AIAA Infotech@Aerospace 2007 Conference and Exhibit
7 - 10 May 2007, Rohnert Park, California

AIAA 2007-2971

# A Virtual Rover Interface for Collaborative Human-Robot Exploration Teams

Catharine L. R. McGhan[1] and Ella M. Atkins[2]
*University of Michigan, Ann Arbor, Michigan, 48109*

**Future planetary exploration missions will employ teams of rovers as astronaut assistants and independent scouts. Individual and cooperative tasks must be accurately modeled. Mission controllers and astronauts must maintain situational awareness through global and local perspectives of the exploration team members and their environment. This paper describes the development and integration of a mixed-reality graphical interface that provides a series of real-time three-dimensional views of the team. Astronaut and rover activities are rendered on an environment map with a variety of negotiable global views and local perspective views corresponding to onboard video acquired by each rover. The virtual reality (VR) graphics system described in this paper is integrated within a planner/scheduling framework that also supports real astronauts and rovers, enabling the overall team to be composed of real and virtual entities. VR graphics are combined with a set of menu interfaces that facilitate the specification of goals, commands, and constraints through the same environment in which the graphics are presented. Update rates (computational overhead) for the VR software are evaluated as a function of model fidelity and kinematics computations, providing a measure of practicality for this system as a tool for mission controllers and for suited astronauts with limited perceptual capabilities.**

## I. Introduction

Future planetary exploration missions will require collaborative teams of rovers and astronauts for scientific exploration and habitat construction/maintenance activities. Rovers may act as astronaut assistants, independent scouts, or "equal" collaborators with each other and their astronaut companions. Maintaining situational awareness is challenging in an extraterrestrial environment. No infrastructure exists beyond what has been constructed or left behind from precursor missions, implying resources (power, fuel, oxygen) are highly constrained. It is imperative that missions be highly efficient (optimized) to minimize resource usage but missions also must be highly adaptable since the environment and human/robot performance may be uncertain particularly for early missions. Maintaining situational awareness throughout a planetary surface extraterrestrial vehicular activity is challenging but necessary from both productivity and safety perspectives. This paper describes the development and integration of a mixed-reality graphical interface that provides an operator a series of three-dimensional views of the team. By default a full overhead map of the exploration region is provided, with the option for an operator to graphically negotiate their 3-D viewpoint and perspective to a set of fixed or dynamically-negotiated coordinates. Simulated views from each rover also enable assessment of the rover's ability to visually detect and navigate around/over obstacles. Our virtual reality (VR) graphics system is integrated with an automated planner/executive system developed in previous work,[1,2] enabling the overall system to build, execute, visualize, and update plans for large-scale simulated and physical astronaut-rover teams. The current rover entities are based on a six-wheel physical rover with rocker-bogie suspension that will fuse sensor feedback and kinematically-generated entity positions into the VR display and planner/executive world models.

To facilitate rapid prototyping, maximize extensibility, and ensure portability, the VR system has been implemented in a Java/Xj3D environment using the VRML scripting language.[3] Virtual rover passive and active joint motions are modeled with a simplified kinematics model adequate for visualization and efficient for replication and real-time display by the graphics engine. This VR interface connects to the planner/executor through standard sockets, providing virtual entity state estimates and operator commands while receiving 'real' state data from the environment and task execution directives/status across all agents. VRML simplifies the environment and mobile

[1] Graduate Fellow, Aerospace Engineering Dept., University of Michigan, Ann Arbor, MI, 48109, Student Member.
[2] Associate Professor, Aerospace Engineering Dept., University of Michigan, Ann Arbor, MI, 48109, Associate Fellow.

entities within this environment through a flowdown model that translates kinematics to entity positions/orientations on the graphics display(s).

The creation of a VR interface to facilitate research and rapid development is not a new concept, but the use of standard VR languages to facilitate creation of VR environments is not yet commonplace. One aerospace-related VR example is VEVI.[4] In this environment, a human teleoperated a remote rover using a virtual environment including kinematic and dynamic models of the rover system. Subsequent development split along two tracks. One produced a next-generation system, Viz,[5] that focuses on VR architecture in the context of coverage planning and map building tools to control multiple physical rovers in a Mars testbed environment.[6] The second track[7] explores human-robot interface standards, with teleoperation of robotic platforms as a target application. Our paper focuses on scalability and portability of an analogous VR graphics environment, but with emphasis on a "plug-and-play" structure that enables a mix of physical and virtual robotic and astronaut "agents" to coexist in a single execution cycle. An automated planning/scheduling system interfaces with the VR system to allocate individual and collaborative tasks across the astronaut and rover agents according to their abilities and resources. Although use of such an automated tool for scheduling astronaut tasks remains controversial due to the perceived rigidity of such tools, this strategy has the potential to better enable astronauts and missions controllers to focus on science activities as well as to identify anomalies (potential problems/dangers) and opportunities.

This paper describes a *Virtual Rover Interface* (VRI) facilitates real-time collaboration between one or more human agents and one or more rover systems. The VRI is coupled with the planner/scheduler, agent simulation software, and physical hardware (when available) to provide graphics-based situational awareness and a multi-functional command interface. This VRI system has both simplified and high-fidelity entity models appropriate for *in situ* astronaut and remote mission controller use, respectively. Rovers are classified as either "real" physical agents that maneuver in indoor and outdoor test environments, or virtual rover agents that exist solely in simulation and have no physical impact on the world. "Astronaut" entities can also be physical or virtual agents. At runtime, the VRI creates a set of virtual rovers and astronauts. The subset of physical rovers and astronauts are linked to a corresponding subset of VRI rovers/astronauts. For physical entities, the VRI defaults to displaying rover state based on state feedback from the physical rover, effectively "slaving" the virtual rover to mimic the corresponding physical rover. If an astronaut assumes direct control of the physical rover, however, the virtual rover becomes the master, issuing motion commands to the physical rover. In this sense, the VRI acts as a "switch" to direct data between physical and virtual rover entities, controlling operating modes based on user (or planner/scheduler) directives.

Figure 1 shows the VRI architecture. The planner and coordination executive communicate with the VR environment to distribute schedules across the team that accomplish as many tasks as possible given resource constraints. Once specified, an initial (potentially optimized) plan is communicated to all agents and real-time operations begin. In the event of an anomaly or opportunity that prompts replanning, the VR world environment communicates with the executive to expediently respond. The planner, coordination executive, and associated physical rover code base have been developed in parallel work.[1,2] This paper focuses on the development of a real-time reactive VR environment, visually-accurate rover and environmental models, and a communications protocol that acts as "middleware" between the coordination executive and rover agents. This paper begins with an overview of VR and classification of this research within traditional VR application paradigms. Next, the VRI software architecture is discussed in detail, focusing on the world model as well as the rover and environment submodels. Results VRI software benchmark tests are presented along with ongoing efforts to expand the current VRI implementation and deploy the VRI in realistic large-scale test scenarios.

## II. VR Background

Virtual reality (VR) is most generally defined as a three-dimensional (3D) computer-simulated environment. Some definitions also mandate full immersion for "true VR", where senses beyond only sight are stimulated with virtual cues. Realism is achieved through detailed sensory models and entity motions as well as physically-accurate real-time response to user input. Originally, the idea of virtual reality research was to find a way to simulate the physical world as closely as possible, in such a manner to trick a human's brain into being unable to tell the difference between reality and the virtual world. This tends to raise ethical questions and dramatizations of possible misuse of such technology (e.g., movies such as "The Matrix"). Researchers have developed 3D viewing technologies such as head-mounted displays (HMDs), head-coupled displays (HCDs), CAVE (Cave Automatic Virtual Environment), and table projection systems, along with their associated input devices and other methods of interaction with the user in a virtual environment.[8] However, as research has progressed, the actual application of VR in areas ranging from architecture and computer-aided design (CAD) to medicine, gaming, and training simulations, has prompted the question of whether full immersion is necessary, or even wanted, in all applications.

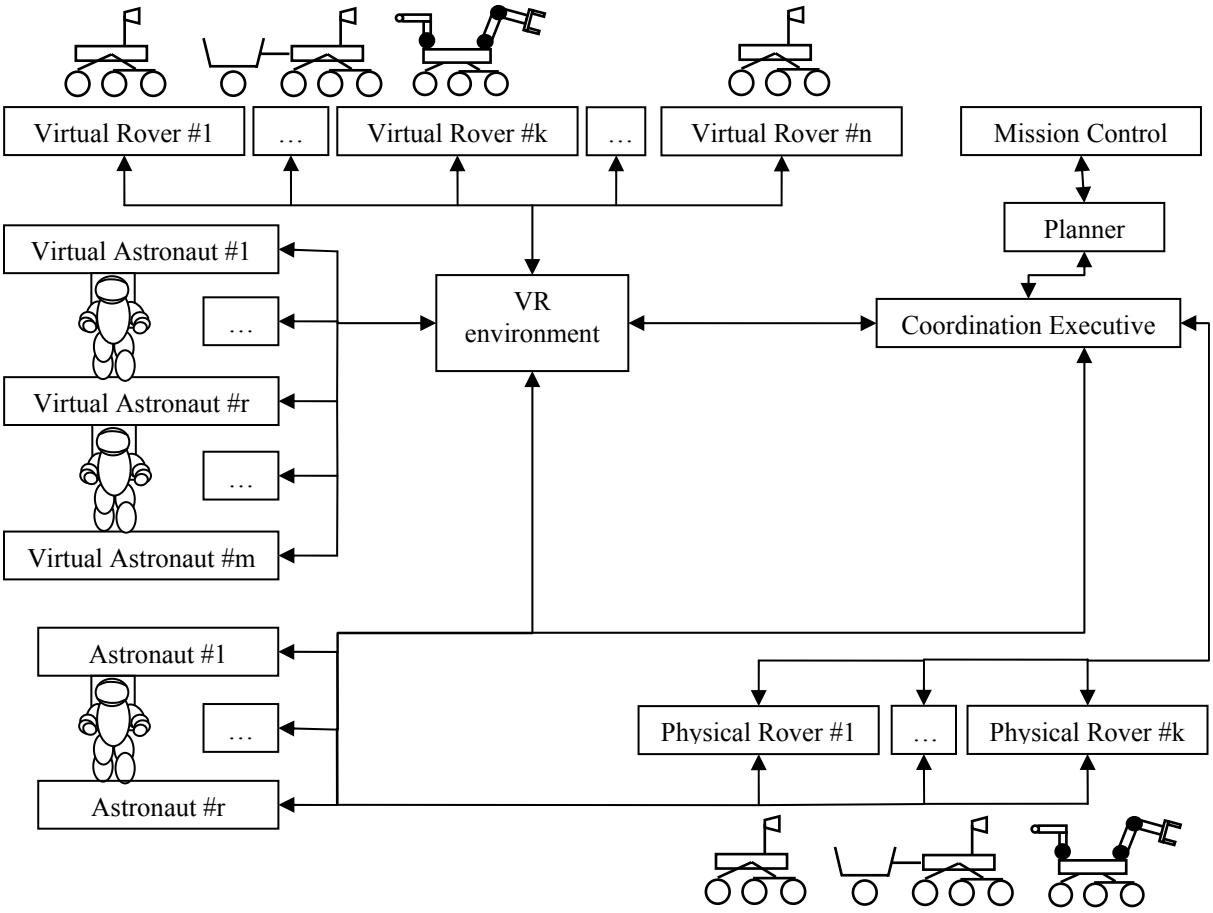



**Figure 1. Astronaut-Rover Collaboration Architecture**

In many cases, display on a large overhead screen or even a personal computer is sufficient, such as in low-mid range flight simulators. Because of this, current research tends to fall into one of two categories – one involving the development of hardware for immersion in virtual environments, and the other more focused on the software that supplies the virtual environments to be displayed. On the software side, the growing demand for a universal way of representing VR environments has led to the creation of the older VRML97 and current X3D scripting language ISO standards.[3,9] Many open source players and development kits support them. One example is Xj3D,[10] a Java toolkit and browser that allows direct internal access to virtual scenes in VRML and X3D.

Since space missions are costly and most ground testing and training is a simulation of conditions in space, using VR to supplement ground testing had been a natural extension of normal operational development. Although limited hardware testing is essential to develop skills such as path planning, navigation, and physical task execution capabilities (e.g., sampling), developing a VR simulation for early mission prototyping is faster, easier, and cheaper than developing hardware test platforms, especially when the mission design criteria are in still in flux. This is particularly the case when extending the research to study many rovers working with multiple astronauts. Also, from an operations standpoint, a well-designed VR interface can also communicate information more intuitively and flexibly than static indicator panels and simple text displays. VR applications typically fall under one or more of the following classifications.

*1. Simulation*

Simulation is the most basic VR, where the user passively views a preselected image or sequence of images, and the environment does not react. This is most commonly used for entertainment or educational purposes. Computer-animated movies (e.g., IMAX) are one example. Another example with user interaction is NASA JPL's Solar System Simulator, which creates static images of heavenly bodies for which the user may specify targets, viewpoints, and viewing times before running the simulator.[11] The VRI system described in this paper includes the

capability for playback of a saved scenario, which is done by creating solely virtual rover and virtual astronaut agents and supplying them explicitly with the data from a previous execution cycle.

*2. Teleoperation/Telepresence*

Teleoperation and telepresence allow a human to control a robotic system remotely but supply additional information such as overlaid or enhanced video, explicit tasks (commands), or simulated over-the-shoulder views. The addition of haptic controllers can also allow direct tactile force-feedback that is processed by the human more quickly and accurately than is possible through visual displays alone. Teleoperation (with VR) can allow astronauts to control systems like Robonaut from inside a safer pressurized environment or other remote location rather than complete tasks on a higher-risk extravehicular activity (EVA). The addition of graphics-based VR augments visual awareness thus by extension improves control of the robotic system.[12] Another good example of a flexible VR system for teleoperation is VEVI, which has been utilized on multiple robotic systems, including the underwater robot TROV, the 8-legged Dante II crawler, the 6-wheeled Marsokhod rover, and the Ranger robotic arm.[5] Teleoperation is the lowest level of reactive operation available in the VRI. In this mode, an astronaut or mission operator directly drives a virtual rover and the corresponding physical rover is slaved to it, provided such a physical rover exists.

*3. Human/Astronaut Training*

VR for training purposes typically utilizes a mix of both simulation and teleop/resence. The user interacts with an active simulation that changes in real time according to the user's input. Flight simulation in fully immersive environments helps train pilots/astronauts how to handle any number of worst-case scenarios they might face on-orbit while still on the ground under safe conditions. Also, since the VR environment is virtual, the computer system can log each user's response times and choices, enabling assessment of overall performance as well as quantitative identification of the user's specific strengths and weaknesses. Training programs can be adapted according based on individual needs, or the programs can be repeatedly executed with no or minimal (tailored) changes. VR was used during astronaut training for the Hubble servicing missions as early as 1994.[13] Another Aerospace example involved parachute mission planning, where multiple users trained together in the same virtual environment under varying scenarios, including differences in altitude, landing site, and wind speeds.[14] Our VRI could be adapted to user training as well as supporting situational awareness enhancement and multi-agent mission simulations.

*4. Force Multiplier*

As a hybrid training simulation and system performance evaluation tool, "force multiplier" VR programs enhance human oversight capabilities, real-time or delayed, for one or more robotic systems. The typical goal is to enable a single operator act more capably through sensory augmentation than would be possible otherwise. This human oversight could take the form of command and control, or it could involve interaction or collaboration with the robotic systems, virtual or physical. One example of this application class is an offline planner supplemented with VR that helps NASA scientists plan daily robotic missions for the Mars rovers, where time-delay is high and the environment and rover system's dynamics are well-known.[6] In its fully-operational mode, the VRI connects to both virtual and physical astronaut and rover agents. Under nominal conditions direct teleoperation is not used by the astronaut(s) – the rover agents are able to create trajectories and execute simple tasks without astronaut intervention. The primary distinction of our VRI from previous work is its bias toward "force multiplication" in multiple dimensions. Certainly, the VRI can enhance situational awareness for the user. It more literally can "multiply forces" through virtual entity (astronaut and rover) replication to supplement the set of available physical entities. This enables both the VRI interface and our automated planning/scheduling system to be evaluated through real-time coordination of commands and activities across a relatively large team.

VR languages and protocols are common across all the above application classes. Given real-time constraints, there is also typically a split between simplified and high fidelity models. Many applications can benefit from inclusion of both types. For instance, a rover mission using the VRI could utilize a highly simplified model to provide an onsite operator/astronaut real-time data, while a high-fidelity model would be used to enable mission controllers/scientists to accurately perceive detailed activities and the environment either in online or offline scenarios.

## III. VR Architecture

Our VRI software was developed using the VRML97 and X3D industry standards.[3,9] The virtual rover and environment models were created in 3DStudioMax and exported in VRML format. Early versions of the VRI were designed entirely in VRML and vrmlscript (the VRML-standard version of Javascript), primarily due to the low learning curve for development, the availability of freeware VRML players, and the ease of separating code (modularity) and "cloning" models and subsystems (object-oriented design) using the Inline and EXTERNPROTO nodes.[15] The newest version of the VRI is being ported to the X3D standard, since it has better support of externals (the VRML EAI was not well-implemented in any standard VRML player/browser, while the X3D SAI is generally both better developed and better supported). Xj3D is being used as the toolkit of choice due to the need for platform-independent development and support for socket layer communications which Java handles well. Xj3D was also chosen because it is currently one of the most advanced open source X3D toolkits and is being developed in parallel with the evolving X3D standard.[10] Figure 2 provides an overview of the components and communication links present within our VRI software.

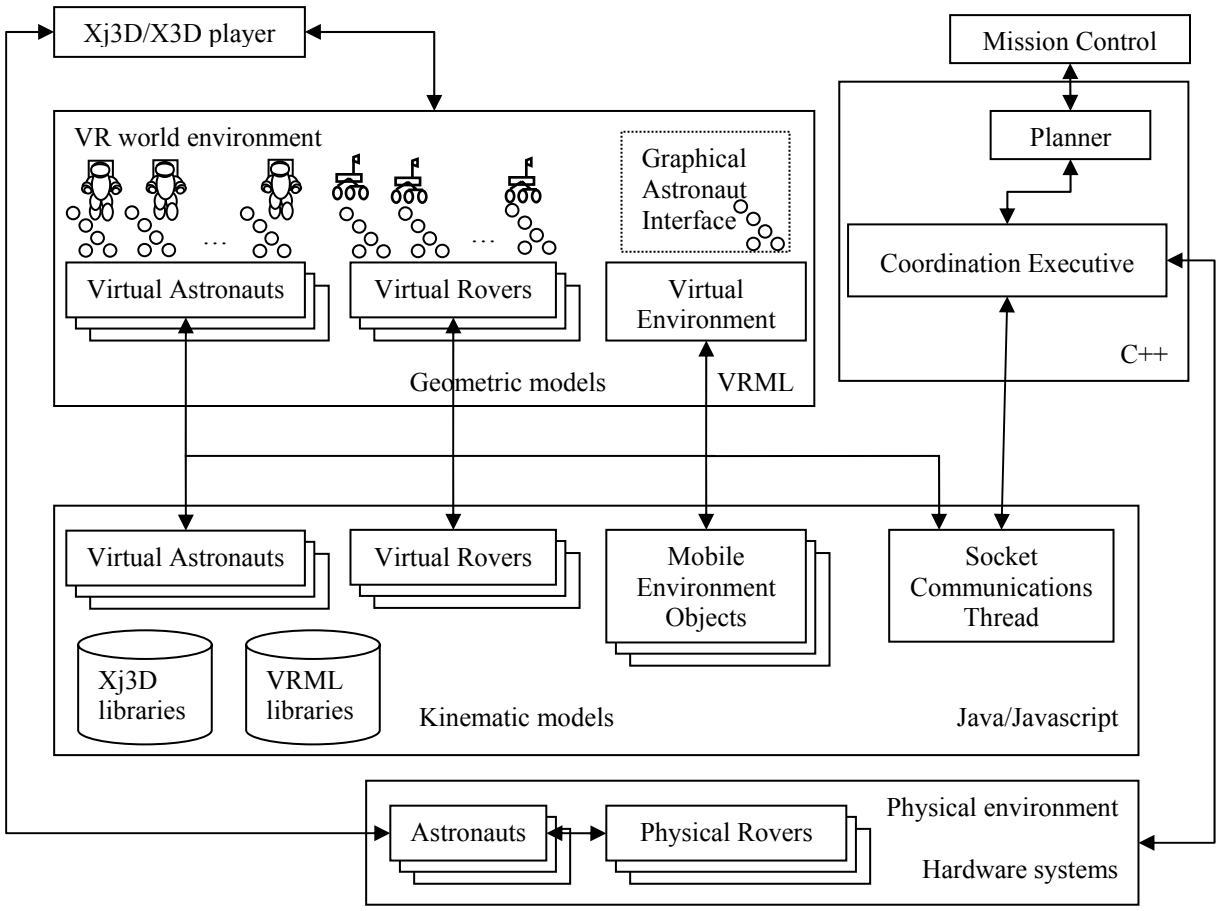



**Figure 2. VR Software Modules and Data Flow**

The Figure 2 VRI software system has four main components. A symbolic (AI) task planner and coordination executive, coded in C++, were developed in previous work.[1,2] The four components developed in this work include: (1) the virtual rover, environment, and astronaut geometric models coded in VRML, (2) kinematics models of the virtual agents (rover and astronaut), (3) dynamic environmental factors that determine the various virtual agent reactions and other changes in the environment, coded in Java, and (4) the socket communications thread, also coded in Java. The Java code is compiled with the necessary Xj3D and VRML libraries using the free Borland JBuilder 2005 Foundation software development kit. The VRML geometric models, as well as the compiled Java code, are subfiles referenced by one VRML global world file. This global world file, when run in a capable X3D browser, initializes the environment and rover models and starts the Java executable, which then initializes a

connection to the global world model and runs parallel to it. The Java executable evaluates the kinematics of the various models and communicates updates to the VRML global world model in soft real time, while keeping track of the current state of those models internally. The individual instantiations of the kinematics models – the Java classes – interact with as necessary and communicate with the coordination executive software via the socket communications thread. The Graphical Astronaut Interface (the user's current view of the global world model, which includes all virtual agent and virtual environment models, plus additional VRML code to support various operational modes) is displayed in the Xj3D player/browser, and updates accordingly. Under nominal conditions, the planner transmits a plan to the coordination executive, and the coordination executive tells the physical and virtual agents what tasks to accomplish and when. The physical agents return state information to the coordination executive, which relays this information through the communications thread to update the corresponding virtual agents. The purely simulated virtual agents send state data to the coordination executive through the socket thread, and update the VRI world model automatically. During off-nominal execution periods, entity states still evolve as normal, but the planner/scheduler (autonomously or with human controller assistance through the VRI) updates the task specifics (e.g., waypoints) and schedules over all virtual and physical entities. In the event that an astronaut elects to teleoperate a rover manually, the astronaut communicates this intent via the GAI, which forwards the request to the coordination executive. The coordination executive then handles the mode change between the virtual rover being slaved to the physical rover's movements and the physical rover being slaved to the state of the virtual rover, now to be commanded via the GAI.

With this paradigm, the planner treats all agents – physical and virtual astronauts and rovers – similarly when designing and evaluating plans for task completion: they are modeled as "resources" with differing abilities. Some overriding operational requirements, such as safety of the astronauts coming first, are given in the planning software, but these requirements could be specified for any particular agent. Astronauts are thus able to redirect goals/activities but can also be "directed" by the planner/scheduler such that their attention can be focused on task execution rather than team management.

### A. VR World Model

Figure 3 shows a snapshot of the default view of the current GAI interface, running under Firefox 2.0.0.3 with the Cosmo Player 2.1.1 (VRML) plugin. This example global model file includes both static and mobile environmental objects, a console for basic teleoperation by an astronaut operator (one mode of operation), and two virtual rover agents. The virtual rover agent shown in the center of the screen is configured to be teleoperated using the console buttons and sliders along the edges of the screen; the second rover agent (shown near the ramp on the right) has pre-scripted motions that run when the rover is left-clicked.

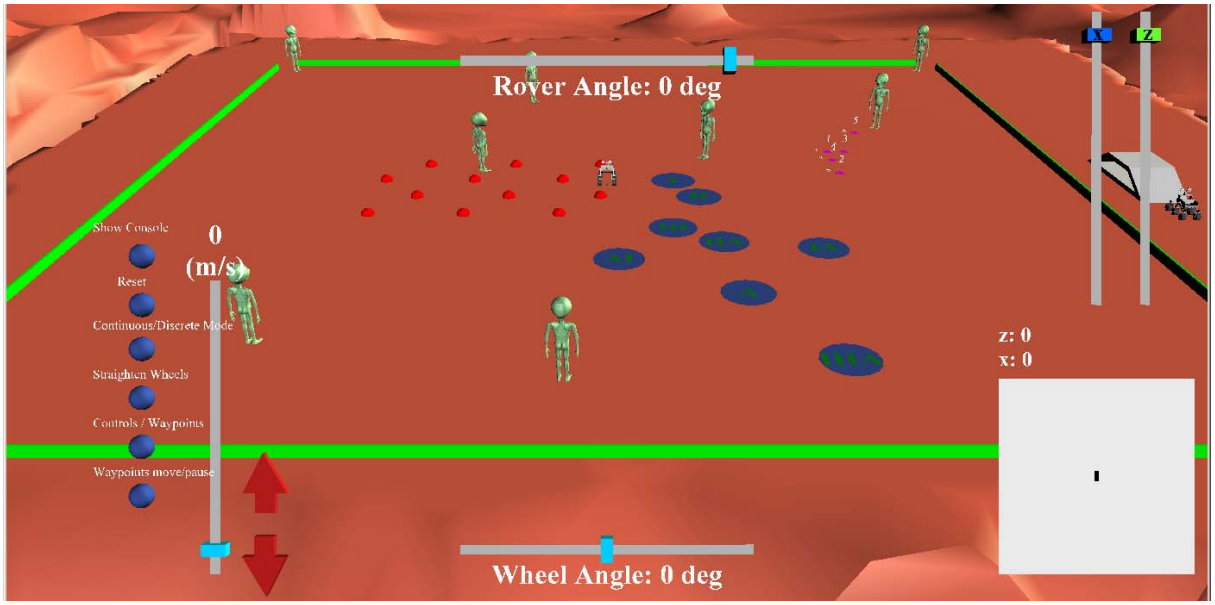



**Figure 3. Default View of the VR Graphical Astronaut Interface**

The illustrated environmental objects include static objects such as red spheres representing obstacles that the virtual rover agent must traverse, green traversal "boundaries", and "Martians" (modified from a preexisting file, see Ref. 16) in the center of the field. Mobile objects are environmental objects that may change over time. The

interface displays objects of this class with designators such as Roman-numerated circles, called waycircles, that respond to the rover by changing color when the rover passes over them. The numbered pink waypoints are mobile objects that can influence the rover agent. Collision detection occurs in the rover kinematics model, and if a collision event occurs, this information is passed to the colliding entity's corresponding physics model. Thus, mobile environmental objects are treated in the same way as "active" virtual agents, although their movement is only prescribed by simple physical models, rather than by sensor data or path planner directives received by the coordination executive.

All of the models in the world file are currently made from simple geometric primitives placed in the scene via multiple, consecutive transformations of the local coordinate frames fixed to the objects. In the default view in the GAI, the right-handed (world) inertial (*N*) coordinate frame's x-,y-, and z-axes point to the right, up, and out of the display, respectively. Each rover is assigned a "centroid" frame (*R*), the position and orientation of which are defined with respect to *N*. Figure 4 shows an overhead view of the rover before and after a local traversal.

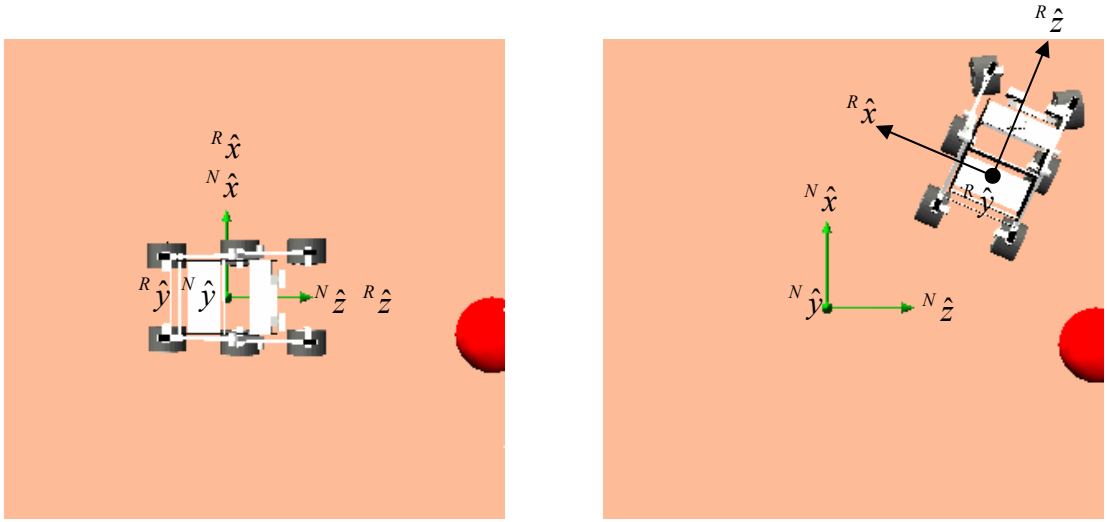



**Figure 4. Inertial and Local Coordinate System Example, Overhead View**

When a rover moves, the kinematics for that rover system is computed in Java, and the corresponding virtual rover model in the VRML global world model is updated. This information is inserted into the world model by way of updating the location of the local *R* frame, and all of the dependent virtual rover model components are then automatically re-rendered and displayed to screen by the X3D player software. Figure 5 illustrates the VRML script representational hierarchy.

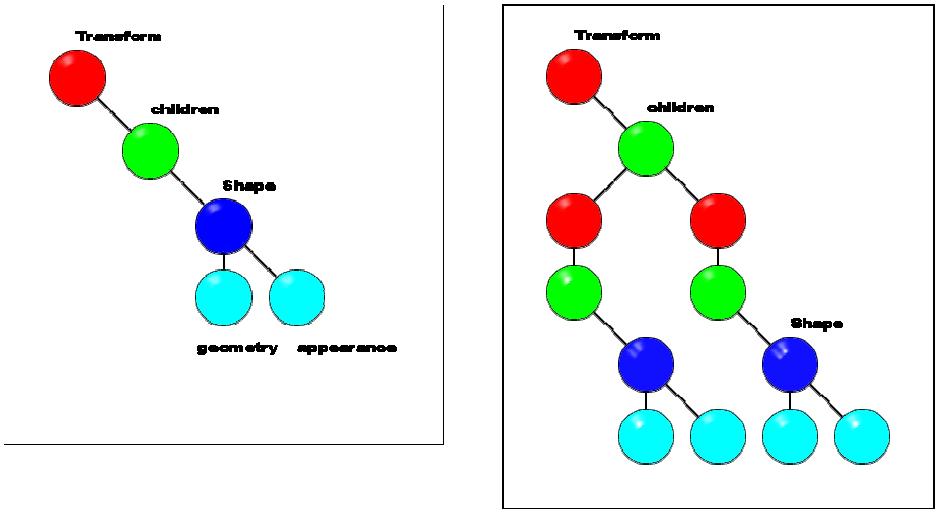



**Figure 5. VRML Hierarchy Examples (from Ref. 17 and 18)**

There are four basic types of VRML model components. *Nodes* describe objects (such as Shape in Figure 5), actions, and locations, while *fields* describe node functionality (such as geometry, appearance). *Events* support information access to and from "active" nodes (such as timers or proximity sensors). *Routes* are the pipelines created to shunt data between events. *Transform* nodes are wrapped around subsets of VRML nodes, and hold *translation* and *rotation* fields that apply a transformation to every subnode. *children* fields allow several object nodes to be grouped into a "super-object" node. An example would be assembling several snowballs in a specific configuration – say, on top of each other – to create a snowman. To do this in VRML, as shown in Figure 5, both Shape nodes would be set as *Sphere*s, with certain radii and white coloring picked. The *children* fields would neatly separate the *Shapes* from one another (and allow more shapes, such as black spherical buttons or an orange cone nose, to be added later), and the *Transform*ation nodes directly above them would hold the offset vectors necessary to stack the "snowballs" one on top of each other. The higher *children* field would allow the two branches of nodes to be grouped together, and the *Transform*ation at the top of the tree would arbitrarily move and rotate the entire snowman however we wanted in the world environment, relative to the inertial frame. This flowdown model of updating allows us to only update the one highest-level transformation to move an entire object, reducing computational overhead in Java, since this is handled automatically by the VRML player when the geometric models have been correctly constructed. For instance, if a robotic arm model was created from the base to the tooltip, top-down in the tree, changing the location of or rotating the base at the top of the tree would automatically propagate down the tree, enabling the VRML simulator to perform forward kinematics automatically. However, specific translation of a manipulator tooltip still requires solution of the inverse kinematics problem then explicitly specifying all joint motions for VRML.

Collision detection between objects (such as the rover and a spherical obstacle) must be done in the kinematics models rather than VRML. In order to reduce computational overhead, the idea of a flowdown model is applied to collision detection for our VRI, in the form of tiers of increasingly computationally intensive collision checks before completing a full object collision analysis. Figure 6 shows pseudocode for the testing procedure; it is assumed that only one object intersects the wheels at a time.

```
for i = 1 to number of obstacles
    if obstacle i is within rover footprint
        if obstacle i intersects with 1 + of the rover wheels
            calculate the rover roll angle and offset to R frame
        end
    end
end
```

**Figure 6:** Collision Detection Algorithm.

At the first level, computations are done to check whether the obstacle is sufficiently close to the rover that it might intersect with one of the rover wheels. If intersection is a possibility, then more computationally-intensive checks are done to determine whether one or more rover wheels intersect the specified obstacle. If an intersection is found, then the kinematics reactions of the rover to the obstacle are computed. By creating this hierarchy of system checks, many unnecessary calculations are avoided that would otherwise be computationally expensive.

### B. Rover Model

Entity models were defined to maintain adequate computational efficiency while providing realistic component motions. The rover geometric model, coded in VRML, was created in 3DStudioMax using measurements of the physical rover system. Whenever possible, model components were made with the simplest primitives, so that the computations necessary to display the 3D model in VRML would be minimized. Figure 7 shows a picture of the physical rover system along with the virtual rover model. The virtual rover geometric model has local coordinate frames corresponding to joints of the physical rover to facilitate quantification of the VRI system with "real" versus strictly simulated virtual rover agents.

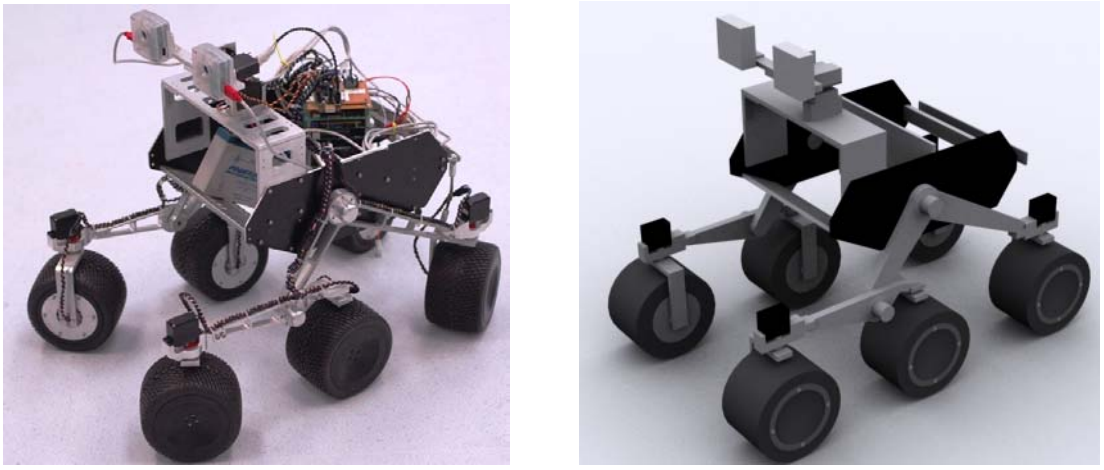


**Figure 7. Physical Rover and Virtual Rover**

The rover kinematics model, coded in Java, provides simulated motion data to the coordination executive and updates all entity coordinate frame locations and orientations for the VRML-generated graphics. Due to the object-oriented architecture, any kinematics model can straightforwardly be replaced with any other kinematics model. The current rover model is the focus of this section; astronaut and mobile environment entities (e.g., rocks) are defined analogously.

Rover kinematics is represented through two equation sets. The first describes motion in terms of the 2D bicycle equations, adequate for macro-level runtime scenarios since further fidelity would not alter component motion perceptibly in the VR graphics. Note that this approximation also has minimal impact on energy/fuel use computations, particularly given relatively smooth and consistent terrain. Illustrated in Figure 8, this model assumes the back (bicycle) tire is the instantaneous center of rotation at all times and that no sliding occurs at the rear tire location. This allows straightforward estimates of front and back/rear wheel velocities, $\vec{v}_F$ and $\vec{v}_B$, respectively:

$$\begin{aligned} |\vec{v}_F| &= u \\ \vec{v}_B &= (\vec{v}_F \bullet \hat{e}_r)\hat{e}_r \end{aligned} \tag{1}$$

The simple 2D lateral plane equations of motion are then specified as follows in the discrete form that directly translates to the Java implementation:

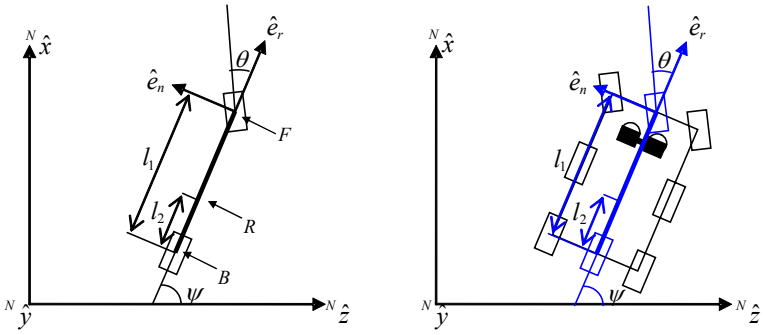
$$\begin{aligned} \dot{z}_k &= \left( \frac{-l_2}{l_1} \sin\theta_k \sin\psi_k + \cos\theta_k \cos\psi_k \right) \cdot u_k \\ \dot{x}_k &= \left( \frac{l_2}{l_1} \sin\theta_k \cos\psi_k + \cos\theta_k \sin\psi_k \right) \cdot u_k \\ \dot{\psi}_k &= \left( \frac{1}{l_1} \sin\theta_k \right) \cdot u_k \\ z_{k+1} &= z_k + \dot{z}_k \cdot \Delta t \\ x_{k+1} &= x_k + \dot{x}_k \cdot \Delta t \\ \psi_{k+1} &= \psi_k + \dot{\psi}_k \cdot \Delta t \end{aligned} \tag{2}$$

A second set of equations manages the rover's traversal over an obstacle. The treatment in our implementation presumes small roll angle $\theta_{roll}$ and ignores pitch, since the pan-tilt unit (PTU) on which the rover camera(s) are mounted can compensate for pitch given near-forward-looking cameras. Such simplifications enable minimum-overhead updates for multiple rovers and can be straightforwardly expanded for large obstacles. Figure 9 provides overhead view and rear rover views that illustrate rover component coordinate frames. Component positions are given by:

$$
\begin{aligned}
{}^{N}\vec{p}_{Obs} &= \begin{bmatrix} {}^{N}z_{Obs} \\ {}^{N}y_{Obs} \\ {}^{N}x_{Obs} \end{bmatrix}, {}^{N}\vec{p}_{R} = \begin{bmatrix} {}^{N}z_{R} \\ {}^{N}y_{R} \\ {}^{N}x_{R} \end{bmatrix}, {}^{N}y_{R} = {}^{R}y_{R} \\
{}^{N}\vec{p}_{Obs\leftarrow R} &= {}^{N}\vec{p}_{Obs} - {}^{N}\vec{p}_{R} \\
{}^{R}\vec{p}_{Obs\leftarrow R} &= \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix} {}^{N}\vec{p}_{Obs\leftarrow R} \\
{}^{R}\vec{p}_{Obs\leftarrow W} &= {}^{R}\vec{p}_{Obs\leftarrow R} - {}^{R}\vec{p}_{W\leftarrow R} = \begin{bmatrix} \Delta z \\ \Delta y \\ \Delta x \end{bmatrix}
\end{aligned}
\tag{3}
$$

where ${}^{N}\vec{p}_{Obs}, {}^{N}\vec{p}_{R}, {}^{R}\vec{p}_{Obs\rightarrow R}, {}^{R}\vec{p}_{W\rightarrow R}$ represent the position of the obstacle/traversal feature (*Obs*) with respect to the inertial (*N*) frame, the rover center-of-geometry (*R*) frame with respect to *N*, the obstacle location with respect to *R*, and the traversing wheel location (*W*) with respect to *R*. Once all positions have been transformed to common frame *R*, relative positions are evaluated to determine whether the rover wheel intersects the obstacle (i.e., $H_{Obs}$ from Eqn. (4) is real). In this case, roll (tilt) angle $\theta_{roll}$ is evaluated as shown in Eqn. (4).

$$
\begin{aligned}
H_{Obs} &= \sqrt{{R_{Obs}}^{2} - (\Delta z)^{2} - (\Delta x)^{2}} \\
\theta_{roll} &= \tan^{-1}\left(\frac{H_{Obs}}{d}\right)
\end{aligned}
\tag{4}
$$




a) Bicycle model representation. b) Rover geometry relative to bicycle model.

**Figure 8. Approximate Rover Two-Dimensional Kinematics Model**

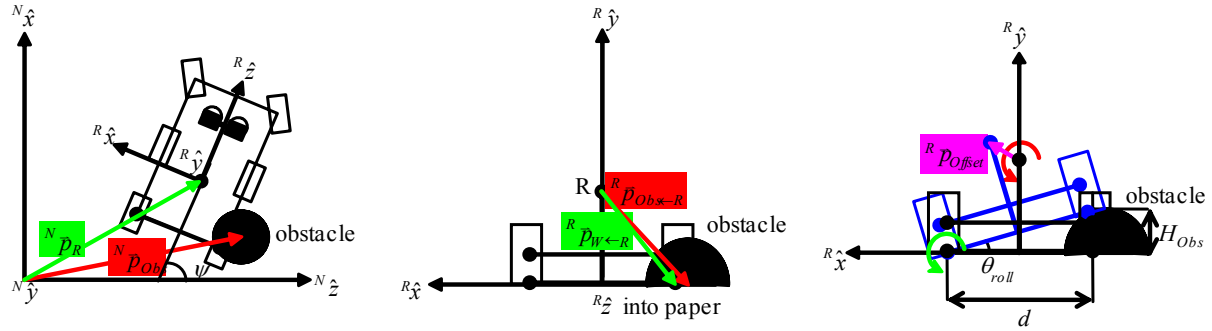



**Figure 9. Approximate Roll Kinematics Model**

Roll/tilt occurs in opposite directions for left versus right wheel obstacle traversal. Both are handled straightforwardly through characterization of rover axis rotation. Per the VRML standard, rotations of the full rover model are carried out at the center-of-geometry frame (*R* in Figures 3 and 4). As shown in Eqn. (5), roll about *R* is represented as a rotation angle and offset adequate for single-wheel traversal of any small obstacle encountered. This offset is added to nominal rover frame position ${}^{N}\vec{p}_{R}$, then the heading $\psi$ and roll rotations are performed to appropriately orient the *R* frame.

$$
\begin{aligned}
&\text{For tire on RHS, } \theta_{roll} > 0 \\
&\begin{bmatrix} {}^{R}y_{off} \\ {}^{R}x_{off} \end{bmatrix} = \begin{bmatrix} \cos\theta_{roll} & -\sin\theta_{roll} \\ \sin\theta_{roll} & \cos\theta_{roll} \end{bmatrix} \begin{bmatrix} {}^{R}y_{R} \\ -d/2 \end{bmatrix} + \begin{bmatrix} -{}^{R}y_{R} \\ d/2 \end{bmatrix} \\
&\text{For tire on LHS, } \theta_{roll} < 0 \\
&\begin{bmatrix} {}^{R}y_{off} \\ {}^{R}x_{off} \end{bmatrix} = \begin{bmatrix} \cos\theta_{roll} & -\sin\theta_{roll} \\ \sin\theta_{roll} & \cos\theta_{roll} \end{bmatrix} \begin{bmatrix} {}^{R}y_{R} \\ d/2 \end{bmatrix} + \begin{bmatrix} -{}^{R}y_{R} \\ -d/2 \end{bmatrix} \\
&(note: {}^{R}z_{off} = 0) \\
&{}^{N}\vec{p}_{Offset} = \begin{bmatrix} \cos\psi & 0 & -\sin\psi \\ 0 & 1 & 0 \\ \sin\psi & 0 & \cos\psi \end{bmatrix} {}^{R}\vec{p}_{Offset}, {}^{R}\vec{p}_{Offset} = \begin{bmatrix} {}^{R}z_{off} \\ {}^{R}y_{off} \\ {}^{R}x_{off} \end{bmatrix}
\end{aligned}
\tag{5}
$$

## C. Environment Model

The environment model consists of both static and mobile objects. While some of the object entities were discussed earlier as related to the kinematics models in the Javascript code in some way (static objects at least had their coordinates "known" by the code for computational purposes), there are other environmental objects that are just as important. The environmental objects are classified as terrain, lighting, cameras, and console controls.

Terrain objects include obstacles and waycircles described previously, as well as the ground plane and sky scene that was modified from a pre-existing environment file.[16] These were placed in the environment as static objects.

Lighting in the scene is important, since the user must see objects clearly to determine the state of the environment. Too much light can lead to distracting "glare", while not enough lighting can distort the images so that edges and object details cannot be seen. By default a point light "sun" is specified overhead that illuminates the entire scene, as if the example rover mission occurs midday.

Camera views are extremely important for situational awareness. Although the operator can use the controls in a general VRML/X3D player interface to change the view manually, it is often more efficient and realistic to provide present viewpoints. Some of these views are static, but some are mobile objects that move in conjunction with the selected virtual rover agent, thereby representing onboard views. Figure 10 shows the dropdown menu list of selectable views, and Figure 11 shows examples of several primary views.

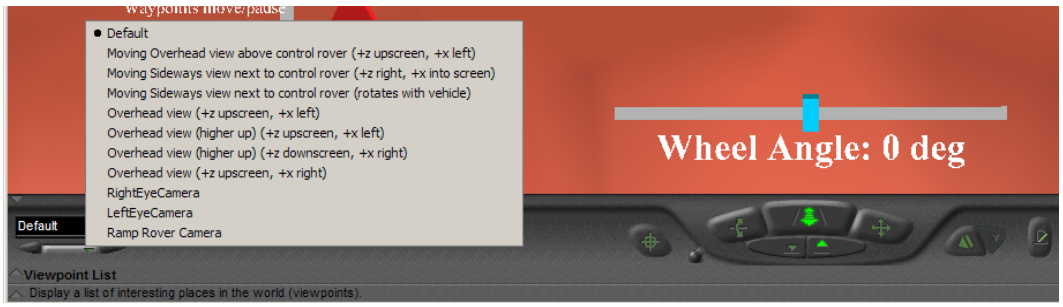



**Figure 10. Menu of Preset Viewpoints**

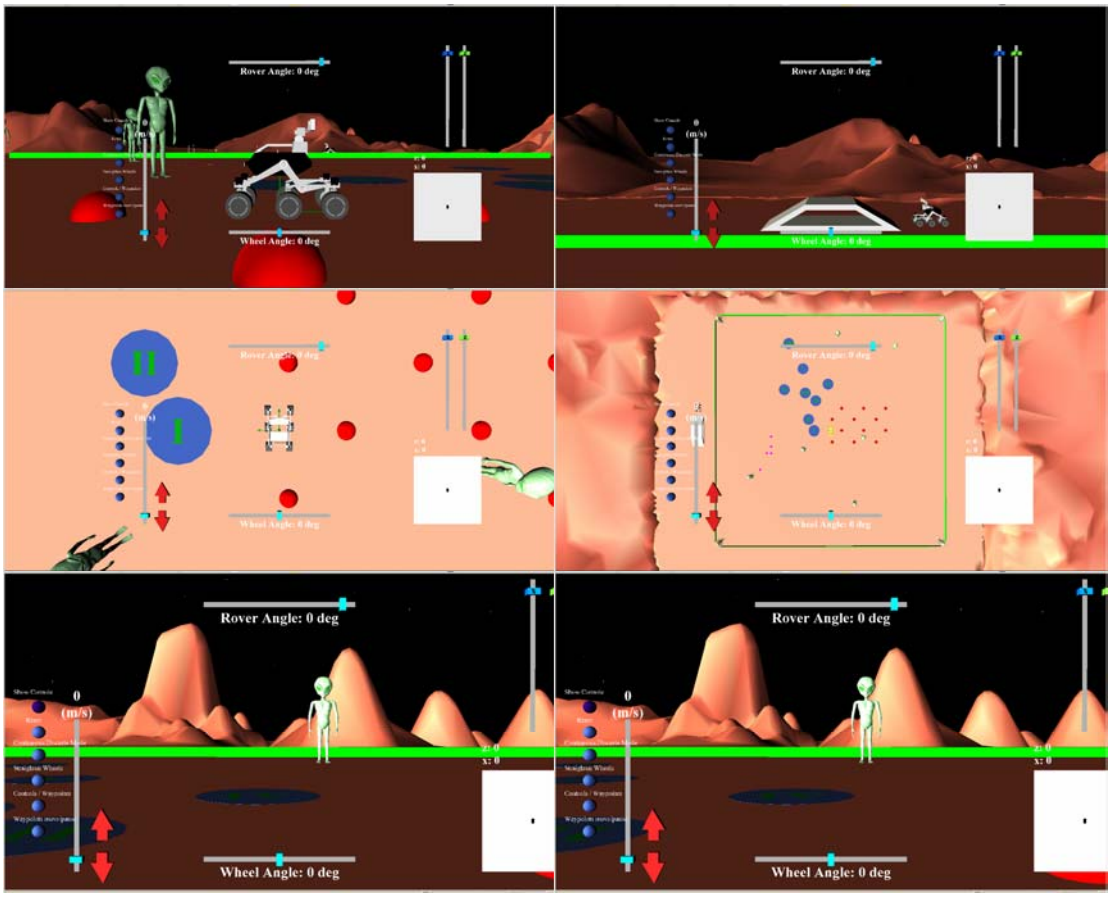



**Figure 11. Example Views, Selections 3 (Side View), 11 (Ramp Rover View), 5 (Low Overhead View), 6 (High Overhead View), 10 (Rover Left Eye), 9 (Rover Right Eye)**

The console buttons and sliders along the edges of the screen currently allow an astronaut to teleoperate a rover agent using the GAI interface (see Figure 3 and/or Figure 11 view 6). There are three modes of teleoperation available to the operator. The first mode uses the waypoints (numbered pink dots) shown in the middle left of the terrain plane in Figure 11, view 6. When the "Controls/Waypoints" button is selected, the rover system will automatically attempt to navigate between the waypoints. The user may move the waypoints to different locations by clicking and dragging them around the screen along the 2D ground plane. This movement is handled by the rover kinematics model. When this button is not depressed, the rover can be teleoperated by the other controls. One mode – "realistic" teleoperation – uses the slider on the left (up-down), which controls the travel speed of the rover, and the slider on the bottom (left-right), which controls the steering of the rover. Clicking either of the red arrows on the left (pointing up and down) commands the rover to move forward or in reverse. Clicking the "Continuous/Discrete Mode" button sets the rover to move constantly, or only a small distance, every time a red arrow is clicked. This "driving" movement is handled by the rover kinematics model. The other teleoperation mode can only occur when the "Continuous/Discrete Mode" button is not depressed (discrete mode). This mode uses the "Rover Angle" and "X" "Z" sliders (top and right of the interface, respectively) as "hand-of-god" controls, which can be used to directly change rover position and orientation. This movement occurs without any restriction by the rover kinematics model. It is useful when initializing rover state, but is unrealistic as a mode of teleoperation. Note that, as apparent in Figure 11, the console tools are configured through the use of proximity sensors and simple routing to follow the user's point-of-view, regardless of where the current camera viewpoint is located.

## IV. Benchmark Testing – Impact of Model Complexity on VRI Frame rate

Performance of our VRI implementation has been evaluated in terms of computational overhead for a variety of modes and scenarios. The evaluated VRI includes the rover kinematics model described above, a reactive virtual environment, and a GAI with a basic interface for direct teleoperation. This version was evaluated without the planner/coordination executive to isolate performance results to the VRI entities. Note that, although an upgrade is

underway, the current benchmarks were computed with uncompiled vrmlscript run within the VRML/X3D parser rather than compiled Java code.

Benchmarks were recorded with this "VRI version 0.9" running under Firefox 2.0.0.3 with the Cosmo Player 2.1.1 (VRML) plugin on a 2GHz laptop with 1GB RAM and a NVIDIA GeForce Go 6200 TE video card with 64 MB RAM at a screen resolution of 1400x1050 in 32 bit color. Table 1 lists execution speeds (complexity) represented by VRI frame rate in frames-per-second (fps).

**Table 1. Benchmark Tests of Version 0.9 of the VRI**

| Test Case | Environment fidelity | | Martian Fidelity | | # of Martians | # of Obstacles | Collision Checks (obstacles) | | | Frame rate (in fps) | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | high | low | high | low | | | No check | Full check (all obstacles every time) | Flowdown (region-based) | Range | Average |
| 1 | X | | X | | 10 | 12 | | | X | 11-30 | ~13 |
| 2 | | X | X | | 10 | 12 | | | X | 11-30 | ~20 |
| 3 | X | | | X | 10 | 12 | | | X | 29-30 | ~30 |
| 4 | | X | | X | 10 | 12 | | | X | 30-50 | ~30 |
| 5 | X | | X | | 10 | 12 | | X | | 9-20 | ~13 |
| 6 | | X | X | | 10 | 12 | | X | | 10-23 | ~17 |
| 7 | X | | | X | 10 | 12 | | X | | 19-30 | ~24 |
| 8 | | X | | X | 10 | 12 | | X | | 17-25 | ~20 |
| 9 | X | | X | | 10 | 12 | X | | | 10-25 | ~15 |
| 10 | | X | X | | 10 | 12 | X | | | 11-30 | ~20 |
| 11 | X | | | X | 10 | 12 | X | | | 29-30 | ~30 |
| 12 | | X | | X | 10 | 12 | X | | | 30-60 | ~30 |
| 13 | X | | X | | *30* | 12 | | | X | 4-20 | ~10 |
| 14 | | X | X | | *30* | 12 | | | X | 4-35 | ~14 |
| 15 | X | | | X | *30* | 12 | | | X | 25-30 | ~25 |
| 16 | | X | | X | *30* | 12 | | | X | 30-45 | ~30 |
| 17 | X | | X | | *4* | 12 | | | X | 16-30 | ~20 |
| 18 | | X | X | | *4* | 12 | | | X | 25-58 | ~30 |
| 19 | X | | | X | *4* | 12 | | | X | 29-30 | ~30 |
| 20 | | X | | X | *4* | 12 | | | X | 30-58 | ~30 |
| 21 | X | | X | | 10 | *30* | | | X | 11-30 | ~20 |
| 22 | | X | X | | 10 | *30* | | | X | 11-30 | ~20 |
| 23 | X | | | X | 10 | *30* | | | X | 27-30 | ~29 |
| 24 | | X | | X | 10 | *30* | | | X | 30-58 | ~30 |
| 25 | X | | X | | 10 | *30* | | X | | 7-16 | ~10 |
| 26 | | X | X | | 10 | *30* | | X | | 9-17 | ~13 |
| 27 | X | | | X | 10 | *30* | | X | | 12-19 | ~14 |
| 28 | | X | | X | 10 | *30* | | X | | 11-20 | ~12 |
| 29 | X | | X | | 10 | *30* | X | | | 11-30 | ~20 |
| 30 | | X | X | | 10 | *30* | X | | | 10-30 | ~20 |
| 31 | X | | | X | 10 | *30* | X | | | 29-30 | ~30 |
| 32 | | X | | X | 10 | *30* | X | | | 30-58 | ~30 |
| 33 | X | | X | | 10 | *4* | | | X | 10-30 | ~20 |
| 34 | | X | X | | 10 | *4* | | | X | 10-30 | ~20 |
| 35 | X | | | X | 10 | *4* | | | X | 29-30 | ~29 |
| 36 | | X | | X | 10 | *4* | | | X | 30-58 | ~30 |
| 37 | X | | X | | 10 | *4* | | X | | 10-30 | ~20 |
| 38 | | X | X | | 10 | *4* | | X | | 10-35 | ~20 |
| 39 | X | | | X | 10 | *4* | | X | | 26-30 | ~27 |
| 40 | | X | | X | 10 | *4* | | X | | 27-42 | ~30 |
| 41 | X | | X | | 10 | *4* | X | | | 10-30 | ~20 |
| 42 | | X | X | | 10 | *4* | X | | | 10-30 | ~20 |
| 43 | X | | | X | 10 | *4* | X | | | 29-30 | ~30 |
| 44 | | X | | X | 10 | *4* | X | | | 30-58 | ~30 |

The impact of environmental fidelity was evaluated through use of the most complex background model (high) or flat-plane terrain with a simple sky (low). "Martian" fidelity was tested with either the "realistic"-looking Martian (high) (Figure 12) or a simple red cylinder of equivalent radius (low). On average, with simplifications of the geometric model, the frame rate jumped to three times the original speed. The simplification of the environment had less impact than simplification of the Martian model due to model construction techniques. Both the environment model and Martian model were modified from pre-existing 3DStudioMax files[16] then exported as VRML models. Originally, these files were large and slowed the VRI substantially. To alleviate this, model

optimization was performed inside 3DStudioMax to reduce complexity of the exported scene. More complex reduction was possible with the environment model without losing the general impact of the scene than the Martian, thus replacing Martians with cylinders had a greater impact on the frame rate. The reason why both these models are more complicated than any of the others is because their primitive shapes are complex, built of 3D splines, surfaces created from the intersection of multiple splines, or detailed lists of grid points. Note that when there are fewer Martians in the field of view displayed by the GAI, the frame rate does increase dramatically, with rate increases of 10 fps or more. This is also tied to the fact that the average rates for a particular viewpoint can vary 10-20 fps between differing viewpoints in the same world model file.

**Figure 12. Close-up of Martian Geometric Model**

Alteration to the collision-checking procedure did not have nearly as drastic an impact on performance as the geometric model changes, which is expected since the simple algebraic computations in the vrmlscript are orders of magnitude less computationally intense than displaying even a simple 3D model. The difference in frame rate when the rover agents were moving versus stationary (not shown) was negligible. This suggests that certain sets of world model files as a whole will have similar performance when initialized with certain versions (high-fidelity vs. low-fidelity) of components (geometric models, kinematics models) at startup, regardless of transient actions that occur during runtime.

Additionally, since nominally one wants the update rate to be at least 10fps at all times, and nominally at 30fps or higher if possible, the "realistic" environments (of complexity test case 1 or less) are all within acceptable operating parameters. It is notable that in all of these tested world models, only the environment models and collision checks on obstacles were changed. All other operations and agents were of the same complexity. This implies that even if the vrmlscript computational overhead is increased, a high frame rate can be recovered by simplifying model complexity.

## V. Conclusions and Future Work

This paper has described a Virtual Reality Interface (VRI) that supports mixed physical and virtual agent teams in a common graphical interface environment. Straightforward kinematics models minimize computational overhead but provide adequate resolution for simple traversals. The use of VR software promotes portable agent entity definition and management and facilitates specification of the environment with multiple levels of graphics fidelity.

Continued development of the VRI will enable full test scenarios with many rovers and astronauts, all managed by a combination of autonomous planner/scheduler and human-initiated directives. Quantitative and qualitative evaluation of the interface through human operator testing and performance evaluation with quantitative metrics (e.g., for planning/scheduling and resource consumption) is the next step to demonstrating the utility of this somewhat alternative design of collaborative robot-astronaut mission simulations. With respect to software development, models will be developed on both "micro" and "macro" level scales. Macro level enhancement will focus more on the energy/time cost computed by the VR simulation, which is important when characterizing high-level situational awareness and the ability of the planner/scheduler to direct activities. Uncertainty must be injected into the test scenarios and managed appropriately at all levels, resulting in anomalies and opportunities to which the system must appropriately react. Micro level development will focus on the details of task completion in VR graphics (e.g. high-fidelity rover and terrain kinematics and dynamics to accurately represent the motion) and expected resource utilization for complex traversals (e.g., traversing a bed of rocks or a steep incline). In this respect, characterization of environmental and rover state changes is critical to maximize efficiency. From the graphics perspective, multiple levels of model fidelity should be available to enable the human operator to accurately perceive the agents and environmental states. Without a specific and well-mapped target lunar or Martian environment, virtual terrain for the simulations must be generated through at least partially randomized functions to provide variety for missions and traversals. These extensions, along with a significant suite of tests, will provide additional data to evaluate the performance of our VRI over a variety of conditions, models, and with a variety of users and interfaces.

## Acknowledgments

The authors would like to thank Klaus-Peter Beier and the other project collaborators from the ENG477 rover project group – Carl von Buelow and Joshua DelCastillo – for their support throughout this project. This research was supported in part by an NSF Graduate Research Fellowship (Catharine McGhan).

## References

[1]Ransan, M., and Atkins, E., "A Collaborative Model for Astronaut-Rover Exploration Teams," *AAAI-2006 Spring Symposium on Human-Robot Teams*, Palo Alto, CA, March 2006.

[2]Ransan, M., and Atkins, E., "Human-Robot Team Task Scheduling for Planetary Surface Missions", submitted to *AIAA Infotech@Aerospace*, May 2007.

[3]"VRML97 Functional specification," Web3D Consortium, URL: http://www.web3d.org/x3d/specifications/vrml/ [cited 27 October 2006].

[4]Hine, B., Hontalas, P., Fong, T.W., Piguet, L., Nygren, E., and Kline, A., "VEVI: A Virtual Environment Teleoperations Interface for Planetary Exploration," *SAE 25th International Conference on Environmental Systems*, July, 1995.

[5]Nguyen, L. A., Bualat, M., Edwards, L. J., Flueckiger, L., Neveu, C., Schwehr, K., Wagner, M., and Zbinden, E., "Virtual Reality Interfaces for Visualization and Control of Remote Vehicles," Autonomous Robots, Vol. 11, No. 1, July 2001, pp. 59-68.

[6]Fong, T.W., Bualat, M., Edwards, L., Flueckiger, L., Kunz, C. , Lee, S.Y., Park, E., To, V., Utz, H., Ackner, N., Armstrong-Crews, N., and Gannon, J., "Human-Robot Site Survey and Sampling for Space Exploration," *AIAA Space 2006 Conference*, AIAA, September, 2006. AIAA-2006-7425

[7]Ferketic, J., Goldblatt, L., Hodgson, E., Murray, S., Wichowski, R., Bradley, A., Fong, T.W., Evans, J., Chun, W., Stiles, R., Goodrich, M.A., Steinfeld, A., King, D., and Erkorkmaz, C., "Toward Human-Robot Interface Standards II: An Examination of Common Elements in Human-Robot Interaction Across the Space Enterprise," *AIAA Space 2006*, AIAA, September, 2006. AIAA-2006-7388

[8]Vince, J., *Virtual Reality Systems*, Addison-Wesley, New York, 1995.

[9]"X3D International Specification Standards," Web3D Consortium, http://www.web3d.org/x3d/specifications/x3d/ [cited 27 October 2006].

[10]"Xj3D Java-based X3D Toolkit and Browser," Web3D Consortium, http://www.web3d.org/x3d/xj3d/ [cited 30 April 2007].

[11]"NASA – JPL Solar System Simulator," NASA JPL, http://samadhi.jpl.nasa.gov/ [cited 30 April 2007].

[12]Rehnmark, F., Bluethmann, W., Huber, E., Rochlis, J., and Ambrose, R., "Useful Interactions Between Human and Robotic Agents Performing a Cooperative Assembly Task," *AIAA Space 2003 Conference*, AIAA, September, 2003. AIAA 2003-6273

[13]Homan, D. J., "Virtual Reality and the Hubble Space Telescope," *Space Programs and Technologies Conference*, AIAA, Sept., 1994. AIAA-1994-4558

[14]Pelz, C., Brickman, N., Liang, C., Hogue, J., and Aponso, B., "Tactical Insertion Mission Planning and Rehearsal Using Virtual Reality Simulation," *AIAA Atmospheric Flight Mechanics Conference*, AIAA, August 2003. AIAA 2003-5610

[15]Carey, R., and Bell, G., *The Annotated VRML 2.0 Reference Manual*, Addison-Wesley, New York, 1997.

[16]"Group Project Report for Eng 477," Virtual Reality Rover Project Website, http://www-personal.umich.edu/~cmcghan/eng477_f06/download.html [cited 30 April 2007].

[17]"Eng 477: Laboratory Notes – VRML 2.0, VRML Lab 1: Transform," Eng477 Website, http://www.engin.umich.edu/class/eng477/f06/docs/Laboratory/VRML/intro/transform.html [cited 30 April 2007].

[18]"Eng 477: Laboratory Notes – VRML 2.0, VRML Lab 1: Grouping and Children," Eng477 Website, http://www.engin.umich.edu/class/eng477/f06/docs/Laboratory/VRML/nodes/children.html [cited 30 April 2007].