INTEGRATION OF VOICE COMMANDS INTO A VIRTUAL REALITY ENVIRONMENT FOR ASSEMBLY DESIGN

Zhao, W. and Madhavan, V.
Department of Industrial and Manufacturing Engineering
Wichita State University
120 Engineering Building
Wichita, Kansas – 67260
Corresponding author’s e-mail: vis.madhavan@wichita.edu

Abstract: This paper addresses integration of voice commands within immersive virtual reality sessions. While wands, 3D mice, etc., are extensively used for navigation and other interaction in 3D environments, voice commands can provide a much more natural means of interaction, especially in applications in which the immersed user’s hands and vision may be otherwise occupied. The IBM WebSphere™ Voice Server SDK has been used to develop a VoiceXML application for recognition of voice commands issued by the immersed user. The Resin® web server is used to broadcast these commands on a TCP port. A client written using the JackScript™ API for Jack reads the commands and executes corresponding functions.

A comprehensive set of voice commands has been developed to provide significant functionality for assembly and maintenance simulations. One of the most important applications is a method by which the immersed human can walk and fly through the virtual environment, thereby greatly expanding the envelope of the virtual environment navigable under the constraints imposed by the limited tracking volume. Voice commands are also used to switch between different modes of interaction (such as grasping between fingers as opposed to using the palm), for capturing specific postures, and for replaying captured posture sequences so the immersed user can evaluate them. Voice commands allow one person to control the virtual environment while immersed in the environment, thereby improving productivity.

1. INTRODUCTION

Product assembly is an interactive process between the operator(s), tools and components and usually involves complex tasks and task sequences. It is necessary to evaluate difficult process sequences in order to have an efficient assembly design. The application of Virtual Reality (VR) to assembly design, especially immersive virtual reality in a complete virtual model of the assembly environment, can help users design and evaluate assembly tasks and different possible sequences, and choose the best alternatives. This is especially important in the current volatile manufacturing market, with frequent changes in the specification of the product, which typically affect the assembly process.

VR has currently found significant applications in the field of manufacturing. Understanding the actual manufacturing capabilities (analyzing assembly process, evaluating design constraints, or training) using virtual environments and digital prototypes can help eliminate need for developing physical prototypes, which is time consuming and expensive. Many industries now use VR as a standard tool in their business processes. One such major area of application is the use of VR for education and training. Loftin (1993) detailed the use of VR by NASA’s Lyndon B. Johnson Space Center to train astronauts to handle equipment in zero gravity for the Hubble Space Telescope Repair and Maintenance missions. VR applications can also be seen in aircraft inspection training (Vora et al., 2002). Proper training is necessary as inspecting is a complex task due to interaction between human and large number of components.

VR is an ideal tool for the modeling and analysis of complex task sequences often found in activities such as assembly and maintenance. Programmable digital humanoids can help to provide a cost efficient method to perform and analyze assembly operations. Lockheed Martin uses digital humanoids to perform assembly process design and as well as for human factor analysis (Wampler et al., 2003). Kibira and McLean (2002) have studied the application of VR to design of manufacturing assembly lines and to identify and analyze discrete assembly events. Successful application of virtual reality to design and validate maintenance guidelines has been a key enabling technology in the Lockheed Martin Tactical Aircraft System (LMTAS) program (Abshire and Barron, 1998). Similarly, weapon equipment maintenance is very essential as this equipment is very expensive and new equipment is introduced often. A virtual assembly system for equipment maintenance VASMET was developed by Chen et al. (2003).

The use of immersive virtual reality to evaluate assembly sequences in manufacturing environments will benefit from more sophisticated interaction of the human with the virtual environment, along with improved feedback. This is especially important since the immersed operator’s hands and vision may be occupied with the assembly tasks. Use of voice commands and auditory feedback could potentially play a vital role in assembly environments.
Wan et al. (2004) developed MIVAS (a CAVE based Multi-modal Immersive Virtual Assembly System) which is the integration of physical elements like tracking devices, data gloves etc. and interacting functionalities such as voice commands and visual feedback systems situated in a fully immersive environment. Use of optimization techniques, assembly planning constraints, and CAD data, in combination with these physical elements enables the assembly planner to analyze and perform the complex assembly tasks in immersed virtual environment.

According to Kranzlmiller et al. (2001), “advanced human-computer interface is necessary to evaluate the complete functionality of any system”. They developed a speech processing unit, VIRO, to provide improved control of the environment while functioning in the virtual environment. This helps to eliminate the traditional ways of controlling and navigation using keyboard and mouse. Simple voice commands are converted to specific actions. Being in immersive stage the commands can be given to control any device, which in reality reduces the manual handling of these devices. Similarly, Zachmann and Rettig (2001) proposed the use of robust and natural interaction techniques to perform assembly tasks. The use of multimodal interaction techniques such as gesture recognition, speech input and menu options have enabled precise positioning of the assembly components.

Ozaki at al. (1999) developed and applied a combined verbal/non-verbal interaction approach between the user and the computer to an assembly environment. Their system successfully interprets the user’s vocal commands into specific actions and facilitates the assembly process by executing the given instructions. Non-verbal interactions, such as pointing at an object to help identify the object, are used to complement the verbal interactions. By integrating verbal instructions along with non verbal actions, it was found that the user’s intentions can be more clearly conveyed to the system and interpreted into appropriate actions.

Zhang and Sotudeh (2004) detail the utility of combined approach of auditory and visual feedback into virtual assembly environments. Based on surveys conducted in the context of a peg-in-a-hole assembly task, they concluded that combined auditory and visual feedback of collision and misalignment, helped improve the performance of immersed operators.

This paper presents our approach towards integration of voice commands into a virtual assembly environment developed in Jack to aid the immersive operator performing the assembly tasks. While the user’s hands and vision are occupied with the actual assembly task at hand, he can use voice commands to carry out various auxiliary tasks typically required for facilitating the immersive virtual reality session. Section 2 below gives a brief description of the virtual environment for assembly. Section 3 describes how voice commands are recognized and passed to Jack and section 4 describes how they are used by Jack to facilitate virtual assembly sessions.

2. DESCRIPTION OF EXPERIMENTAL SETUP FOR IMMERSIVE ASSEMBLY DESIGN

Jack®, the Human Simulation and Ergonomic Analysis software marketed by UGS Corporation is the software in which the virtual environment has been developed. Jack can map the motion of an immersed operator onto a humanoid and provide stereoscopic views from the humanoid’s eyes. The built-in tools for motion analysis, realistic behavioral control, anthropometric scaling, task animation and evaluation, view analysis, automatic reach and grasp, collision detection etc., provide an ideal platform for immersive virtual reality based assembly and maintenance design.

The Flock of Birds (FOB)™, electromagnetic motion tracking system is used to track the position and orientation of key segments of the immersed operator. The FOB system comprises of an extended range controller (Master bird) connected to the host computer and to a set of slave birds. Each of the birds individually tracks the position and orientation of one 6 degree-of-freedom sensor. An extended range transmitter which is connected to the extended range controller emits a known pulsed DC magnetic field 120 times per second. The magnetic field measured by each sensor is used to compute the six degrees of freedom information giving its position and orientation relative to the transmitter. The transmitter, a 12-inch cube, is mounted from the ceiling 8 feet above the ground and the operating range of the sensors is a hemisphere of 8 feet radius with the transmitter as its center.

In our environment, five sensors are used to track the position and orientation of five parts of the human immersed in the virtual world. Two of the five sensors track the right and left palms, the third tracks the head position and orientation, the fourth tracks the pelvis and the last one is attached to the back of the neck to estimate the curvature of the spine. The position and orientation of the sensors is indicated within Jack by figures called Birds, each of which looks approximately like the sensors. The position and orientation of each of these sensors is interpreted by Jack as being relative to a cube named FOB_trans_icon, which can be positioned by the user as required.

Within the virtual environment in Jack, we create a humanoid scaled to represent the body proportions of the immersed human operator. The palms, head, pelvis and back of the neck of the humanoid are constrained to follow the location of the five sensors tracking the corresponding locations of the actual immersed human operator. The locations of the other parts that are not tracked, for instance, the elbows, are computed by Jack using inverse kinematics algorithms.
Since we do not have any sensor below the pelvis, the legs get translated with the pelvis. Thus, this environment only tracks the posture from the waist upward. As the immersed user’s posture changes according to the movements of his/her body parts, the motion of the sensors attached to the body parts is passed to Jack, which in turn updates the position and posture of the humanoid. When the humanoid is scaled to the correct dimensions of the immersed user, the postures of the humanoid accurately mimic those of the immersed user.

The experimental setup also includes the user being immersed in the virtual world wearing a Head Mounted Display (in the present case, i-glasses Pro 3D™ HMD developed and marketed by IO Display Systems Inc.). Jack uses the current location and orientation of the eyes of the humanoid to generate and output each eye’s view to the corresponding LCD screen on the HMD so that the immersed human sees what the humanoid ‘sees’ in the virtual world.

As may be imagined, detailed information about the fingers is necessary for virtual assembly operations. This is obtained by using left and right hand Cyber Gloves™. The joint angles of each of the finger joints are mapped by Jack onto the corresponding joints of the humanoid.

3. SOFTWARE UNITS FOR RECOGNITION OF VOICE COMMANDS

3.1 IBM WebSphere Voice Server

We have used the IBM WebSphere® Voice Server Software Developers Kit (SDK) to create a voice application written in Voice eXtensible Markup Language (VoiceXML). VoiceXML is a XML-based markup language to generate voice based applications, which is analogous to the use of HTML for generating visual applications (VoiceXML Programmer’s Guide, 2000).

The VoiceXML browser is started using the command line interface method. The WebSphere voice libraries are included within a Java run time environment and an initialization VoiceXML file is executed. This verifies that the user wants to begin an assembly simulation and requests the Resin application server for the actual VoiceXML application file containing the grammar of interest in the virtual environment. Note that different grammars can be invoked for different environments based on the choice made in this startup file. When a well-differentiated and compact grammar is used, it is found that the IBM speech recognition engine is able to understand different accents without error, even without training.

Details of the logic of the voice recognition application as well as the interaction between the VoiceXML browser and the web server are given in the flowchart in Figure 1. The VoiceXML application provides an introduction to the voice commands that are available, waits for speech input and processes it using the speech recognition engine. If the input matches the defined grammar, it is passed to the Resin server for output to a TCP/IP port, along with a request to send back the application document. This set of actions is repeated in a loop till the “good bye” command is received.

3.2 Resin Application Server

We have used the Resin® application server, which is one of the better servers from the point of view of serving up dynamic content (Taylor, 2005). In the Resin web application folder we have created a JavaServer Pages file which is executed upon the request sent by the VoiceXML browser. This outputs the voice command recognized by the VoiceXML application to a TCP/IP port that Jack listens on. It then returns a VoiceXML document, containing the grammar of interest in the virtual environment being used, to the Java virtual machine running the VoiceXML browser.

4. PYTHON PROGRAM IN JACK FOR EXECUTING THE VOICE COMMANDS

A python program using the socket module, and running within the python console provided by Jack, serves as the client for the voice commands output by the Resin server. This python program contains different subroutines for carrying out the intent of the different voice commands defined in the grammar for the virtual environment in use. The main program initializes the variables used and repeatedly calls the function cmdReceive() in a loop till the “quit” command is issued by the user. The socket connect method is invoked within the function cmdReceive() to connect to the TCP/IP port that the Resin server broadcasts the voice commands on. Then, it sends the grammar for the assembly virtual environment, comprising the list of commands that have been implemented so far, is as shown in Figure 1. For each command received, the corresponding subroutine is executed. Variables are used to remember the state of the simulation. For instance, a global variable called dataLastCommand is created in this program to record the last navigation command (slide, rotate or fly) so that it can be sped up or slowed down if one of the commands “faster” or “slower” is received.
During the execution of the actions in many of the subroutines, the function `cmdReceive()` is called again, so that sequential actions ordered by the voice input from the operator can be executed. For example, during posture capture, if the “slide back” command is issued by the operator, the Jack humanoid will move back until the operator gives another command. Once these interrupt actions are finished (by the “freeze” voice input for the example chosen), the posture capture can be continued.
4.1 Capture Function

One of the main outputs of a virtual assembly session is a set of posture sequences showing the postures of the human interacting with the environment. These may be used later for simulation of the task sequence and for ergonomic analysis, visibility analysis, etc. The program logic for capturing sets of discrete postures is as shown in the flowchart in Figure 2.
Note that the capture command stores the current posture of the humanoid as well as the position and orientation of any object that may be grasped by the humanoid, and begins to wait for additional voice input. For each “next posture” command, the posture at that instance is added to the array of stored postures. When the “end capture” command is issued, the stored postures are saved to a channelset file. In practice, we find it convenient to save postures in short sequences, for ease of review and revision. If the command “capture” is issued while still within a parent “capture” loop (identified by the global variable ‘capturing’ being set to “true”), the user’s command is interpreted as the desire to save another posture and the subroutine nextPosture() is executed to save the current posture. Similarly, if an “undo” command is issued during the capture function, capturing is terminated, all the stored postures in the array are saved to a channelset file and the undo() subroutine is called to set the object grasped and humanoid to the first frame positions, thereby taking both these to their initial positions when capturing began.

4.2 Undo and Redo Functions

While carrying out assembly tasks, the user may sometimes not be satisfied with the step or sequence he has just finished. If the user wants to review the sequence, the voice command “undo” can be issued. Once the undo command is given, the first index value of the global array objectGrabbedarr set by the capture() function is popped and used to generate the channelset file name to be loaded into the Jack window. It also creates a duplicate humanoid named “playbackHuman” and shows him as well as the object grabbed in the position of first frame of the channelset file. Thus the entire sequence is undone. If the user still wants to go back more steps he can give additional “undo” commands.

If the user wants to review the sequence undone, he can give the command “redo”. The redo() function will load the channelset file with the name given in the variable filetobeplayedback that is updated by the undo() function. All the frames in this channelset file can be played back at a given speed. This will help the user to decide whether the posture sequences satisfy requirements and constraints.

4.3 Slide Function

While the movements of the immersed human naturally control the position and posture of the humanoid onto which the motions are mapped, as described previously, the range of virtual space the humanoid can inhabit is restricted by the motion capturing volume. To circumvent this, we note again that the position of the humanoid is set by Jack with respect to the FOB_trans_icon and that the range of virtual space accessible to the humanoid can be changed by changing the position and orientation of the FOB_trans_icon. Also note that, since the humanoid body parts are constrained to the bird figures in the Jack environment, if we attempt to directly move the humanoid figure, the pelvis of the humanoid will move with respect to the waist sensor and this offset will cause the postures to be incorrect. Since the position and orientation of every bird is based on the FOB_trans_icon, any change of location of FOB_trans_icon will cause all the birds to change their locations. A function myTrans (direction, distance) is used to make the FOB_trans_icon move any distance we want in the specified direction while the user is still in his original location in the real world. The direction is determined with respect to the horizontal component of the view direction of the humanoid. By this approach, the spatial constraints imposed by the finite tracking volume can be eliminated and the entire virtual environment can be made accessible to the immersed human.

The logic for commands in the slide group is detailed in Figure 3. Using a similar logic, we have also implemented the fly and rotate functions. The difference between fly and slide is that in the flying mode the user’s movements can be in all three dimensions, as opposed to being constrained to be in the horizontal plane for the slide commands. The rotate commands rotate the FOB_trans_icon about a vertical axis passing through the humanoid, to give the impression that the humanoid is spinning about its own axis.

4.4 Speed Control Function

With the commands in the slide, fly and rotate groups, the humanoid can easily move to any place the immersed human wants to and at a specified speed which can be controlled by commands like faster and slower. Once one of the voice commands “faster” or “slower” is issued, the mySpeed (“chgspeed”) function will be called with this modifier as the parameter. Knowing the current type of motion from the variable dataLastCommand updated by the function cmdReceive() the corresponding speed is either doubled (for “faster”) or halved (for “slower”).
4.5 Assembly Function

In any immersive environment, achieving the balance to assemble the virtual components exactly to the expected position is very challenging. Using the function assembly(), once an object is close enough to its assembled position, the immersed operator can move the grasped object to its expected assembly location (that should be recorded prior to the beginning of the simulation, from an environment with all the objects in this assembled position, using the function getfinalpos()).

4.6 Function to Switch the Grasping Mode

The switchGrab() function passes the Boolean variable fingerTF to switchGrab.tcl, which sets the TCL/TK global variable fingergrab to this value. This global variable is used by Grasp.tcl to change the mode of humanoid grasping between grasping between the palm and the fingers and pinch type of grasping between two fingers.

5. APPLICATION

The virtual environment along with the voice commands has been used to develop the assembly simulation of the windshield deicer of a Cessna aircraft as a trial case. Models created in IGRIP consisting of approximately 200 components which form the final assembly are imported into the virtual assembly environment. Voice commands were used to navigate in the virtual world, to capture the key postures, etc. Since many of the objects to be grasped were very small, new voice commands were added to trigger the grasping of parts in a predefined assembly sequence and this was found to reduce the effort required in IVR. The voice commands also allowed the user to position themselves in any desired posture without any external assistance. The “assembly” to position objects in their final position was also found to be very useful.

6. CONCLUSION

We have presented a highly flexible approach to the integration of voice commands for carrying out various miscellaneous tasks required in virtual reality sessions. Grammars specifically tailored for each type of immersive VR task can be called via the VoiceXML application. The use of a web server allows the voice recognition to be carried out on a separate PC. The use of voice commands has proven to be helpful for capturing key postures and for evaluate these postures. Follow-up work currently being carried out is focused on extending the command set to include additional commands and for implementing enhancements such as auditory feedback by which the virtual environment can inform the user of the status of the tasks being performed.
ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. 0125414. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES


