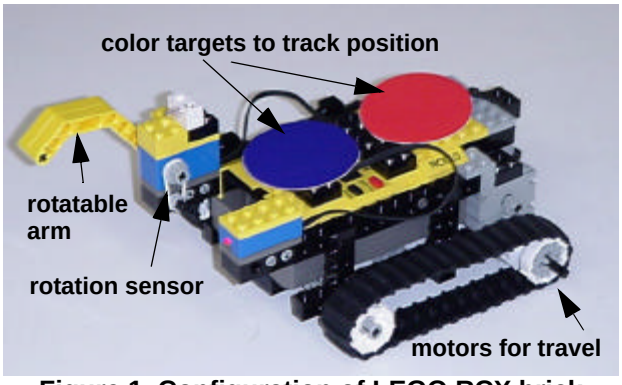


Figure 1 Configuration of LEGO RCX brick

We configured the LEGO robot with left and right motors for travel. Each motor can be addressed independently allowing it to turn. The RCX brick had a build in speaker. We also added a light emitter and a rotation sensor. Each sensor and emitter is independently addressable.

The core of the system is the Java based Virtual Environment Device Integration server, JVEDI [10], which mediates between the virtual environment and the users and system input. The JVEDI architecture illustrated in Figure 2, allows for a loose integration of systems via sockets, file and serial ports for the physical devices. The server runs as a stand-alone Java application on the host computer.

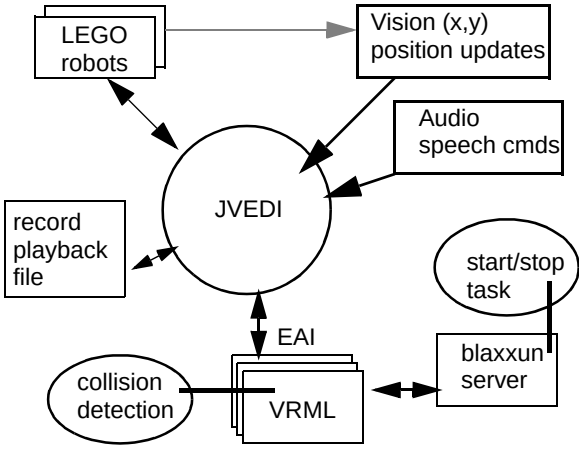


Figure 2 System Architecture

Communications between JVEDI and the LEGO robots occurs via a serial interface to an infrared (IR) transceiver. The vision system writes position data into a file which is read by the JVEDI process. Audio commands are sent via a socket interface. The VRML to blaxxun server communication is via sockets using off-the-shelf product configurations.

The environment consists of two work surfaces, one on the left side of the lab, and the other on the right side of the lab. We can easily posit that these work surfaces are in different areas of the country, ignoring of course the obvious reality of network lag. Each work surface is populated with a single LEGO Mindstorm robot which represents the equipment of interest in the VE. Multiple robots per work surface are also possible. The VE, a multi-user blaxxun environment [1] displays a single virtual work area consisting of the sum of all objects of interest in the real work areas. Engineers at each working area can view the VE and see the entire collaborative area. For our testing purposes, we used a single display of the VE as the two engineers were collocated, however we can view the VE on any system with access to the Web inside our firewall.

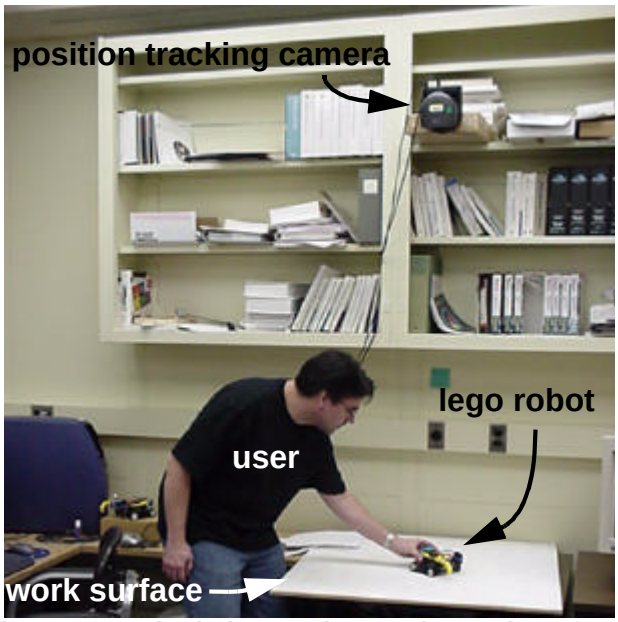


Figure 3 Manipulating a robot on the work surface

Users wishing to collaborate, sit or stand as illustrated in Figure 3, in front of the worksurface and a corresponding computer displaying the multi-user environment. Figure 4 illustrates a typical display of a two person collaborative virtual environment.

Above each work surface we place a video camera looking down at the surface. With a computer vision program this camera provides 2D position, and orientation data of the robot on the surface to the Java server. When users move the robots on the work surface the position is sent to the Java (JVEDI) server which relays the information to the multi-user VE and updates the display.

Figure 4 Collaborative virtual environment with two participants.

In addition to moving the robots haptically, we provide a modest user interface consisting of buttons and arrows in the VE. This interface allows an engineer without access to the tangible interface to remotely move objects, whose position is again reflected in the VE. The VE and the real environment are kept synchronized. They are mirrors of each other. Synchronization was accomplished by always taking the position from the values reported by the video system. When a user clicked on a forward button in the VE, the system causes the LEGO robot to move forward and the video system reports a new position. The new position is then used in the VE. Note that if the position was directly used to control the objects in the VE that the real world and VE would quickly get out of sync.

3 Integration Issues

Our most challenging aspect in creating the work environment was the integration of all the processing elements. VRML browsers only have one common method for interfacing with programs and devices in the real world, a method called the External Authoring Interface (EAI). The EAI provides a loosely coupled integration mechanism upon which we built functionality such as controlling robots, reading and writing data from a position tracking device, and the ability to read and write data files.

A fully configured version of the environment requires up to six separate computers. Each of the two working areas requires a computer for the video processing and one for the VE. A fifth computer is used for the JVEDI processing. Finally a sixth computer is used to provide an overview of the combined VE. Of course, we could combine the video processing and the VE display for a workstation onto a single computer with a corresponding impact on performance. The overview VE computer is also not strictly necessary as each of the VE displays for the work surfaces is equivalent. Running each of these processing elements on separate computers illustrates our design goal to increase the scalable and distributed nature of the architecture.

Communications between each of the computers was performed via the existing laboratory ethernet. Communications to the robot devices is via serial ports to the infrared devices which perform the low level communication to the RCX bricks. This system of loosely connected computers and serial port devices is fairly fragile. The distributed nature of the system is robust in that it is easy to replace any component.

4 Major Components

The major components: Java-based device interfacing; vision processing; LEGO device interfacing, and multi-user VRML, are described below.

4.1 The JVEDI Java Hub

The Java hub which provides the interface between system components has been packaged up as a software package we call JVEDI (Java based Virtual Environment Device Integration). JVEDI had its origin in some earlier work [20] at NIST where we used similar but less robust Java code to integrate a Sony remote control camera and a 6 degree of freedom position tracker with VRML worlds. Integration of these devices with the new code would require relatively minor changes primarily concerning the specific device protocols.

We created a set of Java classes which function as interfaces between the three primary processes, the clients which request and consume services (such as position information), the server (JVEDI) which manages interactions, and the service processors which talk at a low level to the specific devices (LEGO, serial ports, files). The Java classes are as follows:

- **LegoBase.java:** Contains all the commands shared by both the client and server.
- **LegoClient.java:** (subclass of LegoBase) The client-side class of the LEGO controller program. Client applications create a LegoClient instance in order to control or monitor the LEGO RCX.
- **LegoRequestProcessor.java:** (subclass of LegoBase) Keeps a socket open to the requesting client. It acts upon a client's command or inquiry, and returns information to the client.
- **LegoServer.java:** The server-side class of the LEGO controller program. It receives a client connection and creates a LegoRequestProcessor thread to handle the new client's requests.
- **LegoCommandSet.java:** Contains all the byte-code commands that the LEGO RCX understands.
- **LegoRCX.java:** (subclass of LegoCommandSet) Contains the higher-level Java routines used for communicating with the LEGO RCX.

- **SerialIO.java:** Contains all serial port I/O related procedures.

As an example here is the order of execution for the action of a pushing a position button in the multiuser VRML world, causing the LEGO to move.

- **Initialization:** The LegoServer process starts. It checks to see how many LegoRCX instances are running. If there are no LegoRCX instances running an error is reported, indicating that the physical LEGO RCX is most likely not turned on. (Note: The vision system is started or is already running but is separate from these Java processes.) A LegoClient process starts when the multiuser VRML world is started. When a new client starts it connects to the LegoServer process via a socket. The LegoServer forks off a new LegoRequestProcessor instance.
- **Step 1:** The LegoRequestProcessor reads the client's command and sends the LegoRCX instance a new command which it, in turn, sends as a low level command to the physical LEGO via the IR interface. The LEGO RCX moves.
- **Step 2:** At this point the vision system, which is constantly updating a file with the x,y position and orientation of the LEGO, is used. The LegoRequestProcessor examines the x,y file for position and orientation changes. Upon detecting a change it returns the new position to the requesting client.
- **Step 3:** The VRML LEGO's position is now updated. Note that the position of the VRML LEGO is updated according to the change in position of the physical LEGO. This ensures that the physical and virtual LEGOs remain synchronized. It does however introduce a slight delay for the user in the update speed of the VRML LEGO's position. Steps 1-3 are repeated for each press of the position button.
- **Shutdown:** When the client closes the connection to the LegoRequestProcessor it, the LegoRequestProcessor exits. The client connection generally remains open for as long as the user has selected the LEGO for control.

4.2 Vision Processing, Speech Input

The position and orientation of the LEGO robots are computed in real time using a computer vision method based on color tracking. The vision program uses an inexpensive camera and can track multiple robots at 10 frames/sec. Position data is sent to the JVEDI server and then to the VRML world which updates the virtual LEGO, also in real time.

Computer vision algorithms that are used for real time tracking must be fast and efficient. Many methods for computer vision exist [2][12][18][8][15] and are computationally expensive. We decided to use a fast, simple method based on color tracking [5]. To reduce noise in the data we used a Kalman filter [11] smoothing method.

To track the LEGO robots, we pasted two colored circular cards with different colors on the robot. The computer vision program is calibrated with these colors and uses color probability distribution to find the center of the two squares. The position is the mean of the two centers of the squares. The orientation is calculated using the arctangent of the difference of the centers. The equations used are illustrated in Figure 5.

A. Calculate the zeroth moment for the two probability distributions (only one is shown).

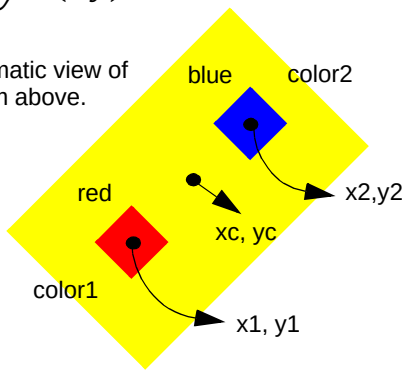
$$M_{00}^{1} = \sum_x \sum_y I^{1}(x,y) \quad (color1 = red)$$

B. Calculate the first moment.

$$M_{10}^{1} = \sum_x \sum_y x I^{1}(x,y)$$

$$M_{01}^{1} = \sum_x \sum_y y I^{1}(x,y)$$

Diagrammatic view of robot from above.



C. Compute the center of the red square

$$x_1 = \frac{M_{10}^{1}}{M_{00}^{1}} \qquad y_1 = \frac{M_{01}^{1}}{M_{00}^{1}}$$

The centroid xc, yc, is the average of the positions for each color.

D. The orientation is calculated as:

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

Where I1(x,y) is the probability of the first square color (red in this example) and I2(x,y) is the probability of the second square color (blue in this example).

Figure 5 Centroid and orientation calculations

The processing system views the robot and displays (for debugging purposes) red and green marks over the colored targets on the video display. In addition, the centroid is marked with a small crosshair shown in Figure 6.

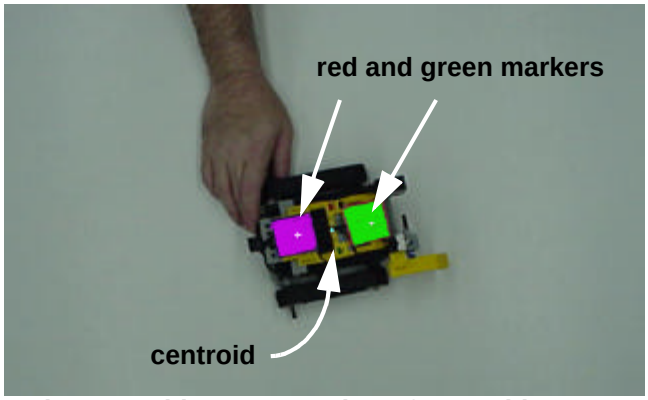


Figure 6 Video camera view after position processing, note centroid position

A user who is moving robots cannot easily access a keyboard. Also, during development, it was often necessary to move robots on different surfaces simultaneously, which is difficult. This prompted us to implement a simple speech system to allow the user to select different virtual geometries while remaining with the work surface and robot. We implemented a crude speech user interface utilizing the Microsoft speech SDK [16]. The speech processing system consists of a Java applet which recognizes several commands. The user can speak excavator, truck, lego and robot to change geometry. The brick can be selected and moved by speaking the commands “select_red”, “select_blue”, “left”, “right”, “forward”, “backward”. The voice commands were translated into the commands used by the VRML control panel interface and were sent through JVEDI for execution.

4.3 LEGO Interfacing

Adding control for the LEGO RCX robots was very similar to the JVEDI remote control of a camera [20]. We needed to send commands to move the robot in any direction and get feedback from the robot about sensor values and robot status.

The communications was achieved using the LEGO Infrared (IR) tower pictured in Figure 7, which comes with the LEGO Mindstorms Robotics Invention System [13].

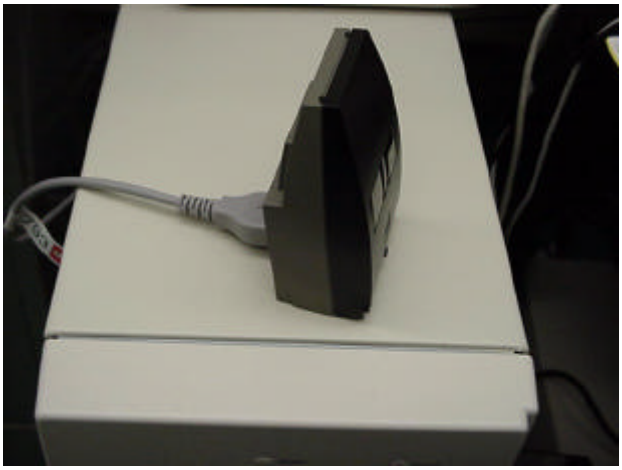


Figure 7 LEGO infrared tower

The IR tower attaches to a computer's serial port and is treated as just another serial port device in JVEDI. The IR tower sends/receives commands to/from the RCX brick.

The position of the RCX brick in the real world is obtained using the vision tracking system described in an earlier section. The RCX's current position data is passed to the JVEDI server which then gives the data to any JVEDI clients requesting position data. The current work environment version uses data files to transfer the information from the vision system to the JVEDI server but will be upgraded to use network sockets to increase performance by eliminating disk I/O lag time.

Addition of the RCX control and monitor commands to the JVEDI server required specific command bytes to be sent and received between the IR tower and the RCX brick. We had a good starting point with the Java RCX Tanks demonstration that was given at a Java One conference [4]. That code proved an effective way of accomplishing the initial communication between Java and the LEGO RCX bricks. There are also some excellent sources on the web for getting detailed information about the RCX internals and the codes that are used for communications [19][17].

Commands (IR signals) are transmitted to the RCX for the desired actions (left, right, etc.). The program running on each of the RCXs receive and interpret these signals. Each of the RCX programs has unique signals for their actions, this way an individual RCX can be controlled while the others are unaffected. Note that communication to the bricks goes to all bricks at the same time because of the broadcast nature of the IR tower.

Internally, the LEGO RCX bricks were programmed so that each individual sensor or effector responded to a unique code. This allows us to control or read data from specific bricks. Each control command, such as moving the left wheel of the red brick, was prefaced with the specific code number for that effector. Internally the bricks each were programmed to recognize and respond only to the codes for itself. Other control mechanisms are, of course possible. However, the primary constraint is the broadcast nature of the commands to the bricks which requires some method of directing commands to the correct brick and effector/sensor.

4.4 Multiuser VRML

The multiuser aspect of our VRML world was accomplished using off-the-shelf commercially available software, blaxxun Community. The VRML world consists of two parts. The first is the virtual world which includes two LEGO bricks (red and blue), the floor, and one cylinder object. (See Figure 4.) The second part is the control panel, illustrated in Figure 8. Each LEGO has its own control panel which allows the user to control the operation of the virtual LEGO and the real LEGO.

The control panel has several buttons. First, the on/off button to turn on or off the connection between the real LEGO brick and the virtual LEGO brick. When it is on, four arrows appear. A user can click the arrows to cause the real LEGO brick to move forward, backward, left, or right. When the LEGO control is off, the arrows disappear.

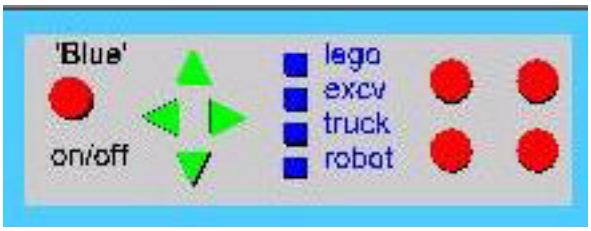


Figure 8 VRML button panel for robot control

Next on the control panel are square buttons connected to a VRML switch for selecting the different virtual device geometries. Finally the four buttons on the right are used for recording the real LEGO positions to a file and playing back the recorded sequence from the file.

The virtual LEGO communicates with the real LEGO through the EAI and JVEDI. The EAI Java class “legotrack” generates LEGO clients which talk to the JVEDI server. Movement of the virtual LEGO is only controlled by the real LEGO's motion. When a user turns on a LEGO from the control panel, “legotrack” creates a Java Thread for the selected LEGO and runs the Thread. The Thread polls the real position information from JVEDI approximately every 10 milliseconds. When it finds the current position is not the same as the previous position, it sends the position data to the VRML world and the virtual LEGO will move to that position. If the user clicks the arrow buttons in the control panel, the commands to move forward, backward, left, or right, will be send to the JVEDI server, and the server will instruct the real LEGO to move.

The geometry and scripts needed to interface the VRML objects with the serial port has been encapsulated into a LEGO Proto. The interface to this EXTERNPROTO is as follows:

```
EXTERNPROTO LEGO [
    field SFVec3f translation
    field SFRotation orientation
    field SFVec3f scale
    field SFColor color
    eventOut SFBool lego_getpos
    eventIn SFVec3f lego_pos
    eventIn SFRotation lego_rot
    eventIn SFBool getpos_isActive
    eventOut SFString legocommand
    eventIn SFBool forward_isActive
    eventIn SFBool backward_isActive
    eventIn SFBool left_isActive
    eventIn SFBool right_isActive
    eventIn SFVec3f tankpos
    eventOut SFRotation tankrot
] “proto.wrl#LEGO”
```

The Script node inside of the LEGO PROTO serves as the interface between the “legotrack” EAI Java applet, the VRML geometry and the user interface. The “legotrack” applet communicates with the serial port. The user interface is encapsulated in the DASH PROTO which is visually instantiated as illustrated in Figure 8.

The VRML world runs under the blaxxun multiuser environment. Several events must be shared. First, the motion of the virtual devices are shared so all participants in the multiuser world can see devices move. We defined the position of the virtual device as a shared event in the blaxxun environment. Second, we want all users to be able to control the device if it is not controlled by another user. This means that when one device is controlled by someone, other users can not control it until the first user gives up his control. We use the Lock shared event in the blaxxun environment to implement this. The shared event with Lock access type, means that as soon as someone sends this event, it cannot be modified by other users until the locking client releases the lock or leaves the place which automatically releases the lock. In our virtual world, the control button's on/off event was set to be a Locked shared event, and a related Script Node was programmed. When a user first turns on the device control, an avatarname is assigned and four arrow buttons show up in his control panel. This user now has the right to click the arrow buttons to control the motion of the real device. At this time, other users should see the button is red, and there are no arrow buttons on their control panel. As soon as the first user turns off the control, the control button of other users become green, and they are allowed to gain control.

The geometry alternatives in the current world are LEGO brick, excavator, dumptruck, and LEGO brick with an arm. The user can select one of them in the control panel. The tank with an arm and excavator are of particular interest. The actual LEGO bricks are configured with a rotation sensor attached to the arm. When the real arm is rotated, the virtual excavator arm moves. This is a particularly fast and effective way of positioning and controlling the virtual excavator which, in an actual application, can perform a digging task.

5 Auxiliary Processing

In addition to the position tracking and updating of events in the VE, other types of calculations are easily implemented. We created three types of processing: collision detection, specific task commands and position recording/playback.

5.1 Collision Detection

Suppose the robot on one work surface virtually collides with the robot on the other work surface. In addition, to visually seeing a collision in the VE we can cause the robot to beep and flash a light. This is quite useful when pushing the robot around on the work surface. The collision detection computations, a simple bounding box algorithm, is performed in the VE (the VRML world). We could have performed the calculations in the JVEDI server, however we also wanted to detect collisions between the robot and other virtual objects unknown to the JVEDI server. For example we place a blue cylinder, representing a construction crane, in the VE and the robot beeped when it collided with the virtual object. Knowledge of this additional object exists only in the VE. It may prove useful to build a mirror, a kind of world model, of the entire VE in the JVEDI server itself to use for these auxiliary processing functions.

5.2 Task Commands

We also created small programmatic tasks that could be executed by the robot. In the blaxxun multiuser environment, we created a LEGO Agent and programmed a script for it. The script can read the text command in the chat area typed by the user, and send specific events to the virtual world. For example, if we type the command “red_square”, the red robot executes a preloaded program to run around the work surface along a square path. The robots would autonomously perform these tasks once given the

starting command because they were preloaded with control programs. If we wanted to increase the repetoire of tasks we could simply load the programs dynamically from the JVEDI server, although we have not implemented this capability to date.

5.3 Recording/Playback

We implemented the ability to record and playback the position of the robots for use in future usability experiments and to increase the utility of the work environment. Several buttons on the control panel illustrated in Figure 8 allow recordings of robot positions. At a later time, such as during analysis of an experiment, we can playback the sequence by reading from the position file. Two types of recordings were enabled, the position of the robot and user clicks on the directional arrows of the control panel. Note that the architecture illustration in Figure 2 is slightly inaccurate as the record/playback capability is currently implemented using a separate Java process and is not part of the JVEDI program, but is planned as a minor upgrade.

6 Summary and Future Work

We have described how we integrated a haptic active device, the LEGO MindStorms robot for use in a collaborative engineering environment. The prototype system contains two work surfaces connected by a local area network. The JVEDI code is available on our project Web site at: http://ovrt.nist.gov/jvedi/ and the source code is in the public domain.

A number of enhancements are needed to improve the usability and utility of the work environment. More robust, less light sensitive color tracking, dynamic loading of LEGO RCX brick programs. More flexible position recording and playback file manipulation is also necessary for the conduct of realistic usability experiments. There is also a need to define specific tasks with metrics to meaningfully conduct evaluations of the system to measure effectiveness.

The tangible nature of the project has yet to be fully exploited. The existing configuration allows for only 2D detection and position tracking. The addition of a second camera perpendicular to the first would allow us to detect the position of items of interest in 3D.

The use of off-the-shelf components and the public availability of the source code makes duplication of the environment relatively straightforward and low cost. The actual arrangement of the camera and the work surface is somewhat awkward and needs to be refined for more robust and simple deployment. It is our intent to work with other engineering projects at NIST such as instrumented construction sites [23] and advanced IT for manufacturing [22].

7 ACKNOWLEDGEMENTS

Thanks to Bob Lipman for allowing us to use his excavator and dumptruck VRML models. We would like to thank the continuing sponsorship of the NIST Systems Integration for Manufacturing Applications (SIMA) program for supporting this work.

References

[1] blaxxun Interactive http://www.blaxxun.com

[2] Cheriet M., Yang Y. H., "Vision Interfaces, real world applications of computer vision". 1999, ISBN 981-02-4109-7

[3] Collaborative Virtual Environments (CVE) Conference reports and updates, http://www.fxpal.xerox.com/ConferencesWorkshops/cve/index.html

[4] Michael Deleo, A Demonstration of JiniTM Technology and the K Virtual Machine Java Tanks demo, http://developer.java.sun.com/developer/technicalArticles/jini/JavaTanks/Javatanks.html

[5] P. Fieguth and D. Terzopoulos, "Color-based tracking of heads and other mobile objects at video frame rates," In Proceedings of IEEE CVPR, pp. 21-27, 1997.

[6] Fitzmaurice, G., Ishii, H., Buxton, B., Bricks: Laying the Foundations for Graspable User Interfaces, Proceedings of CHI'95 May 7-11, 1995, pages 442-449.

[7] Gelernter, D. Mirror Worlds: Or the Day Software Puts the Universe in a Shoebox, 1992.

[8] M. Isard, A. Blake, "Contour tracking by stochastic propagation of conditional density," Proceedings of 4th European Conference On Computer Vision, Cambridge, UK, April 1996

[9] Ishii, H., Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, Proceedings of CHI'97 March 22-27, 1997, pages 234-241.

[10] JVEDI, Java Virtual Environment Device Interfaces, http://ovrt.nist.gov/jvedi/

[11] Kalman filter, http://www.innovatia.com/software/papers/kalman.htm

[12] M. Kass, A. Witkin D.Terzopoulos, "Snakes: Active contour Models," International Journal of Computer Vision (1) #4, pp. 321-331, 1988.

[13] LEGO Mindstorms. http://www.legomindstorms.com/

[14] R. Lipman, K. Reed. Using VRML In Construction Industry Applications *Proceedings Web3D-VRML 2000 Fifth Symposium on the Virtual Reality Modeling Language*, Monterey, California, pages 119-124, 2000.

[15] T. Maurer, and C. von der Malsburg, "Tracking and learning graphs and pose on image sequence of faces," Proceedings Of the Second International Conference On Automatic Face and Gesture Recognition, pp. 176-181, 1996.

[16] Microsoft Speech SDK 4.0 http://www.microsoft.com/speech/

[17] 'Not Quite C' language, http://www.enteract.com/~dbaum/nqc/index.html

[18] A. Pentland, B. Moghaddam, T. Starner, "View-based and Modular Eigenspaces for face recognition," CVPR'94, pp. 84-91, 1994.

[19] RCX Internals, http://graphics.stanford.edu/~kekoa/rcx/

[20] S. Ressler, B. Antonishek, Q. Wang, A. Godil, K. Stouffer. When Worlds Collide - Interactions between the Virtual and the Real. *Proceedings Twente Workshop on Language Technology TWLT15 Interactions in Virtual Worlds*, Enschede, The Netherlands, May 19-21 1999, pages 165-170.

[21] S. Ressler, A. Godil, Q. Wang, G. Seidman. A VRML Integration Methodology for Manufacturing Applications *Proceeding VRML99 Fourth Symposium on the Virtual Reality Modeling Language*, Paderborn Germany, pages 167-172, 1999.

[22] Systems Integration for Manufacturing Applications (SIMA) Program *http://www.nist.gov/sima*

[23] VRML for Construction Site Activities, http://cic.nist.gov/vrml/