SAE 25th International Conference on Environmental Systems, San Diego, CA, July 1995 (pre-print)

VEVI: A Virtual Environment Teleoperations Interface for Planetary Exploration

Butler Hine, Phil Hontalas
Intelligent Mechanisms Group, NASA Ames Research Center

Terrence Fong, Laurent Piguet
RECOM Technologies, Inc.

Erik Nygren
Massachusetts Institute of Technology

Aaron Kline
University of California, Berkeley

Abstract

Remotely operating complex robotic mechanisms in unstructured natural environments is difficult at best. When the communications time delay is large, as for a Mars exploration rover operated from Earth, the difficulties become enormous. Conventional approaches, such as rate control of the rover actuators, are too inefficient and risky. The Intelligent Mechanisms Laboratory at the NASA Ames Research Center has developed over the past four years an architecture for operating science exploration robots in the presence of large communications time delays. The operator interface of this system is called the Virtual Environment Vehicle Interface (VEVI), and draws heavily on Virtual Environment (or Virtual Reality) technology. This paper describes the current operational version of VEVI, which we refer to as version 2.0. In this paper we will describe the VEVI design philosophy and implementation, and will describe some past examples of its use in field science exploration missions.

I. Introduction

A. Background

Remotely operating complex robotic mechanisms in unstructured natural environments is difficult at best. When the communications time delay between the control station and the remote mechanism is large, the difficulties become enormous. These mechanisms tend to have a large number of degrees of freedom to control, such as actuators, sensors, and discrete powered subsystems. The mechanism will interact with its environment during the control of these degrees of freedom, and the mechanism's subsystems will interact with each other. The cost of a typical planetary exploration mission is often driven by the need to maintain a large ground control staff in order to monitor and control the mechanism during the mission.

B. Current Practice

Current practice for the control of a planetary exploration mission is for the robotic mechanism to have minimal on-board automation, in the sense of being able to make operational decisions itself. The control of the mechanism is accomplished by monitoring telemetry received from the system, interpreting these telemetry data to determine the state of the mechanism, and uplinking commands to the mechanism to achieve a desired new state. The telemetry data are often displayed as numeric streams or graphs, and the uplinked commands are entered by the operators through typed text strings and/or button presses. The interpretation of the system state from the telemetry data is usually so complex that each subsystem has one or more dedicated human controllers monitoring it at all times. The controllers are organized in a hierarchy, with the lowest controller levels responsible for monitoring and controlling the lowest hardware levels on the spacecraft, and the top of the hierarchy responsible for the entire spacecraft and the mission. This layering of responsibility is done to ensure that any single controller is not overly tasked with the information flow from the spacecraft and decision-making for the mission. Except for quiescent phases of the mission, such as interplanetary cruise, these controllers are on constant duty at a dedicated control center.

C. Drawbacks

Although the conventional approach described above has worked well enough in the past, it is very inefficient in terms of science return and very expensive in terms of total mission cost. The science return for planetary exploration missions can be quantified as science data returned per command uplink cycle. Traditional mission control approaches have a significant delay between the time telemetry data are received and a command uplink is performed in response to the data. This is due to the time required for expert controllers to interpret the telemetry data, pass it up the hierarchy, make a decision, pass the decision down the hierarchy, and uplink it to the vehicle. The decision making process time is affected by the time required to synthesize the current state of the vehicle from the subsystem states. For complex mechanisms with a large number of subsystems, interacting strongly with the external environment, the current state of the vehicle is not easily synthesized.

For traditional missions, the total mission cost is strongly affected by the size of the ground control staff. The more complex a mechanism and the more the mechanism interacts with its environment, the larger the ground control staff required to accomplish the mission. Not only is the ground control staff large, but it is also highly trained in the mechanism design and operating characteristics. This is required in order for the controllers to properly acquire and interpret the telemetry data from the mechanism.

D. A Different Approach

An entirely different approach to operating complex planetary exploration mechanisms is to significantly reduce the size of the ground control staff by putting more software automation on board the vehicle for local decision making. The type of local vehicle automation we refer to is position and path control of the vehicle, safety reflexes, and goal driven behavior. The ground control staff can be further reduced by implementing highly automated ground control software to receive, interpret, and synthesize the state of the mechanism for the ground operator, presenting the state of the vehicle in a very concise and comprehensible fashion.

The human visual system is a very high bandwidth means for communicating complex information to a human operator, but only if that information is properly presented. A fast reader is able to acquire information from a well organized text stream at several hundred words per minute, which corresponds to approximately 10^4 bits per second (bps). If the information is plotted as a graph (1000 points presented at 10Hz, with a resolution of 10 bits per point), then the data rate is increased to approximately 10^6 bps. If the information is presented as animated full color images (1e6 pixels with 20 bits per pixel at 30 Hz), then the data rate increases to over 10^8 bps. In addition to increasing the data acquisition rate for a human operator, presenting complex geometric information visually allows the operator to discern patterns and relationships which are not apparent otherwise.

Our goal is to develop a user interface to control complex mechanisms which allows an unsophisticated operator to comprehend the current and past state of the system quickly, to plan and review high-level commands to the system, and to send those commands for the system to execute. To take advantage of the highest bandwidth sensory channel available to a human operator, and to encode the information in the most natural manner possible, we have applied techniques from virtual environment (VE) or virtual reality technology [1][2]. Virtual Environments consist of highly interactive three-dimensional computer-generated graphics, typically presented to the user through a head-mounted, head-tracked stereo video display [3]. Although the techniques allow a designer to present almost any information to the user in any arbitrary fashion, the information is typically presented as a familiar physical environment. This creates the illusion that the user is physically present in the environment and is termed “full immersion.” In its extreme form, full immersion attempts to replicate all human senses, such as sound and touch.

The operator interface software described in this paper is called the Virtual Environment Vehicle Interface (VEVI). The system not only consists of the virtual environment interface itself, but also includes the distributed software architecture required to support it. This paper describes the design and implementation of the first versions of VEVI (through version 2.0), which have been used extensively for various field tests we will describe later. A complete rewrite of VEVI is nearly completed, which incorporates all of our current experience with the current version and addresses the current version’s limitations. This next version is called VEVI 3.0, and a paper describing its initial design and implementation has been published by Piguet [4].

II. Requirements

During the initial design of VEVI we tried to satisfy a list of requirements. Some of the requirements are imposed upon us by the types of planetary exploration missions we want to address, and some of them we generated ourselves in order to make the system easily maintained and extensible. The next sections list and discuss these design requirements.

A. Time Delay

One of the primary requirements for the operator interface is that it be usable and efficient in the presence of long communications time delays. In typical terrestrial remote telerobotic applications, the communications time delays vary from milliseconds to a few seconds, depending on the proximity of the operator to the mechanism and wether or not satellite communications are involved. For short time delays direct teleoperation of the mechanism is possible, where rate or position control input devices are employed by the operator to send commands to the mechanism or vehicle, and live telemetry and camera views from the remote site are sent back and displayed to the operator. If the cameras are configured as anthropomorphic stereo pairs and are slaved to the operator’s head, then the system is often described as a “telepresence system”. When the time delay grows much larger than the dynamic time scale of the remote mechanism, this approach starts to become very inefficient because the delay between the sensing and the actuation makes the control system unstable. When faced with this kind of time delay, human operators will quickly adopt a “move and wait” control strategy, in which the results from the last command are confirmed before the next command is sent [5][6]. This strategy is very inefficient in terms of operations time because during the round trip delay time the mechanism is typically idle. The move-and-wait strategy also quickly induces fatigue in the human operator. For very large time delays, such as the tens of minutes light travel time between Earth and Mars, a move-and-wait control strategy would be very expensive in terms of the science accomplished during the lifetime of the mission.

B. Communications Bandwidth

A related problem to that of time delay is the communications bandwidth from the human controller to the remote mechanism. The communications systems employed for a planetary exploration mission are typically limited by the transmission distance, the power available to the remote mechanism, antenna sizes and pointing requirements, and downlink system availability. Typical Mars mission designs allow communications bandwidths of much less than 10^6 bits per second, with some as small as tens of bits per second. With these bandwidths, it is difficult or impossible to support any kind of closed-loop control over the vehicle if the dynamic time scale of the vehicle in its environment is short. The typical response is to either rely on an even less efficient move-and-wait strategy, or to incorporate local decision making capability into the system [7]. The operator interface should be usable and efficient when operating over low bandwidth communication channels.

C. Efficient Command Cycles

Since the fundamental metric of any planetary exploration mission is the quality and amount of science data returned during the life of the mission, the efficiency of the control strategy is proportional to the average science data return per command uplink cycle. If a controller must send a dozen commands to the vehicle in order to position it for a single science measurement, with the separation between each command being at least as long as the round trip light travel time, then this is less efficient than a controller acquiring the same science measurement in response to sending a single command. The difference between the single command and the dozen commands to accomplish the same goal in the above description is the “level” of the command. A high-level or task-level command is one which accomplishes an entire task in response to a single command [6][8][9]. Lower level commands to do the same task would consist of a series of smaller tasks. Each command to the vehicle is delayed by the round trip communications delay, but the high level command takes one delay time, while the low level command sequence takes multiple delay times. The operator interface should support both high and low level commands, to allow the operator maximum flexibility in sending efficient command sequences.

D. Situational Awareness

A typical problem with control interfaces that depend on fixed camera views is that it is difficult for the operator to develop sufficient situational awareness of the mechanism in its surroundings. The field of view of the cameras is typically restricted, and the cameras cannot be slaved to the operator’s head due to the communications delays mentioned above. Humans operating vehicles on the Earth have a very wide field-of-view, which is coupled with a very fast and flexible scanning mechanism (the neck and body) which allow them to quickly build up an accurate global view of the surroundings with which to safely operate the vehicle. A human operator will often remember where obstacles and hazards are without having to look at them repeatedly. When operating a vehicle using fixed moderate-field-of-view cameras, in the presence of significant communications time delays, a human operator will have a greatly reduced situational awareness. Planetary exploration missions in hazardous environments under these conditions are at increased risk. This results in the operator making decisions more slowly and more conservatively than necessary. This in turn leads to reduced mission efficiency, as described above. The operator interface should lead to greater situational awareness than can be accomplished with the traditional control approach.

E. Lower Operations Cost

A major life cycle cost in space flight science missions is the traditional requirement for a large ground staff to monitor and control the spacecraft, in addition to whatever science teams are directing the science operations for the mission. Ground control of the vehicle is required because the spacecraft have minimal capability to monitor themselves and take autonomous action for normal and emergency operations. The large size of the ground support staff is dictated by the requirement that each subsystem of the spacecraft have one or more controllers each shift who are experts in the subsystem. The virtual environment-based operator interface we are developing can potentially reduce the number of mission controller required to run a space flight mission, which can in turn lower mission costs dramatically.

F. Distributed Operations

Another significant cost for science exploration missions is the requirement that science operations team be co-located at a centralized science operations facility. Not only does this increase the cost of the mission, but it also tends to reduce active participation in the mission by large segments of the science community. One way of addressing these problems is to design the mission operations system to support distributed command and control, allowing science team participation in the mission from widely distant sites, with minimal support hardware required. The communications infrastructure in the U.S. and in most of the world is easily sufficient to support this type of science operation during a mission. The ground operations interface software should allow highly distributed mission monitoring and control. In addition, it should allow wide distribution of the science and mission data and status for educational purposes.

G. Modular Components

Modern large software systems, such as this one, can only be efficiently maintained and extended if the software is written in a modular fashion. Not only does this allow the system to be more easily designed and implemented by large teams of programmers, but it also allows for future modification and extension of the system with lessened impacts of possible side effects on the rest of the system. This design approach also allows for a large degree of code re-use from mission to mission, as only small parts of the system need to be changed to support different vehicle configurations and designs. We will describe several radically different vehicles upon which this same system has been implemented, demonstrating the amount of re-use possible with this implementation.

H. Commercial Products Where Possible

In the past, the traditional NASA approach to designing mission software has been to start from scratch for each mission, which repeatedly incurs a very high cost. Current missions have become too expensive to allow this approach to continue. The capability, quality, and maturity of software and hardware products available commercially off-the-shelf (COTS) from industry is such that NASA cannot hope to develop similar systems at reasonable cost. We believe that leveraging from commercially available tools and products is the only cost efficient approach to large software systems for NASA space missions. The VEVI system has been designed and implemented using as many COTS components as feasible, as will be described below.

III. Design

A. System Architecture

The VEVI system was designed to satisfy the above requirements. From previous work with controlling a Puma manipulator arm from within a virtual environment operator interface [10][11], we expected that an operator interface based on virtual environment techniques would be able to compensate for long time delays between the operator and the remote vehicle. For this remote manipulator work, we were using the Task Control Architecture (TCA) [12] to send commands to the Puma arm, and to operate a distributed machine vision processing system. We were impressed with the manner in which tasks could be distributed over a homogenous computer network. We decided to design the VEVI system in a distributed manner, to take advantage of multiple processors when needed.

The distributed architecture we are using has been described previously [13]. We will briefly summarize it here. The system components are organized as processes on a communications backbone. The processes consist of 3D renderers, 2D displays, user input devices, and data archivers, all tied together with various communications nodes. Each process sends and receives data through a communication node, which relays data to other communication nodes in the system. Figure 1 is a diagram of a

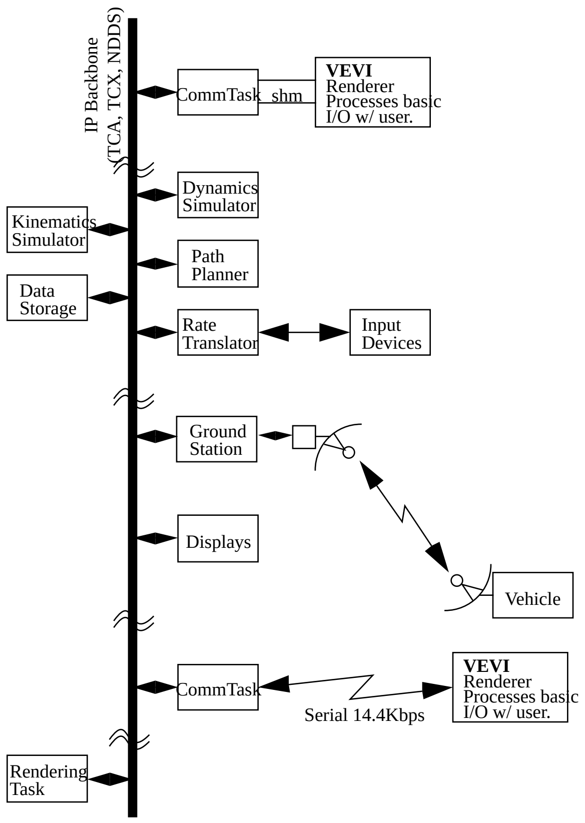

Figure 1 System configuration diagram of VEVI communications. All of the processes communicate to each other through communications nodes.

typical VEVI system configuration for a single operator and a single vehicle. We have used various communications layers to tie together the communications nodes, but the most common one was TCA.

B. Rendering/Simulation Layer

When an operator is using VEVI in a the virtual environment mode, he or she will be mainly interacting with the 3D renderer. This process displays the vehicle and its environment to the user as a highly interactive three-dimensional graphical display. The output to the user can range from a flat panel 3D display to a full-immersion stereo head-mounted head-tracked display. The 3D display process exchanges information with the rest of the system through a local communication node over a shared memory link. The 3D renderer is not a simulator, but is rather a system display. Unless explicitly driven by an external dynamic simulator, the objects in the environment represent the actual state of the remote environment. Their behaviors are driven entirely by external messages generated elsewhere in the system (usually by the vehicle to be controlled).

1. WorldToolKit™ (WTK)

The 3D renderer is basically an interactive graphical simulation, which displays the vehicle in a given state along with whatever is known about the environment around the vehicle. At the time of the design of this system there were various options we considered for implementing the 3D renderer. One approach was to develop the renderer from a low level graphics library such as GL or PEX, as was done for VIEWS [10][11]. A second approach was to take advantage of higher level commercially available graphics libraries, such as WorldToolKit™ (WTK) from Sense8 Corp. [14] or from VPL, Inc. A third approach was to extend a full commercial simulation package, such as IGRIP™ from Deneb Inc., and CimStation™ from Silma Inc. Our design philosophy was to use COTS as much as possible and still achieve our goal. This led to our decision to use a high-level commercial graphics and simulation library. After reviewing the capabilities and degree of extensibility of the product, we chose the WorldToolKit™ package from Sense8 Corp. The main advantages of this package are: (i) it is an architecture-independent Application Programming Interface (API), making porting our system to different platforms relatively easy, and (ii) it takes advantage of the native graphics rendering hardware support on most platforms. One of the beneficial aspects of working with a commercial product is the opportunity to influence the vendor to improve the product with respect to NASA's needs, and so return NASA technology back into the commercial world. Over the development cycle of the VEVI system, we continually beta-tested new versions of the WTK library and gave feedback to the vendor on problems we discovered or features we needed. We found the response of the company to our requirements and feedback to be excellent.

2. Communications Nodes

All of the processes in VEVI, such as the 3D renderer, the 2D displays, the data archivers, and the user input devices, communicate to each other using communications nodes. These nodes are most commonly paired with a single process, and communicate with that process over a dedicated shared memory link. The node is responsible for taking data from the process over the shared memory link and sending them out over the communications link to other nodes. The node is also responsible for information flow in the reverse direction, from the external system to the process. The node task handles any data formatting or translation necessary, and serves as a data rate throttle in some cases. The rationale for this layering is to insulate the high performance graphics process, which should run at a constant (high) rate, from the data transmission task, which can run at an unpredictable varying rate (high or low). Another reason for this communications approach is that swapping communications link protocols becomes much easier, as the rendering process need not be changed at all.

A primary benefit with this distributed layered communications approach is that processes can be added or removed at will during mission operations. We commonly use this feature to bring in parallel control stations at geographically remote locations during a mission, for collaboration or educational purposes. We have also used this feature to temporarily add processes such as a data archiver or alternate input device to a running system.

C. Communications Layers

The first communications layer we used with VEVI was that of the Task Control Architecture (TCA), and it still remains the most common one we use. TCA is actually much more than a communications layer, being a tool designed to implement autonomous systems [12]. The higher functions of TCA are built upon a general communications layer which has most of the features we required. TCA has been ported to a variety of platforms, and uses the TCP/IP protocol. The features of TCA which we use are centralized registering and routing of messages, which allows us to start up and shut down communications tasks without a priori knowledge of the system configuration, and message broadcasting, which allows us to generate state information from our various processes without requiring knowledge of the recipient.

We have used or investigated the use of other communications layers for VEVI, such as TCX [15], TelRIP [16], NDDS [17], and various custom protocols such as PiVeCS (University of Maryland Space Systems Lab), and TCPREAD (Stanford University Aerospace Research Lab). These other communications layers either replaced TCA in our system or were used through custom gateways which translated between the communications formats.

D. Communications Protocol

The information flowing into or out of the distributed processes consists of either state information or commands. State information is typically broadcast from some other part of the system, such as the vehicle or another operator, and reflected as a change in the displayed configuration to the operator. A state update might consist of the position and orientation of the vehicle in space, along with the position and orientation of any subsystems of the vehicle which are articulated. Examples of articulated subsystems are pan-tilt cameras or manipulator arms. Commands are sent from the user interface components to the vehicle or the rest of the system, or from components of the user interface to each other. A command will often change the internal state of the user interface, such as turning on a navigation grid or repositioning the viewpoint.

Data are exchanged between the local components of the user interface and the local communications node via a block of shared memory. The shared memory is organized as a data structure with sections for state information and read/write synchronization. The most common shared memory data structure used by VEVI is shown in figure 2.

To broadcast a state update from within the operator interface, in response to a change in viewpoint for instance, the 3D renderer will load the new viewpoint into the data structure,

```
typedef struct remote_control {
 int command;
 int notready;
 int value;
 int status;
 int count;
 float x, y, z;
 float qx, qy, qz, qw;
 char name[NAMLEN]; /* network obj name */
 char obj[OBJLEN]; /* obj file name */
 char time[TIMLEN];
};
```

Figure 2 Shared memory data structure for communicating between VEVI processes and communications nodes.

load its own unique network identifier and icon into the “name” and “obj” fields, then load the message type, “REMOTE_UPDATE”, into the command field. The communications node will see the new command appear in shared memory and will transfer the state information into a TCA message structure and call the TCA broadcast function. Figure 3 shows the TCA message

```
typedef struct { /* object type */
 char *name; /* text name of the object */
 char *obj; /* description of the object */
 long status; /* object status flags */
 POSE_TYPE p; /* position and orientation */
 char *time; /* current time */
 long validity; /* Pose validity */
 };
```

Figure 3 TCA message structure used between communications nodes.

structure. Other communications nodes currently active in the system will then see this state broadcast and act upon it accordingly. The network identifier is a string attached to every message which identifies the source of the message uniquely. It made up of a concatenation of the user name, the machine, the process ID, the object, and (optionally) the subsystem of the object. An example network ID is: “blah@foobar.arc.nasa.gov*42:Observer”.

The reception of state information or command messages is similar. Since a node using the TCA broadcast function will send messages to itself, each node or process will inspect the incoming state messages to see if it should ignore them. If the message is of interest (i.e. not its own), then the node will take the information in the TCA message structure and copy it into shared memory with the “REMOTE_UPDATE” command. The interface process will periodically check shared memory for commands, and upon seeing one will use the information to change its state (if a command message) or change the state of one of the objects it is maintaining (if it is a state update message). Figure 4 is a list of the command messages currently supported in VEVI 2.0. State updates come in through the “REMOTE_UPDATE” command, environment updates come in through the “REMOTE_TERRAIN_NEW” command, and the other commands change the viewpoint or the list of things displayed to the user.

```
#define REMOTE_NULL 0
#define REMOTE_MOVE_VIEW 1
#define REMOTE_MOVE_OBJ 2
#define REMOTE_RELEASE 3
#define REMOTE_CONSTRAIN 4
#define REMOTE_MOVE_NOT 5
#define REMOTE_ORIGINAL 6
#define REMOTE_FLY_TO 7
#define REMOTE_ZOOM_CHANGE 8
#define REMOTE_ZOOM_RESET 9
#define REMOTE_POINT_MODE 11
#define REMOTE_SELECT 12
#define REMOTE_GO_TO 13
#define REMOTE_DRAW_NAV 14
#define REMOTE_DRAW_COMP 15
#define REMOTE_SET_MARK 16
#define REMOTE_DRAW_MARK 17
#define REMOTE_GO_TO_MARK 18
#define REMOTE_SENS_DTRANS 19
#define REMOTE_SENS_RTRANS 20
#define REMOTE_SENS_DROLL 21
#define REMOTE_SENS_RROLL 22
#define REMOTE_CAMERA_SNAP 23
#define REMOTE_VLOCK 24
#define REMOTE_TERRAIN_NEW 25
#define REMOTE_CONT_KILL 99
#define REMOTE_NODE_KILL 199
#define REMOTE_UPDATE 100
```

Figure 4 List of command messages sent over the shared memory link to the 3D renderer process.

E. File Formats

There are many configuration or model file formats used by VEVI 2.0. The first class of mode file formats are those directly supported by WTK, such as NFF and DXF. These are geometric data file formats, and contain polygon descriptions of objects along with colors and textures of the polygons. Since VEVI is built using the WTK library, any model file format it supports is usable by VEVI. In addition to the basic model files, we require additional information which describes the articulation structure of the mechanism. We designed two file formats to accommodate this extra information. The first, and simplest one, is the Object Definition File (ODF). This file lists the position and orientation of each independently articulated component of a mechanism, along with the articulation degree of freedom, in a tree structure. Without this file, VEVI assumes that the object is a single piece. With an ODF description, VEVI builds a tree of attached subsystems, which it can animate. Figure 5 shows Dante II, one of the more complex vehicles we have animated for a mission, and listing 1 shows the ODF which represents it.

The ODF format is very restricted in the types of articulation it can describe. In particular, manipulator arms with more than a few degrees of freedom are difficult to describe in ODF. For this reason, we developed a greatly extended articulation description format, called the Robot Arm File Format (RAFF). RAFF is adequate to describe arbitrary serial linked manipulators with large degrees of freedom, and is described in depth in Piguet [18]. Listing 2 shows a sample RAFF description of a manipulator.

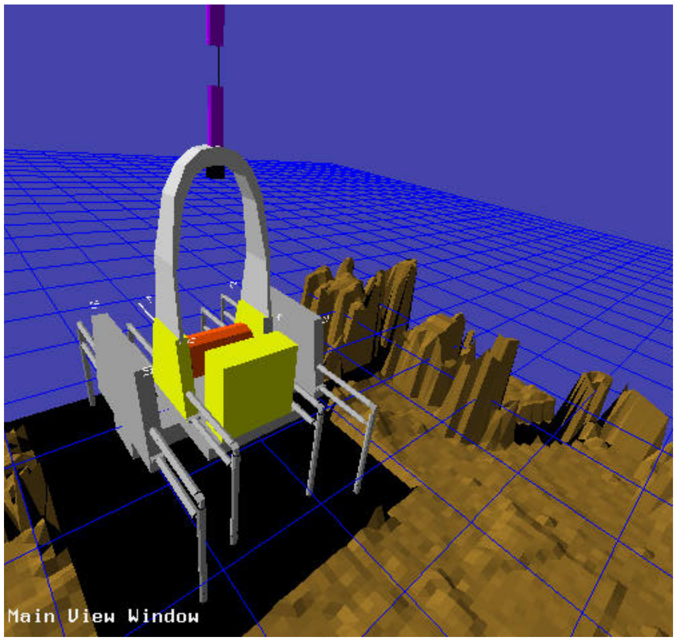

Figure 5 Dante II in VEVI modelled from with the ODF format.

F. Plug-in Modules

The distributed backbone communications architecture underlying VEVI allows for independent support modules to be developed and applied easily. Examples of these are: (i) interface control panels, (ii) state data archivers, (iii) map views, (iv) rate control input devices, (v) vehicle control panels, and (vi) kinematic or dynamic simulators. Activating or deactivating these modules at any time during mission operations is possible due to the communications layering and standards. Figures 6, 7, and 8 show example screen images of the interface control panel, the map viewer, and the vehicle control panel. These typically augment the main 3D renderer interface.

IV. Implementation

As mentioned above, the 3D renderer in VEVI is implemented using the facilities of the WTK library. WTK provides functions to import polygon descriptions of objects in various file formats and then animate them by modifying the object's position, orientation, and other attributes (size, color, etc.). WTK allows the programmer to specify viewpoints and lighting, and provides a particularly simple method for attaching various input devices, such as 6 DOF sensors, to objects and viewpoints. Figure 9 shows a simplified flow diagram of the VEVI 3D renderer.

At start-up, the program reads the command line options, and sets internal flags based on these options. The WTK simulation loop is initialized and the model files are read. If the model file is an ODF file (which can refer to other ODF or NFF files), then it is recursively read until all of the models are read. The lighting and viewpoints are initialized, and the video mode is set depending on the command line flags. Once the initialization is complete, the WTK simulation loop is started and continues until the user (or an external signal) ends the process. During the simulation, the rendering of the objects is maintained by the WTK library, with various user functions called once every pass through the simulation loop. Some of these functions handle rel-

```
#odf
object 1 inner models/Dante_Parts/iframe
object 2 bottom models/Dante_Parts/bottom 1 0.0000 0.0000 0.0000 0 0 0 0 0 0 0 0 0 X
object 3 outer models/Dante_Parts/oframe 2 0.0000 0.0000 0.0000 0 0 0 0 0 0 0 0 0 X
object 4 leg21 models/Dante_Parts/leg1 3 1.0040 -0.4400 0.7240 0 0 0 0 0 0 0 0 0 X
object 5 leg22 models/Dante_Parts/leg2 4 1.0040 -0.4400 1.6552 0 0 0 0 0 0 0 0 0 X
object 6 leg23 models/Dante_Parts/leg3 5 1.0040 -0.1692 1.6552 0 0 0 0 0 0 0 0 0 X
object 7 leg01 models/Dante_Parts/leg1 3 -1.0040 -0.4400 0.7240 0 0 0 0 0 0 0 0 0 X
object 8 leg02 models/Dante_Parts/leg2 7 -1.0040 -0.4400 1.6552 0 0 0 0 0 0 0 0 0 X
object 9 leg03 models/Dante_Parts/leg3 8 -1.0040 -0.1692 1.6552 0 0 0 0 0 0 0 0 0 X
object 10 leg11 models/Dante_Parts/leg1r 3 -1.0040 -0.4400 -0.7240 0 0 0 0 0 0 0 0 0 X
...
object 20 leg42 models/Dante_Parts/leg2 19 -0.5100 -0.4400 1.3372 0 0 0 0 0 0 0 0 0 X
object 21 leg43 models/Dante_Parts/leg3 20 -0.5100 -0.1692 1.3372 0 0 0 0 0 0 0 0 0 X
object 22 leg51 models/Dante_Parts/leg1r 1 -0.5100 -0.4400 -0.4060 0 0 0 0 0 0 0 0 0 X
object 23 leg52 models/Dante_Parts/leg2 22 -0.5100 -0.4400 -1.3372 0 0 0 0 0 0 0 0 0 X
object 24 leg53 models/Dante_Parts/leg3r 23 -0.5100 -0.1692 -1.3372 0 0 0 0 0 0 0 0 0 X
object 25 leg71 models/Dante_Parts/leg1r 1 0.5100 -0.4400 -0.4060 0 0 0 0 0 0 0 0 0 X
object 26 leg72 models/Dante_Parts/leg2 25 0.5100 -0.4400 -1.3372 0 0 0 0 0 0 0 0 0 X
object 27 leg73 models/Dante_Parts/leg3r 26 0.5100 -0.1692 -1.3372 0 0 0 0 0 0 0 0 0 X
object 28 num0 models/Dante_Parts/0.nff3 -1.0040 -1.1000 0.7240 0 0 0 0 0 0 0 0 0 X
...
object 36 tether models/Dante_Parts/tether.nff 1 0.0000 -0.1500 -0.5000 0 0 0 0 0 0 0 0 0 X
object 37 mast models/Dante_Parts/mast 1 0.0000 -1.0000 0.0000 0 0 0 0 0 0 0 0 0 X
```

Listing 1: A partial Object Definition File (ODF) which describes the Dante II articulation structure.

```
# Robot Arm File Format (RAFF) file
# path to geometry files
/u/people/piguet/src/arm/manip/models/
# number of degrees of freedom
4
# number of kinematic frames
4
# number of WTK files for base
0
# file names for base
# Transform matrix: 1
#

# variable joint angle (theta)
180.0
# link offset (d)
100.0
# link length (a)
0.0
# twist angle (alpha)
90.0
# type of joint (1:prismatic 2:revolute)
2
# joint number
1
# lower joint limit
10.0
# upper joint limit
350.0
# number of WTK files for this transform
1
# name of WTK file describing geometry
4base.vst

# Transform matrix: 2
#
```

Listing 2: An example Robot Arm File Format (RAFF) file which describes a simple 4-degree of freedom arm.

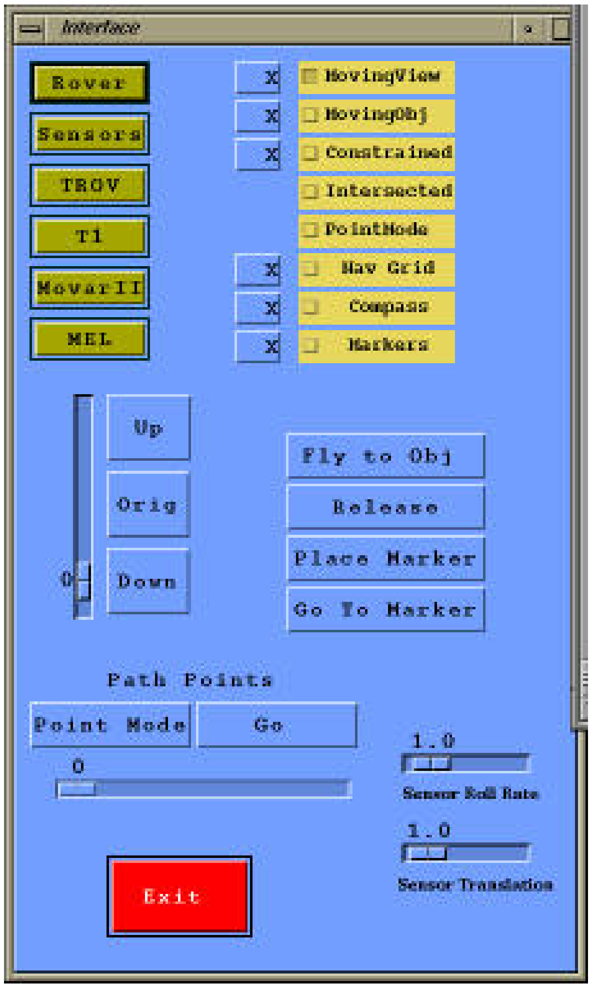

Figure 6 Motif-based interface panel used to send command messages to the 3D renderer over the shared memory link.

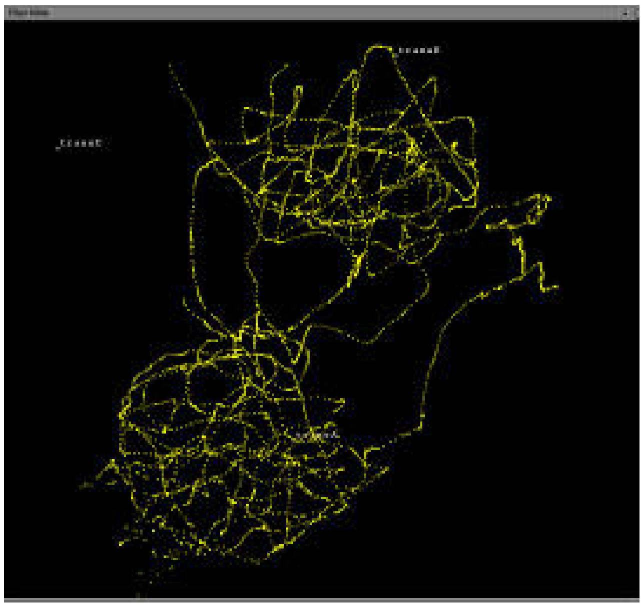

Figure 7 A screen image of the Map View window, showing telemetry tracks of a vehicle during an exploration mission.

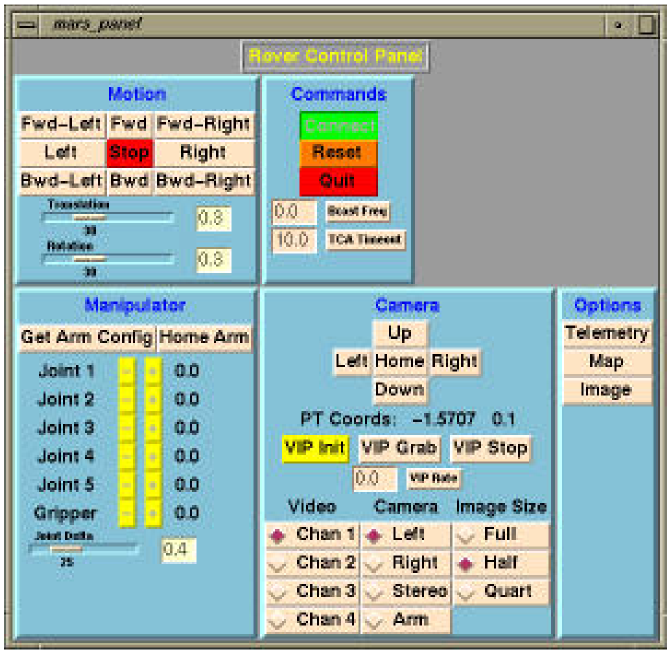

Figure 8 A screen image of a 2D vehicle control panel for the Marsokhod rover.

atively simple tasks like drawing native graphics overlays on top of the WTK polygon rendering. Most of the work modifying the simulation is done in a couple of user functions, however. One of these functions is a general "action" function, executed once per pass. For the VEVI renderer, this function contains all of the command input to the process, whether it be from the keyboard, menu, or external shared memory. The other main functions are tied directly to each graphical object, and are called once per pass if the object is active. In our renderer, this function contains code which perform tasks like terrain following or sending shared memory updates.

A. Interface Modes

The VEVI renderer is capable of supporting a variety of display modes, at the choice of the user. The simplest mode, and the one which requires the simplest output hardware, is a single graphics window on a workstation. Another mode, which requires stereo viewing goggles, will put the workstation screen into a field sequential stereo display mode. Wearing field-sequential LCD goggles synchronized to the display allows the user to view the scene in stereo. A more extreme display mode, which requires significant display hardware, is the full immersion mode. In this mode, the scene is output in stereo to a head-mounted display. Mounted on the head-mounted display is a head-tracking sensor, such as a Logitech^{TM} acoustic tracker or a Polhemus^{TM} magnetic tracker. In this mode, the viewpoint is slaved to the head-tracking sensor. The user experiences a sense of immersion in the scene because the viewpoint changes as the user moves his head, allowing the user to look around in all directions with natural head motions.

Command input from the user can come from a variety of input devices. The simplest one is the common three button mouse, in which the mouse location on the screen is mapped to rates in the three position and three rotational degrees of freedom. More exotic input devices supported by WTK are full six DOF rate and position devices such as pressure sensors (SpaceBall^{TM}), acoustic trackers (Logitech^{TM}), and magnetic trackers (Polhemus^{TM}). These can be used for rate or position input from the hands or head.

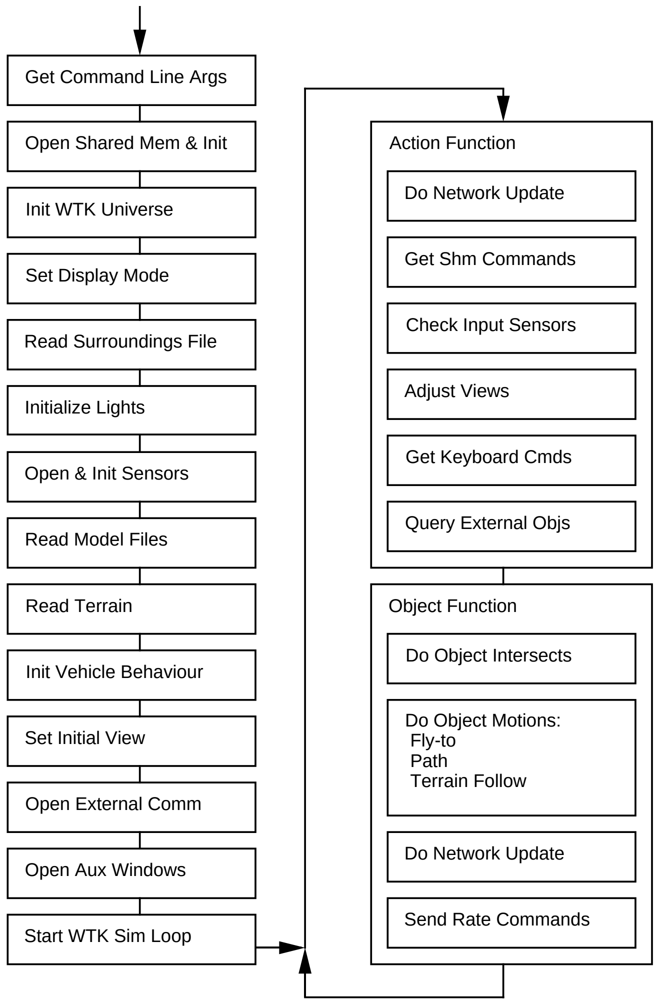

Figure 9 A control flow diagram of the VEVI 3D renderer.

B. Main View

The main view window can be positioned anywhere in the scene by any supported input device, and in addition can be slaved to a head-tracker. In most of the stereo display modes the main view window is the only display, but in the windowed display mode, additional windows can be displayed simultaneously. The most common one is a "bird's eye" window, which shows an overhead view of the scene, centered on the active vehicle. The overhead view also shows the field of view of the main window graphically, and is actively slaved to the main window. Although the operator may position the viewpoint with the input devices, another common command is to attach the viewpoint to an object in the scene. This gives the operator the impression that he is "in" the object and looking out, similar to driving a car. The operator can initiate this attachment at any time, and if the viewpoint is distant from the active object then the viewpoint is "flown" along a trajectory to the object. This is to ensure that the operator maintains a sense of localization in the environment. Figures 10 and 11 show screen images of the VEVI renderer's main and bird's eye windows.

C. Graphics Overlays

In addition to the graphic overlay in the bird's eye window showing the field of view in the main window, there are numerous other auxiliary graphic displays overlayed on the polygon rendering. On the Silicon Graphics platform we normally use

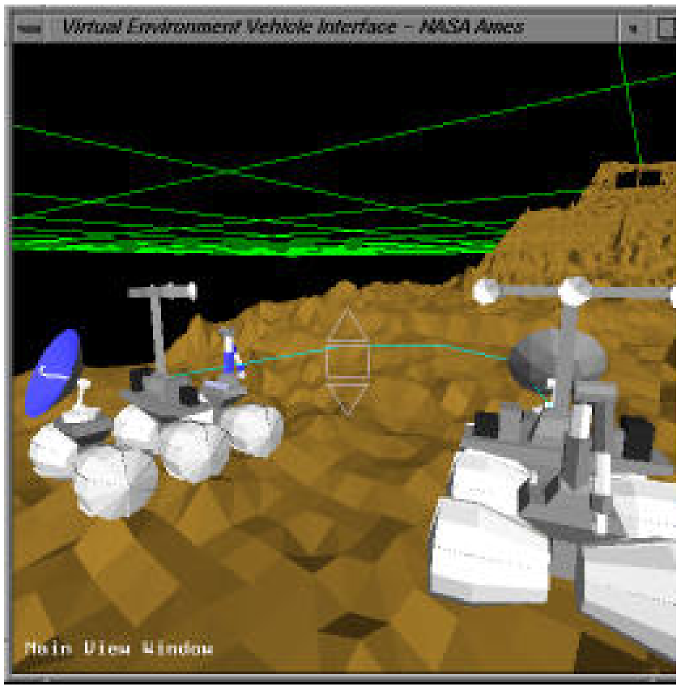

Figure 10 A screen image of the main VEVI view window. The vehicle to the left (in color) is the active vehicle. The vehicle to the right (grayed) is the path planning icon. The green segmented lines connecting them is the path being constructed. The vehicle is rendered on sensed terrain, with a navigation grid visible above.

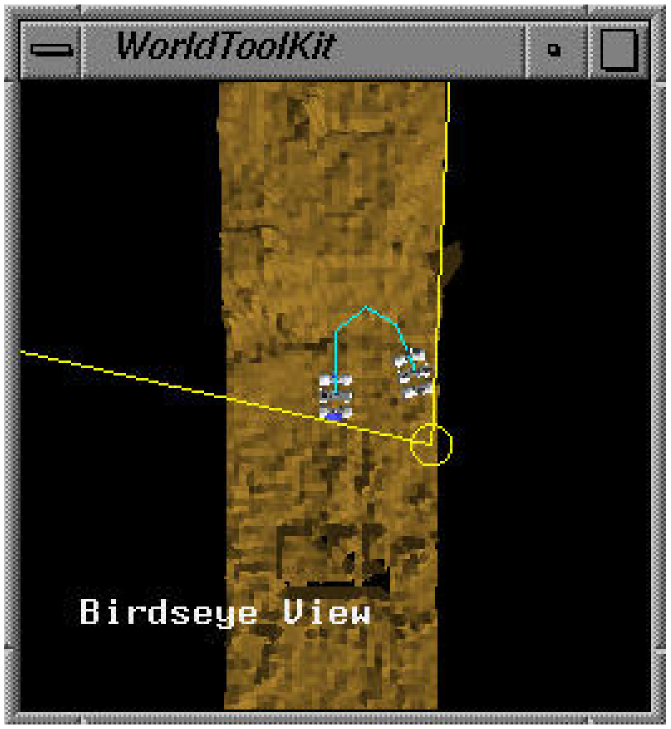

Figure 11 A screen image of the VEVI bird's eye view window, which shows the same scene as in figure 10, but from a viewpoint directly above the vehicle.

for missions, these overlays are simple GL graphics calls designed to display extra information to the operator. There is a navigation grid which hovers a constant height above the active object in the scene, and provides a reference plane when the terrain is at a steep angle. In addition to the navigation grid, there is a compass sphere available to surround the active object in the scene. This is useful as a heads-up navigation display when the viewpoint is locked to the object. Another graphic overlay is available when the user is constructing a path to send to the vehicle. The path is shown as a segmented line. A very helpful graphic display in the scene is a "marker". These can be placed at any time, and are designed to indicate an area of interest in the scene. Some uses of markers in the past have been to designate the starting point for a traverse, to designate a collection point for samples, and to designate science targets to be reacquired later in the mission. An interesting feature of markers is that when active, they are visible from anywhere in the scene no matter how far away they are. This aids in navigating back to them. A final related graphic, which is rendered using the polygon renderer instead of native graphic calls, is the "billboard". This icon is a polygon representation of image data taken at a particular location in the scene. The image data is rendered as a texture on polygons in the scene which are not part of the environment, as a way of organizing and displaying image data collected during a mission.

D. Shared Virtual Environments

A point touched on earlier is that the VEVI renderer not only accepts state updates from the controlled vehicle, but also accepts updates from other renderers present in the system. The active vehicles are represented by models of themselves, while the other renderers are represented by simplified human head models. These "floating heads" in the scene represent the position and viewpoint of other operators in the system. Since all of the operators are viewing the same simulation, remote collaboration during the mission is possible. The network ID discussed earlier, which tags each message sent over the network, contains enough information to allow active communication between the operators. We have tested systems in which an operator could invoke a menu by clicking on a "head" and choose to send an e-mail message to the other operator, open a talk window with the other operator, or open a live digital audio channel to the operator.

The ability to invoke an action by selecting an object in the scene can be used in many interesting ways, aside from the above communications selection. In the VEVI renderer, when an object is selected by the pointing device the current model path in the file system is searched for a file with a name matching the object name, and an extension of ".info". If this file is found, then the file is executed as a shell script. We have used this feature to attach a hypertext design document to vehicle objects, so that the operator could bring up design data on the vehicle by clicking on the object in the simulation. Items like editors for e-mail, talk windows, and hypertext documents are useful when the display mode is windowed, but is less useful when the operator is fully immersed. The resolution of head-mounted displays are still too low for dense text display, and reading text pages in a head-mounted display is awkward.

V. Applications

We have found that the most effective way of developing teleoperations interface software is to bring a prototype of the interface software to an initial level of maturity, and then field test the system in science field experiments. The initial development of VEVI was accomplished with a series of robotic systems and vehicles, and then field tested on several missions described below. We briefly describe each field test, giving a summary of the important lessons learned, but then refer the reader to papers describing each mission more completely.

A. VIEWS and the Puma Arm

As mentioned earlier, the initial work which investigated the use of virtual environments for telerobotic control was performed on a Puma manipulator. The virtual environment system used was the Virtual Environment Workstation (VIEWS), which was a custom developed VE system [10][11]. Figure 12

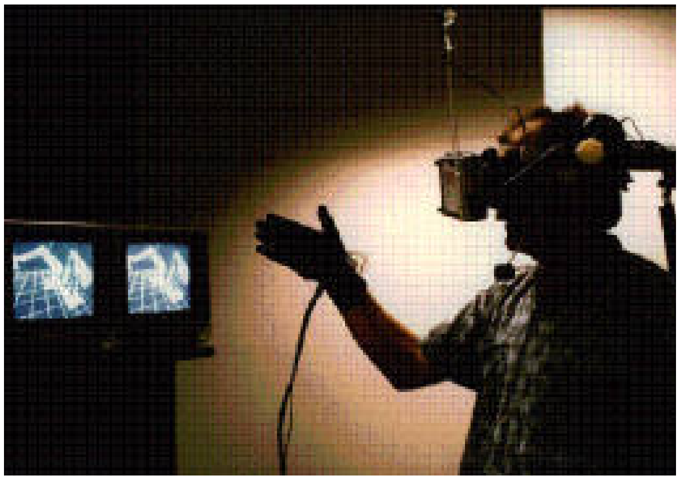

Figure 12 An operator in the VIEWS system controlling a Puma 560 manipulator arm. The operator's view is visible in the monitors to the left.

is a photograph of an operator in the VIEWS system. Although the Puma 560 we used was limited as a mobile platform, it did have high level control capability. High level control allows an operator to send a single command which accomplishes an entire task, rather than sending a stream of low level commands to accomplish the same task. It is much easier to send high level commands from within a VE interface, and this is what makes possible the compensation for communications time delay.

B. NOMAD and MEL

The first platforms used to develop the original version of VEVI were simple mobile bases used mainly for telepresence testing, in which the vehicle's primary purpose was to transport and orient cameras transmitting video streams back to the operator. These vehicles were rate controlled only, which is a very low level control interface. The first vehicle was NOMAD, which is a three-wheeled omnidirectional indoor vehicle. It is radio controlled from a host CPU, and carries a single video camera. The second vehicle was MEL, shown in Figure 13, which is a wheeled outdoor vehicle with an on-board CPU and a radio ethernet link to the operator. This vehicle has been the main platform for VEVI software development, as it has an on-board hardware architecture identical to the science field vehicles. These initial platforms had very limited high-level control capability, but allowed us to address the problems of positional uncertainty common with mobile platforms. Figure 14 shows the operator's view of the vehicle through the VEVI interface.

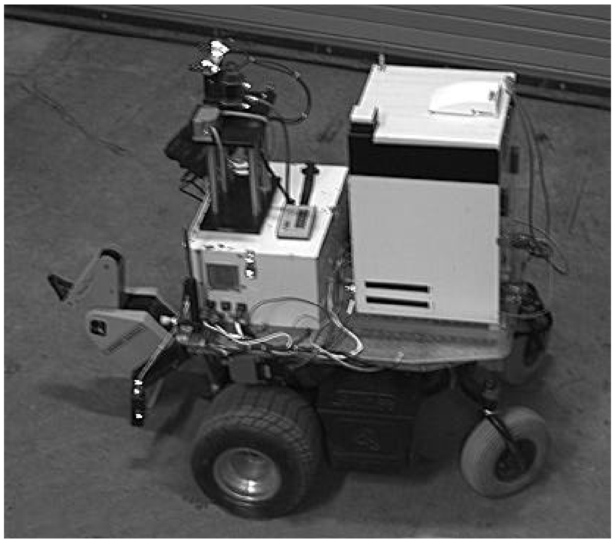

Figure 13 This vehicle, called MEL, is a sensor-rich mobile platform with on-board computation and a high bandwidth communications link to the ground control system. It is the main platform upon which MEL was developed.

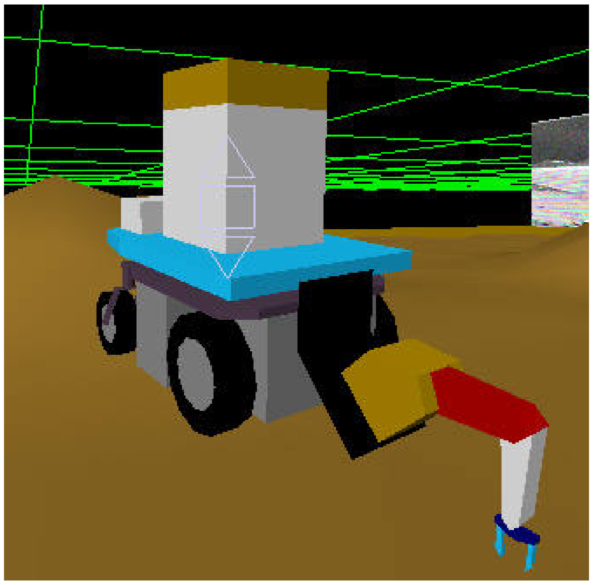

Figure 14 The operator view of MEL from within the VEVI 3D renderer.

C. HEAVENLY

The first vehicle controlled from VEVI which had true high level control capability was an air cushion vehicle called HEAVENLY, developed by the Stanford University Aerospace Robotics Lab (ARL) [8]. The ARL has developed the concept of task-level control of space robots, which makes it easier for human operators to command the robots to accomplish complex tasks. HEAVENLY is capable of performing tasks like station keeping at a commanded position, and flying to and grappling a free flying object. Figure 15 shows the VEVI renderer screen of the operator interface to the vehicle. The floater robot is to the left, and the target object to be grasped is to the right. Both objects ride on a cushion of air on a flat granite table to simulate the drag-free environment of space. The primary lessons learned from this experiment is that adequate sensing of the environment is critical, and high-level commands to the remote

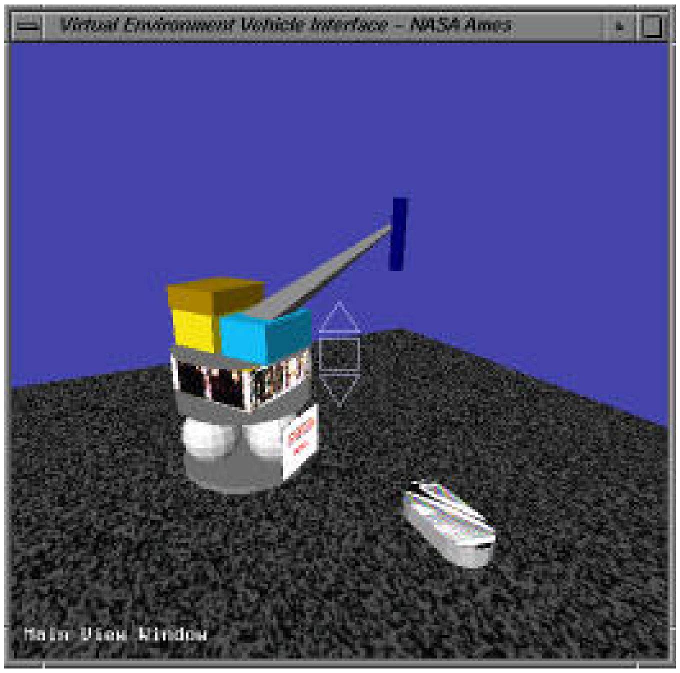

Figure 15 A screen image of the operator's view of the air-bearing floater robot HEAVENLY. This was the first robot to be controlled at a task level from within VEVI.

mechanism are much more efficient to send from a VE interface than low level commands.

D. Antarctic TROV

As mentioned previously, the most rapid advances in VEVI occurred as a result of using the operator interface to control science exploration vehicles in terrestrial science field trials. The first science field trail to use VEVI was the Antarctic Telepresence Remotely Operated Vehicle (TROV) Project [19][20]. In this project, a small robotic submersible was controlled under the sea ice in the Antarctic from NASA Ames Research Center. The vehicle was used to test whether telepresence, or the project of the human sensory apparatus into a remote location, was a viable operator interface for exobiology research. Although most of the detailed driving was accomplished through telepresence, VEVI was used to provide an enhanced situational awareness for the remote operators during the mission. The sensor stream from TROV was too sparse to allow a detailed model of the exploration area to be built in real-time. The terrain models used in VEVI were constructed from the sensor data later in the mission. VEVI was also used as a mapping tool for the science data returned by the mission. Figure 16 shows the TROV vehicle in the Antarctic during the mission, along with a screen image from the operator interface used during the mission shown in figure 17.

E. Dante II and Marsokhod

Two other recent missions to make use of VEVI are the Dante II mission to Mt. Spurr, Alaska and the Marsokhod Planetary Analogue mission to Kilauea, Hawaii. The Dante II mission interface is well described in Fong, et al. [21]. The main advances to VEVI accomplished for that mission was the addition of rapidly updated terrain maps from the vehicle's laser scanner, and the display of haptic information from the vehi-

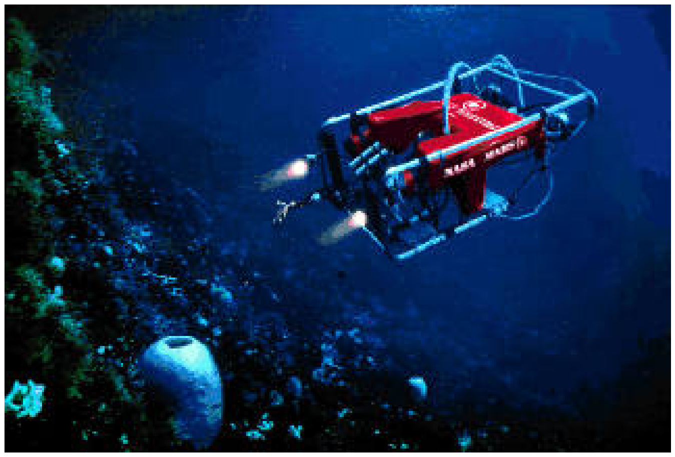

Figure 16 The Telepresence Remotely Operated Vehicle (TROV) under the sea ice in the Antarctic, under remote control from NASA Ames.

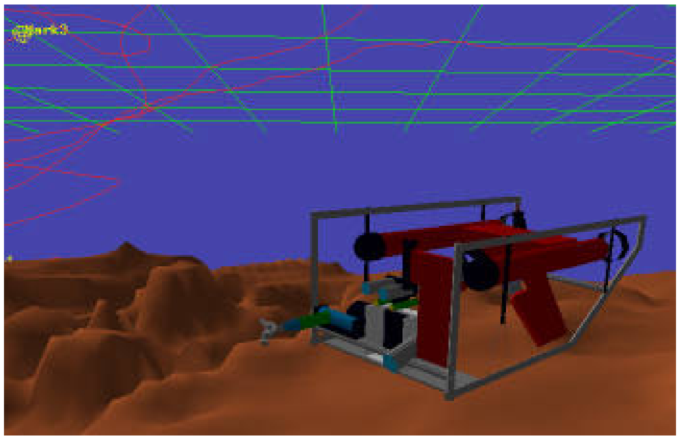

Figure 17 The operator's view of the VEVI 3D renderer during the Antarctic TROV mission. The vehicle is shown above a sensed terrain, with its telemetry tracks visible in the background. A science marker is visible at the upper left of the screen.

cle's legs. The Dante II vehicle also has the most complex articulation we have modelled in VEVI. Figure 18 shows Dante II

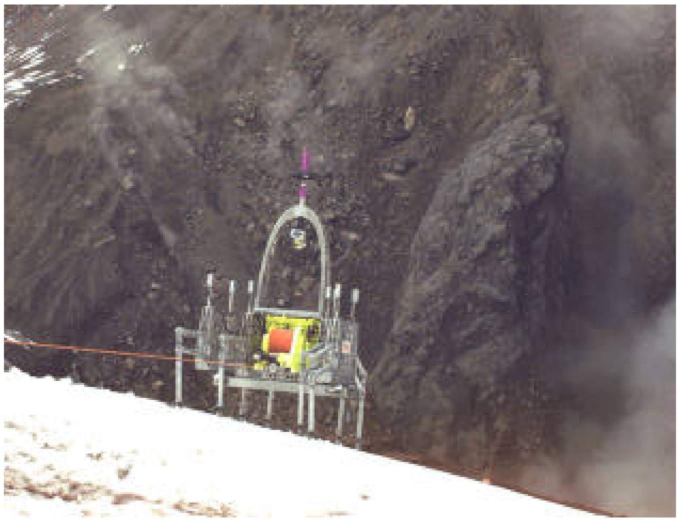

Figure 18 Photograph of Dante II at Mt. Spurr.

during the mission. The VEVI screen from the mission has been shown previously in figure 5. The sensor data of the terrain surrounding the vehicle arrived at a fast enough rate to be used to plan the vehicle's path and determine terrain clearance. This is the first mission in which the sensor stream from the remote site was rich enough that the vehicle could have been controlled entirely from within VEVI without auxiliary camera views.

We have performed several field tests of the Russian Marsokhod rover, to determine its operating characteristics for planetary missions [22]. The most recent Marsokhod Kilauea mission was the highest fidelity terrestrial simulation of a planetary rover mission we have done to date. The mission was organized so that separate Mars and Lunar science teams unfamiliar with the site except through high altitude photographs were given a science objective to perform with a remotely operated rover. The science teams and missions operations were conducted from a control center at NASA Ames, through a satellite communications link to the field site. VEVI was used to provide situational awareness during the mission, although the lack of task level control over the vehicle and the relatively sparse sensor stream limited our use of VEVI to compensate for communications time delay. Since the Kilauea mission, we now have task level control over the Marsokhod rover, and plan a future science field trial to complete the testing of VEVI for task level control of a planetary exploration rover. Figure 19 shows

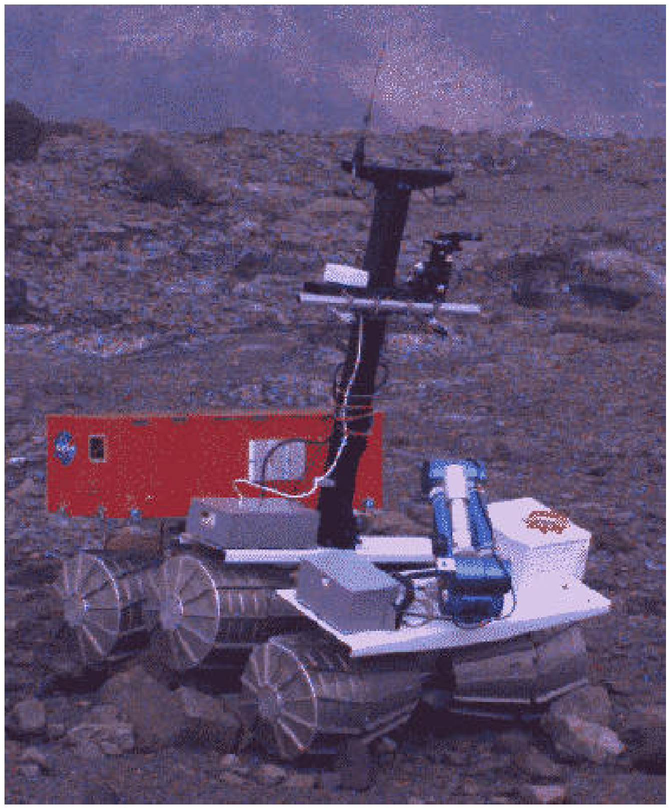

Figure 19 The Marsokhod rover at the Kilauea volcano in Hawaii during the planetary analog testing.

the Marsokhod rover during the Kilauea mission in March 1995, and figures 8, 10and 11 and show the operator interfaces used to control it.

VI. Drawbacks

A. Vehicle Sensor Dependence

The main drawback to our approach for remotely operating a vehicle in an unknown environment is the current dependence of the system on sensed information from the remote site. As stated above, most behaviors of objects in the renderer environment is a reflection of some state sensed by the remote mechanism and relayed back to the operator. This means that when the sensing is degraded for some reason, the virtual environment becomes less useful, and sometimes misleading. A potential solution to this is more rigorous notification to the operator of degraded information in the environment, and more reliance on predictive kinematic and dynamic simulations in the system. Although we do currently have the ability to mix the output of physical simulations with sensed state information, we have not yet relied on that capability very heavily during field missions.

B. High-Level Control

As mentioned earlier, the use of VEVI to control a remote mechanism in the presence of significant time delay becomes more efficient if the mechanism has on-board automation. Two local behaviors are particularly useful. The first is for the vehicle to maintain its own safety in the hazardous environment, through reflexive behaviors which avoid obstacles or halt the vehicle if hazards are sensed [23][24]. The second is for the vehicle to be able to execute high level commands from the operator. Rate control of a vehicle from within VEVI is relatively inefficient, whereas position or path control is much more efficient. One of the benefits of using a virtual environment to control a mechanism is that commands may be previewed and simulated in the environment prior to sending them to the vehicle.

C. Program Structure

This paper describes version 2.0 of VEVI. This version of the 3D renderer software evolved over time as our experience increased through our field trials. New features were grafted onto the original structure, which was not well designed for extensibility. We have come to the point with this version that adding or changing the structure often causes side effects elsewhere in the system. We have also been reluctant to make desired changes to the system, because of the effort necessary to support the changes under the existing structure. We have therefore decided to freeze this version at its current implementation, and rewrite the renderer and communications nodes, and change some of the file formats, with modularity and extensibility in mind from the beginning. The new system is already under alpha testing, and is described in Piguet [4].

VII. Summary and Conclusion

We have developed an operator interface system designed to support the remote operation of complex science exploration mechanisms in the presence of substantial communications time delay. This system, called the Virtual Environment Vehicle Interface, uses a highly distributed communications architecture and commercial software products to implement a virtual environment operator interface tool. Our experience to date with VEVI in terrestrial science field experiments using remotely operated vehicles has demonstrated that an operator interface based on virtual environment techniques can improve the situational awareness of the operator, and can convey a large amount of information about the vehicle and its surroundings quickly and concisely to the operator. If the VEVI ground operations system is coupled with a vehicle with sufficient software automation to allow it to respond to task-level commands and keep itself safe while executing those commands, then the efficiency of the mission as measured by the amount of science performed per command cycle to the vehicle is much improved over conventional control approaches.

Acknowledgments

The development of VEVI was supported by the National Aeronautics and Space Administration under NASA RTOP 233-02-04 (D. Lavery, program manager) and the NASA Ames Director's Discretionary Fund.

References

[1] Ellis, S.R., "Nature and Origins of Virtual Environments: A Bibliographical Essay," Computing Systems in Engineering, Vol. 2, No. 4, 321-347, 1991.

[2] McGreevy, M., "Virtual Reality and Planetary Exploration," 29th AAS Goddard Memorial Symposium, March 1991.

[3] Ellis, S.R., "What Are Virtual Environments?," IEEE Computer Graphics and Applications, pp17-22, January 1994.

[4] Piguet, L., Fong, T., Hine, B., Hontalas, P., and Nygren, E., "VEVI: A Virtual Reality Tool for Robotic Planetary Explorations", Virtual Reality World 95, Stuttgart, Germany, February 1995.

[5] Ferrell, W.R., "Delayed Force Feedback," IEEE Trans. Human Factors in Electronics, 449-455, October 1966.

[6] Sheridan, T., "Telerobotics, Automation, and Human Supervisory Control", MIT Press, Cambridge, MA, 1992.

[7] Miller, D.P., "The Real-Time Control of Planetary Rovers Through Behaviour Modification," in the Proceedings of the 1990 SOAR Conference, Albuquerque NM, June 1990.

[8] Ullman, M.A., "Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-flying Space Robots," Ph.D. Thesis, Stanford University, Dept. of Aeronautics and Astronautics, Stanford CA, March 1993.

[9] Stevens, H.D., et. al. "Object-Based Task-Level Control: A Hierarchical Control Architecture for Remote Operation of Space Robots", AIAA/NASA Conference on Intelligent Robots in Field, Factory, Service, and Space, Houston, TX, April 1994.

[10] Fisher, S.S. et al., "Virtual Environment Display System," paper presented at ACM 1986 Workshop on 3D Interactive Graphics," held in Chapel Hill, NC, 1986

[11] Jacoby, R.H., and Ellis, S.R., "Virtual Menus," Proc. SPIE Technical Conf., Vol. 1666, SPIE, Bellingham WA, 1992.

[12] Lin L., Simmons R., and Fedor C., "Experience with a Task Control Architecture for Mobile Robots," Internal CMU-RI-TR- 89-29, Robotics Institute, Carnegie Mellon University, 1989.

[13] Fong, T. "A Computational Architecture for Semi-

autonomous Robotic Vehicles", AIAA 93-4508, AIAA Computing in Aerospace 9, San Diego, CA, October 1993.

[14] WorldToolKit Reference Manual, Sense8 Corp., Sausalito CA, 1993.

[15] Fedor, C., "TCX: An Interprocess Communications Systems for Building Robotic Architectures", internal document, Robotics Institute, Carnegie Mellon University, January 1994.

[16] Wise, J.D., and Ciscon, L., "Telerobotic Interconnection Protocol," Rice University, 1990.

[17] Schneider, S.A., Ullman, M.A., and Chen, V.W., "ControlShell: A Real-Time Software Framework," Real-Time Innovations, Inc., Sunnyvale CA, 1992.

[18] Piguet, L., "General Kinematics Simulation of Serial Links Mechanisms," M.S. Thesis, Micro Engineering Dept., Swiss Federal Institute of Technology, Lausanne, Switzerland, February 1994.

[19] Hine, B., et. al., "The Application of Telepresence and Virtual Reality to Subsea Exploration", The 2nd Workshop on Mobile Robots for Subsea Environments, Proceedings ROV '94, Monterey, CA, May 1994.

[20] Stoker, C., Barry, J., Barch, D.R., Hine, B.P., "Use of Telepresence and Virtual Reality in Undersea Exploration: 1993 Antarctic Telepresence Experiment," Proceedings of the AAAI Workshop on AI Technologies in Environmental Applications, Seattle WA, July 1994.

[21] Fong, T., Pangels, H., Wettergreen, D., Nygren, E., Hine, B., Hontalas, P., and Fedor, C., "Operator Interfaces and Network-Based Participation for Dante II," SAE 25th International Conference on Environmental Systems, San Diego, CA, July 1995.

[22] Garvey, J., "A Russian-American Planetary Rover Initiative", AIAA 93-4088, AIAA Space Programs, and Technologies Conference and Exhibit, Huntsville, AL, September 1993.

[23] Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," IEEE Journal on Robotics and Automation, vol RA-2#1, March 1986.

[24] Gat, E., "Reliable Goal-directed Reactive Control for Mobile Robots," Ph.D. Thesis, Virginia Tech, May 1991.