

ROAMS: Planetary Surface Rover Simulation Environment

A. Jain, J. Guineau, C. Lim, W. Lincoln, M. Pomerantz, G. Sohl, R. Steele
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109

Abstract

This paper describes the ongoing development of the ROAMS physics-based simulator for planetary surface exploration rover vehicles. ROAMS includes models for various subsystems and components of the robotic vehicle including its mechanical subsystem, an electrical subsystem, internal and external sensors, on-board resources, on-board control software, the terrain environment and the terrain/vehicle interactions. The ROAMS simulator can be used for stand-alone simulation, closed-loop simulations with on-board software or for operator-in-the-loop simulations.

1 Introduction

There has been significant growth in the number space exploration missions devoted to planetary surface exploration using mobile rover vehicles. The Mars Exploration Rover (MER) project due to launch in 2003 is a prime example of a current mission under development, with the Mars Science Laboratory (MSL) representing the next generation of such surface exploration missions. Highlights of the MSL mission include the extended mission life (over 18 months) and the desire to increase the rover's onboard capabilities in order to reduce the amount of ground intervention needed for the exploration activity. There is a strong need to develop validated modeling and simulation capability for the surface system to allow missions to carry out detailed surface system trade studies, develop and test new rover technologies, support the development of onboard flight software architectures, develop mission operations concepts etc.

In this paper, we describe the further development of the ROAMS physics-based simulator for planetary surface rovers beyond what was previously reported in [1]. One of the goals of ROAMS is to support the early development, testing and maturation of new rover technologies for eventual transfer for mission use.

ROAMS includes models for various subsystems and components of the robotic vehicle including its mechanical subsystem, an electrical subsystem, internal and external sensors, onboard resources, on-board control software, the terrain environment and the terrain/vehicle interactions. The ROAMS simulator can be used for stand-alone simulation, closed-loop simulations with onboard software or for operator-in-the-loop simulations.

In this paper we will describe the current status of ROAMS as well as the future development and validation plans.

2 ROAMS Models

A key requirement on ROAMS is that it provide a high-fidelity virtual rover that can be used to test and validate the onboard rover control software. This implies that ROAMS implement all the interfaces to the rover hardware and the models for the hardware. However, to support needs other than onboard software validation, ROAMS also includes representative models for onboard software components such as navigation, locomotion and motor control algorithms. These onboard software models allow users to use ROAMS with component algorithms, eg. external planning engines can give "go to" commands to the ROAMS rover without having to integrate with low level rover navigation software.

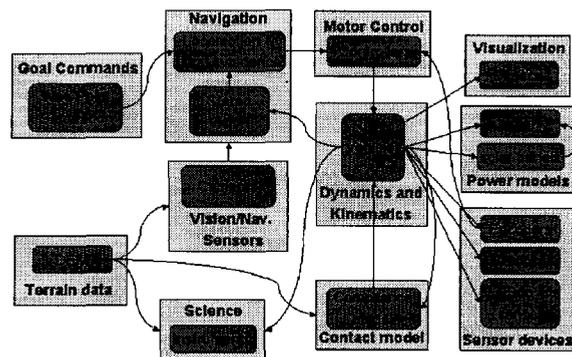


Figure 1:

Figure 2 describes the key sub-systems modeled within Roams - including both the physics based models as well as models for the onboard software. The *Dynamics and Kinematics* module includes a multibody kinematics and dynamics model of the rover. This module interacts with the

wheel/soil interaction *Contact model* to compute the multi-body state of the rover. This interaction includes the use of the *Terrain data* module which contains the terrain model, as well as any steering/wheel motor torques being commanded by the motor controllers. The rover multibody state is used by the the *Sensor devices* as well as the *Power model* to simulate the sensor and power state of the rover. The information is also used by the *Navigation* and *Vision/Nav. sensors* modules to close the loop and provide new set points to the motor control software. The following sections describe each of these modules in additional detail. The rover state is also used to send messages to the *Visualization* module to update the 3D graphics visualization.

2.1 The Rocker/Bogey Vehicle Models

The rover vehicle modeled in ROAMS is the *rocker/bogey* class of 6-wheeled rovers used for planetary surface exploration. There are several variations on the basic design in terms of the location of the differential, the number of steerable wheels and the various mechanical dimensions of the rover. While the underlying DARTS multibody engine supports very general multibody topologies, ROAMS specializes this to rocker/bogey vehicles by using parameterized templates. The templates provide entries for the basic kinematics and inertia properties of the rover, its arm (if any), inertial sensors, number of steerable wheels etc. The templates simplify the definition and addition of new rover vehicle models. For existing rocker/bogey platforms, the templates are filled in with numeric values specific to the rover. However, as described later, the templates make it easy to explore rover design space by using ROAMS within Monte Carlo simulations.

2.2 Mobility Configuration Kinematics

Applications such as the development of navigation and obstacle avoidance algorithms do not require full dynamic simulation of the rover/terrain interaction. For these applications, ROAMS includes a *configuration kinematics* capability to provide high speed kinematic state propagation. When operating in kinematic mode, ROAMS computes the height above ground and roll/pitch angles of the vehicle's chassis as well as the rocker and bogey angles given the rover's 2-D location (x, y) and heading h .

The configuration kinematics problem requires the solution of a set of constraint equations for seven unknowns: 2 rocker angles, 2 bogey angles, chassis height, roll angle and pitch angle. The differential rocker constraint provides a direct relationship between the left and right side rocker angles:

$$\theta_L = -\theta_R$$

Six more constraint equations are provided by forcing the surface of the six rover wheels to remain in contact with the terrain. The user can chose between two different forms of these contact constraints in ROAMS.

The first form of the wheel contact constraint assumes the point of contact is located directly beneath the wheel hub location:

$$w_z - r_w = z_T(w_x, w_y)$$

where w_z is the z coordinate of the wheel center, r_w is the wheel radius and z_T is the height of the terrain at the (x, y) location under the wheel. The position of the wheel center (w_x, w_y, w_z) is related to the unknowns through the rover Jacobian. This form provides good results for rover motion on relatively flat terrain. A second form of the contact constraint allows the contact point to move on the surface of a parameterized wheel shape.

$$S = g(u, v) \quad S_z = z_T(S_x, S_y) \quad \text{cross}(\text{norm}_S, \text{norm}_T) = 0$$

where S is wheel surfaced parameterized by new unknowns u and v . The wheel surface z coordinate matches the terrain height z_T . An added constraint forces the normal of the wheel surface to match the terrain normal.

These constraints are used with the Newton-Raphson solver described in Section 3.5 to solve for the rover coordinates at the new location.

2.3 Vehicle Dynamics

While kinematic state propagation is sufficient for some tasks, others require a dynamics based simulation to accurately model behavior such as sliding, slipping and sinking of the rover on the terrain. The critical element of dynamic simulation is the interaction between the wheel and the soil. Once this contact force has been determined, DSHELL/DARTS provides the dynamics engine required to propagate the rover state.

Writing the static force balance for a six wheeled rover ([2]) results in a under-determined set of equations for the contact forces. A variety of means have been employed to solve this under-determined system. It is possible to decompose the lateral and transverse components ([2]) of the contact force. While this provides good solutions for small roll angles, it is not suitable for a general 6 DOF rover simulation. Another technique is to find a set of contact forces which minimize some external criteria such as energy. This can be very time consuming and does not provide any continuity between solutions. A third method, which is used by ROAMS, is to insert a compliant system between the wheel and soil.

The compliant contact model used by ROAMS is based on a rigid body contact model described in ([3]). This model handles both rolling and sliding contacts by inserting compliant spring dampers in both the normal and tangential directions. The transition between rolling and sliding regions is governed by the Coulomb friction law. While the Coulomb law has limited applicability to wheel-soil interaction, the model provides a well defined and continuous estimate of normal and tangent force at the contact point. We are working to extend this model so that more accurate terra-mechanics based models can replace the Coulomb law.

2.4 Arm Configuration Kinematics

A rover arm needs to position scientific instruments near objects of interest. Field test and mission rover arms for recent and planned missions have four or five degrees of freedom (DOF). MER has a 5 DOF and FIDO has 4 DOF. The arm is defined via the ROAMS template model file.

ROAMS includes an inverse kinematics engine for use with arms with different degrees of freedom. The Newton-Raphson solver described in Section 3.5 is again used to solve for the inverse kinematics. The constraints in this case are on the arm end-effector. The x,y,z end position of the arm and an orientation constraint are inputs to the DSHELL model. The arm model is defined in a Darts model file. A four-degree of freedom arm was created.

2.5 Terrain Models

The terrain is represented by a digital elevation map (DEM). This terrain model is used to drive the wheel/soil interaction models, the navigation and hazard avoidance models as well as the Dspace visualization engine. We have developed interfaces to third party terrain synthesis software for the generation and use of these terrain models. The terrain synthesis software [4] supports the generation of terrain models with specific statistical properties (eg. rocks/crater distributions) as well as the fine-grain enhancement of existing terrain maps.

We are currently incorporating terrains reconstructed from empirical terrain measurements into ROAMS to support the validation activities.

2.6 Inertial Sensors

ROAMS includes models for inertial sensors such as inertial measurement units (IMUs) and sun sensors. The IMUs models include models for gyroscopes and accelerometers. Also included are models for sun sensors, inclinometers, and wheel and steering encoders.

2.7 Motors

The wheel and steering motor models include gears at the wheels including profile generators for the motors which take maximum velocity and acceleration constraints. Motor controller models are provided which are driven by the set points generated by the profile generators.

2.8 Power sub-system

The power sub-system includes solar panel, sun sensor and battery models. These systems track power available to the rover. Parameters define the power draw of devices attached to the rover such as wheel motors and onboard electronics. The sun sensor provides current sun angle information to the solar panel model which outputs power to charge the battery.

2.9 Navigation sub-system

The navigation sub-system's is responsible for accepting goal commands, and generating a sequence of locomotion commands which guide the rover to the goal while avoiding hazardous areas. In lieu of camera image simulation, Roams implements models that process the terrain model information based on the rover location and orientation to generate the vision sensor data products needed by the navigation algorithms. The rover is commanded to follow an arc for a set distance. At the completion of this short traverse more terrain information is gathered and the process repeats until the rover reaches it's desired goal or is blocked by obstacles from achieving it's goal. As a practical matter the goal is not a single point, but some bounded circle around the actual goal.

One of the navigation algorithms implemented within Roams is the Sojourner navigation algorithm which selects a traverse arc based on detected hazards to the left, right and center of the rover (Figure 2).

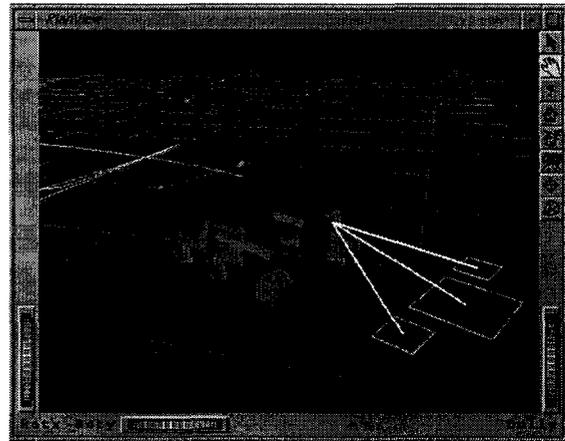


Figure 2: Obstacle detection for Sojourner navigation algorithm

The Gestalt navigation algorithm [5] developed for the Mars Exploration Rover (MER) mission has also been integrated into ROAMS. The Gestalt algorithm generates a set of candidate motion arcs for the rover. The set of arcs are then ranked based upon a goodness map generated based on the terrain properties such as roughness, slope, the size of the rover, and any obstacles. A single arc is selected as the 'best' one to traverse. Figure 3 shows a graphical representation of the goodness map overlaid on the terrain surrounding the rover.

2.10 Locomotion sub-system

The locomotion sub-system takes traverse arc commands generated by the navigation algorithm and in turn generates steering and wheel motor commands needed to execute those

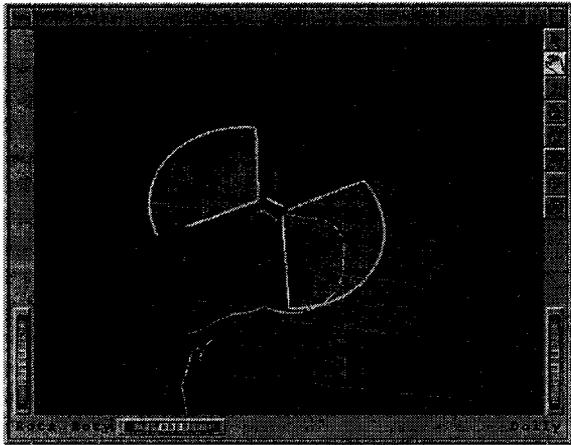


Figure 3: Graphical depiction of the goodness map for the Gestalt navigation algorithm

motions. In kinematics simulation mode, the rover is simply placed at various points along the desired motion path until the desired distance is reached. In dynamics simulation mode, the locomotion module generates set point commands for the motor controller to steer the rover along the desired path. Closed loop controller models are used to drive the steering and wheel motors along the desired trajectory. An odometry model is used to determine when the rover has completed the motion by looking at the amount each wheel has rotated since the start of the command.

3 ROAMS Design Elements

3.1 Dshell/Darts Simulation Framework

ROAMS is an adaptation of the DSHELL/DARTS [6, 7] multi-mission spacecraft simulation framework and tools for the surface rover domain. DSHELL provides a framework for development of sensor, actuator, electrical and mechanical subsystems. Expanding these capabilities, ROAMS includes models for the terrain environment, contact between the vehicle's wheels and the terrain, models for onboard software functions including hazard avoidance and navigation.

DSHELL provides a rich variety of simulation features which are available within ROAMS. Example features include plain ASCII file model files for defining simulation configuration; usability for closed-loop simulation use; a scripting interface; extensive peek/poke capability into simulation data; data-logging and monitoring facility; checkpoint capability etc.

3.2 Rover Model Definition

The DSHELL tool uses plain ASCII data files to define the simulation configuration at run-time - the various models to

be instantiated and their interconnections. This generic capability is specialized for ROAMS using templates to simplify the specification of rocker/bogey rover models. The templates allow the specification of standard kinematic and dynamics vehicle parameters as well as those for the various device models. The number of steerable wheels, the number of links in the instrument arm, mount locations can all be tailored to specific rovers. The user also has the ability to specify mathematical expressions to specify how dependent parameters get defined. This powerful feature allows users to use ROAMS to build up sophisticated rover models and use them for exploring design space.

3.3 Roams Configuration Modes

The rover onboard software can be regarded as a hierarchy of functional layers. At the very bottom is the motor control layer which controls the various steering and wheel motors. The commands for this layer are generated by the locomotion layer based on arc traverse commands it receives from the navigation layer. The navigation layer itself responds to goal commands from planning engines. Roams is being designed to close the loop with the rover onboard software at each of these levels (see Figure 4. To support these multiple simulation modes, Roams includes representative models for some of the rover onboard software functions.

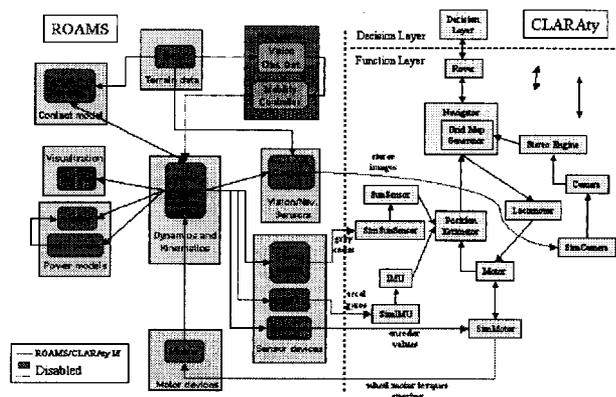


Figure 4: Closing the loop at multiple levels

At the lowest *motor level* simulation mode, ROAMS closes the loop at the motor commands level. In this mode ROAMS runs the full vehicle dynamics to propagate the state of the vehicle. In this mode ROAMS is simulating only the hardware actuators and sensors. At the *navigation level* simulation mode, ROAMS closes the loop at the go-to commands from the rover software. In this mode ROAMS' navigation and locomotion models are used in the simulation. In this

mode the rover has the choice of running the simulation in either dynamics or kinematics modes discussed in Sections 2.1 and 2.2. While the former provides higher fidelity, the latter is significantly faster. The user has a choice of selecting between the Sojourner and Gestalt navigation algorithms described in Section 2.9.

The selection between different modes is done at run-time. This capability uses DSHELL's ability to activate and deactivate individual and collections of models at run-time.

3.4 Roams Run-Time Environments

ROAMS can be run in stand-alone mode for rover simulation. It provides a Tcl command line interface for users to interact with the simulation, or executes command scripts. There is also a Roams IF C++ interface available that allows external applications (such as rover onboard software) to close the loop with ROAMS. In addition to these modes, ROAMS uses DSHELL's DMEX Matlab/Simulink wrapper generator to interface ROAMS with the Simulink environment. The Simulink environment is a popular desktop environment for algorithm development and analysis. The use of the DMEX wrapper allows the use of ROAMS as a Simulink S-function within it (see Figure 5), where other S-function blocks can be developed by the user to develop larger simulations.

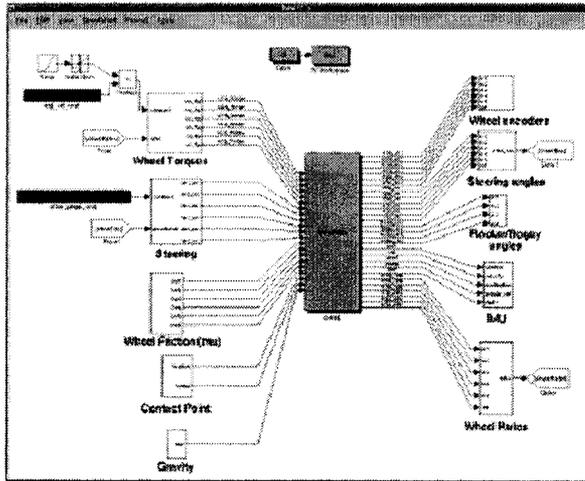


Figure 5: Roams embedded within Simulink via the Dmex interface

3.5 Newton-Raphson Kinematics Solver

ROAMS includes a generic Newton-Raphson kinematics solver library that is used for solving the inverse kinematics problem for different types of constraints problems such as configuration kinematics and arm kinematics. Once we have chosen the form of the constraint equations, we develop the

constraint Jacobian matrix.

$$J_c = \frac{\partial f}{\partial q}$$

where f is the set of constraint equations and q is the rover joint angles. The constraint Jacobian requires terrain derivative (slope and curvature) information as well as the rover kinematic Jacobian. A Newton-Raphson iterative technique is used to find a set of rover roll, pitch and joint angles which satisfy the constraint equations. Generally, a solution is obtained quite rapidly in only few iterations. However, it is possible for highly uneven or rocky terrain to make a solution impossible.

The Jacobian is used as an approximation to f in the neighborhood of the current q . DARTS provides functions to compute the Jacobian for any point on the rover. The Jacobian approximates how much f will change in world space when a q is changed. A solver based on Newton-Raphson is used to compute q^* in the equation $f^* = f(q^*)$ where f^* is the desired end position and orientation.

controller.

3.6 Computational Geometry

The SWIFT++ library [8] is currently used by ROAMS to compute wheel penetration distance into the terrain and the ANN library [9] is used to determine clearances between the rover and terrain.

SWIFT++ and Wheel-Terrain Penetration Distance:

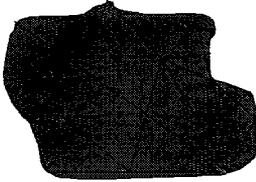
Given a set of two or more convex, polyhedral objects, SWIFT++ ("Speedy Walking via Improved Feature Testing for Non-convex Objects") provides functions to compute the distances and contact points between pairs of objects.

In ROAMS, SWIFT++ objects are created to represent the shapes of the rover wheels and the terrain. To minimize computation time, only small "patches" of the terrain underneath the rover wheels are created and the surface of each patch is decomposed into a set of triangular tiles (see Fig. 1). As the rover moves along the terrain, new patches are created and destroyed. To compute wheel penetration distance, each wheel's surface is represented by a cylinder and a patch of terrain tiles underneath the wheel is used to represent the ground surface. Since SWIFT++ does not provide functions to compute penetration distance, a smaller cylinder within the wheel is created and SWIFT++ is used to find the distance and contact points between the smaller cylinder and the terrain surface from which the wheel penetration distance can easily be computed (see Figure 6).

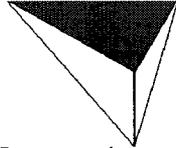
ANN and Clearance Sensors: A drawback to SWIFT++ is that only a small number of 3D objects can be represented in SWIFT++ without incurring a performance penalty. Since the computation of clearances between the rover terrain requires a larger area of terrain, ROAMS uses the ANN ("Approximate Nearest Neighbors", [9]) C++ library to compute the distance between a sensor node on a rover and the nearest point on the terrain surface. The ANN library represents objects as a "cloud" of points (a three dimensional array of

Model Terrain as Solid Triangular Tiles

Terrain patch
composed of flat (2D)
triangles.



Create solid (3D) tile
from each surface triangle.



- From one vertex, drop a normal vector away from the surface.
- Connect the two other vertices to the normal vector.
- Load tiles into SWIFT; each tile is a separate SWIFT object.

Figure 6: Model Terrain as Triangular Tiles

floating point values sorted in a hash table) and provides functions to efficiently compute the distance from a user-specified point (e.g. a rover sensor) to the nearest point in the cloud. Clearance sensors are placed on the four corners of the rover to detect oncoming hazards and an additional sensor is placed at the bottom of the rover's belly to detect ground clearance. The terrain surface (a three dimension array of floating point values) are sorted in an ANN object and the appropriate ANN functions are called to determine the closest point on the terrain surface to each sensor on the rover. We have found that ANN is able to find the closest point in an array of 65,000 terrain points in less than 4000 microseconds on a 400 Mhz Sun ULTRA 10 workstation running Sparc SunOS 5.7.

3.7 Monte Carlo Simulations

Complex simulations with many possible interactions require a Monte Carlo capability to estimate the dependencies between parameters of interest. A Monte Carlo simulation calculates multiple scenarios of a model by repeatedly sampling values from the probability distributions for the input variables and using those values for a separate run.

DSHELL/DARTS contain a Monte Carlo simulation module that is available for running ROAMS' Monte Carlo simulations. This capability is used to evaluate rover designs and performance under different environment conditions and scenarios.

The Monte Carlo capability is centered on a relational database. The relational database provides a convenient way to setup a simulation and store the results for later analysis. There are four main tables that contain information about 1) the experiment, 2) the stage commands, 3) the result specification, and 4) the collected results. The experi-

ment table contains the experiment id, description, and the DSHELL/DARTS script to run. The stage command table has run id, the Tcl command to run at a particular stage, and the simulation stage when the command should run. The result specification table has fields to hold the result name, Tcl command to run when condition is met, and triggering condition when the value should be collected. Finally the results table contains the time and run id stamped results.

The setup of a Monte Carlo experiment requires an existing ROAMS simulation. The top-level script is conceptually split into multiple stages. At each stage input parameters can be set and results collected. The staging is important because it gives sequence control over initialization and data collection.

A process control manager spawns a job on an idle machine corresponding to a single run. Each machine accesses the database for its setup and to store its results.

3.7.1 Dspace 3D Visualization Tool

The 3D graphics visualization component of ROAMS is the Dspace visualization system. Dspace is a C++/OpenInventor/Tcl based visualization system that accepts Digital Elevation Maps (DEM) and OpenInventor scenegraph based CAD files, and uses these, along with DSHELL/DARTS state information, to render high-quality, real-time scenes. Dspace receives state commands from DSHELL/DARTS via the DshellDspace sub-system.

Using C++ and object oriented design principles, Dspace has been designed to be highly flexible and run-time configurable by the user, via a rich API of over 100 C++ or Tcl accessible routines. Routines that directly support rendering, such as viewport management, lighting and shading, camera control, atmospheric conditions such as fog or haze and scenegraph traversal, are combined with ROAMS specific capabilities such as DEM based terrain support, terrain level-of-detail rendering, spacecraft attitude and articulation and obstacle detection footprint display..

An important feature of Dspace, and one in which ROAMS takes full advantage, is the ability to display multiple graphical viewports, each with a unique or with a shared scenegraph. The sharing of scenegraphs is important to ROAMS when varying viewpoints of the same scene are required, such as a "chase" camera fixed to a rover in one viewport, and a "plan" view, that allows the user to view and control the viewpoint of the entire scene via mouse interaction. Scenegraphs unique to a given viewport are useful when a scene must be rendered in some special way, such as the view through an obstacle detection camera, where atmospheric conditions and high fidelity terrain or rover models must be used to achieve higher levels of realism than required by the normal simulation views.

To achieve real-time or near real-time rendering performance, Dspace takes advantage of OpenInventor's level-of-detail support for both DEM based terrain and for CAD based rover models. For DEM based terrain, users specify the amount of level-of-detail subdivision and the number

of levels to be generated, and Dspace constructs a hierarchical scenegraph that contains the various level-of-detail components. For CAD based rover models, users can create multiple levels-of-detail by presenting Dspace with multiple CAD files of the rover, each with a gradual reduction in complexity. OpenInventor, at run-time, decides which level to "swap" in, based upon the distance of the terrain or rover from the viewer and by utilizing predetermined distance factors provided by the user. For example, as the user changes the viewpoint via the mouse or through the API, portions of the terrain that are farther away from the viewer are rendered in lower fidelity (fewer polygons), while the portions of the terrain closest to the viewer are rendered in higher fidelity (more polygons). This mechanism assures that terrain closest (and seemingly of more interest) to the viewer is always rendered at the highest possible fidelity. While somewhat complex to implement, this mechanism produces a substantial increase in rendering performance.

An interesting feature of this level-of-detail design, and one that was driven by the ROAMS requirement to project obstacle detection footprints onto the terrain, is the way in which material and surface normal information is maintained in a data structure that is global across all levels-of-detail. During a typical simulation run, ROAMS can direct Dspace to render an obstacle detection footprint onto the terrain at any given time and location. This change in terrain color for a specific portion of the terrain must be propagated down to all detail levels, because for a given viewpoint, OpenInventor controls which level-of-detail is currently being rendered. Making the material information global ensures that a change in color, or some other material property, is applied uniformly, no matter which level-of-detail is currently being rendered.

3.7.2 DshellDspace Interface Module

DshellDspace is the subsystem by which commands are typically sent from ROAMS to Dspace. DshellDspace provides a layer of abstraction that allows the user to organize and structure a simulation run and provides the "glue" that binds DSHELL/DARTS information to Dspace. DshellDspace organizes simulation data into Assemblies, Graphs and Bodies, each with a unique function and connection to Dspace, and also provides API routines to support general Dspace functionality, such as viewport management. While these general API routines are convenient, the power of DshellDspace lies in its organization of simulation components.

Assemblies can be thought of as the equivalent of an OpenInventor scenegraph and are the high-level construct for managing simulation objects in DshellDspace. Assemblies maintain and manage a list of DshellDspace Graphs that, during initialization time, is transmitted to one or more Dspace viewports. The Assembly supports typical scenegraph operations, such as enabling/disabling scenegraph traversal at run-time. DshellDspace can support multiple Assemblies, but typically the number used during a simulation corresponds in some way to the number of active Dspace

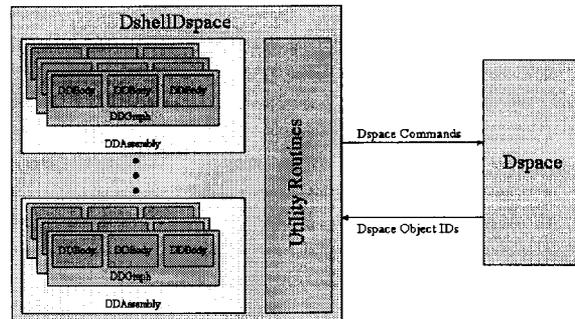


Figure 7: The DshellDspace interface module

viewports.

Graphs are always owned by an Assembly, and contain information related to the loading of OpenInventor CAD files, terrain DEM files, texture files, graphical primitives such as spheres, cones, cubes and lines, material properties and the maintenance of a list of DshellDspace Bodies. Graphs can be thought of as the equivalent of a branch in a scenegraph tree.

If Graphs are the branches, then DshellDspace Bodies are the leaves on those branches. Bodies directly control the connection between the continuously changing DSHELL/DARTS state information and how that information is transmitted to Dspace. For example, a rover may be made of up various parts, all of which might be articulated independently. Chassis, rocker bogey, axles, wheels or camera mast, each have DSHELL/DARTS equivalents that are continuously modified based on state changes. DshellDspace Bodies utilize the DSHELL "watch" variable mechanism to monitor changes in state information, typically attitude or position, and pass the new state to Dspace. By utilizing the watch variable mechanism, DshellDspace allows the user to setup the connection between a DSHELL/DARTS body and a Dspace scenegraph item, at initialization time, and then let DshellDspace automatically update Dspace throughout the course of a simulation run.

Because DshellDspace maintains a hierarchical list and a current set of states for all of the items that Dspace needs during a simulation, Dspace can be started and then restarted, say on a different workstation, in the middle of a simulation run, without the loss of any state information, as long as all DshellDspace Assemblies are transmitted to the newly restarted Dspace.

4 Roams Applications

While ROAMS is being developed currently for eventual use by NASA's 2009 Mars Science Laboratory mission, we describe here some of the other applications where ROAMS is being used.

CLARATy [10] is a reusable rover software architecture being developed in collaboration by several institutions including JPL, NASA Ames, CMU and others. A goal of the CLARATy development is to provide an open architecture for component algorithm developers to develop and integrate their capabilities into. While CLARATy is targeted to run on physical rovers, closed-loop interfaces between CLARATy and ROAMS have been developed to allow users to transparently switch between the physical rover and simulation testbeds. The closed-loop interfaces with CLARATy have been developed at the navigation and the motor levels (see Figure 4). At the highest "rover" level, ROAMS includes models for the key functional elements of onboard software so that the commanding can be at the high-level (eg. go-to commands). At the lowest "motor" level, the interface supports commanding of individual motors in ROAMS with sensor/encoder feedback being provided back to CLARATy. To support the "rover" level closed-loop interface, ROAMS includes a variety of models for obstacle detection algorithms and onboard navigation software. The user can select between these different algorithms at run-time to evaluate their performance.

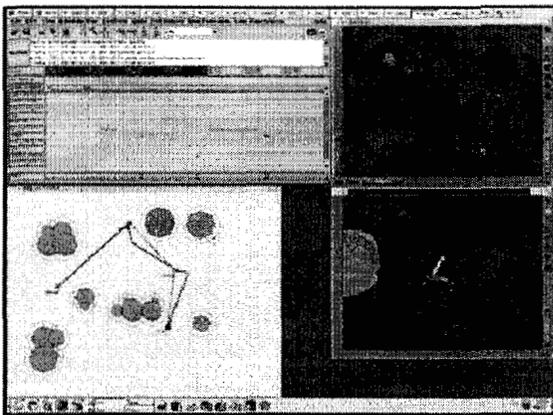


Figure 8: CLEaR, CLARATy and Roams closed loop simulation for rover autonomy development

CLEaR [11] is a high level task planner currently under development. It is being used to provide the *decision layer* within the CLARATy framework. Figure 8 shows a screenshot of the closed-loop integration between CLEaR, CLARATy and ROAMS for the development and test of autonomy capabilities.

The ROAMS team is also collaborating with NASA Ames' Mission Simulation Facility (MSF) [12] to develop a

reusable simulation capability targeted to autonomy technology development. The MSF architecture includes standardized interfaces to allow local and remote model providers and users to develop integrated simulations tailored to their needs. ROAMS is providing a detailed rover engineering simulation capability that can be used by autonomy technology providers to develop and mature their capabilities using realistic simulations for eventual infusion into missions.

The Mission Data System (MDS) project at JPL is developing the next generation flight, ground and test framework for future space missions. The ROAMS simulator is also being used by MDS in closed-loop simulations to support the development and test of the MDS capabilities.

5 Conclusions

This paper provides a brief summary of the current ROAMS capabilities. Even as ROAMS continues to be developed further, it is in use by a number of rover technology developers to support their development. A significant element of the current ROAMS development is on validating the simulation capabilities against experimental data. This is an important step to maturing and benchmarking ROAMS' performance for eventual use by missions such as the Mars Science Laboratory.

Acknowledgments

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract with the National Aeronautics and Space Administration. We would also like to acknowledge the support of NASA's Mars Technology Program which has supported the development of ROAMS.

References

- [1] J. Yen, A. Jain, and B. Balaram, "ROAMS: Rover Analysis Modeling and Simulation Software," in *i-SAIRAS'99*, (Noordwijk, The Netherlands), June 1999.
- [2] K. Iagnemma, *Rough-Terrain Mobile Robot Planning and Control with Application to Planetary Exploration*. PhD thesis, Massachusetts Institute of Technology, 2001.
- [3] P. Kraus, A. Fredricsson, , and V. Kumar, "Modeling of Frictional Contacts for Dynamic Simulation," in *International Conference on Intelligent Robot Systems (IROS'97)*, (Grenoble, France), Sept. 1997.
- [4] R. Gaskell, J. Collier, L. Husman, and R. Chen, "Synthetic Environments for Simulated Missions," in *Proceedings IEEE Aerospace Conference*, (Big Sky, Montana), Mar. 2001.

- [5] S. B. Goldberg, M. W. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *IEEE Aerospace Conference*, vol. 5, (Big Sky, Montana, USA), pp. 2025–2036, Mar. 2002.
- [6] A. Jain and G. Man, "Real-Time Simulation of the Cassini Spacecraft Using DARTS: Functional Capabilities and the Spatial Algebra Algorithm," in *5th Annual Conference on Aerospace Computational Control*, Aug. 1992.
- [7] J. Biesiadecki, D. Henriquez, and A. Jain, "A Reusable, Real-Time Spacecraft Dynamics Simulator," in *16th Digital Avionics Systems Conference*, (Irvine, CA), Oct. 1997.
- [8] S. A. Ehmann, "Swift++: Speedy walking via improved feature testing for non-convex objects," 1997. URL: <http://www.cs.unc.edu/geom/SWIFT++>.
- [9] D. M. Mount and S. Arya, "Ann: Library for approximate nearest neighbor searching." URL: <http://www.cs.umd.edu/mount/ANN>.
- [10] I. Nenas, R. Volpe, T. Estlin, H. Das, R. Petras, and D. Mutz, "Toward Developing Reusable Software Components for Robotic Applications," in *International Conference on Intelligent Robot Systems (IROS)*, (Maui Hawaii), Oct. 2001.
- [11] T. Estlin, R. Volpe, I. Nenas, D. Mutz, F. Fisher, B. Engelhardt, and S. Chien, "Decision making in a robotic architecture for autonomy," in *Sixth International Symposium on Artificial Intelligence, Robotics and Automation for Space (i-SAIRAS 2001)*, (Montreal, CA), June 2001.
- [12] L. Fluckiger and N. C., "A new simulation framework for autonomy in robotic missions," in *International Conference on Intelligent Robot Systems (IROS)*, (Lausanne, Switzerland), Oct. 2002.