VIPER: Virtual Intelligent Planetary Exploration Rover

Laurence Edwards **Lorenzo Flückiger*** **Laurent Nguyen†** **Richard Washington‡**
Autonomy and Robotics Area, NASA Ames Research Center, Moffett Field, CA 94035
{ edwards | lorenzo | nguyen | richw } @artemis.arc.nasa.gov

Keywords: Simulation, 3D visualization, plan execution, planetary rovers.

Abstract

Simulation and visualization of rover behavior are critical capabilities for scientists and rover operators to construct, test, and validate plans for commanding a remote rover. The VIPER system links these capabilities, using a high-fidelity virtual-reality (VR) environment, a kinematically accurate simulator, and a flexible plan executive to allow users to simulate and visualize possible execution outcomes of a plan under development.

This work is part of a larger vision of a science-centered rover control environment, where a scientist may inspect and explore the environment via VR tools, specify science goals, and visualize the expected and actual behavior of the remote rover.

The VIPER system is constructed from three generic systems, linked together via a minimal amount of customization into the integrated system. The complete system points out the power of combining plan execution, simulation, and visualization for envisioning rover behavior; it also demonstrates the utility of developing generic technologies, which can be combined in novel and useful ways.

1 Introduction

Imagine trying to drive when your vehicle only does approximately what you command, you only catch occasional glimpse of your environment, 20 minutes pass between your command and the vehicle's response, and to top it off, you don't really know how your vehicle works. This is the world of scientist-directed planetary rover exploration.

Planetary rovers are scientific tools for exploring an unknown world. One focus of the Autonomy and Robotics Area (ARA) at the NASA Ames Research Center is to design and develop the tools and techniques that allow scientists to control a rover efficiently and effectively. This presents challenges both in the user interface and in the underlying rover control methods.

One important element of the planetary rover control is the ability to simulate and visualize possible execution outcomes of a plan under development. We have developed the VIPER system, which links plan execution, rover simulation, and a high-fidelity, realistic virtual-reality (VR) environment. This system is one part of a larger architectural design under development that includes tools for science goal specification and plan generation.

The ultimate vision for the overall architecture is that scientists at "mission control," and potentially elsewhere in the world, will both specify and observe the rover's operation as well as science products through the VR environment. The scientists can examine physical features of the environment (distance, volume, cross-sections) and specify science-level goals, for example to go to a rock and drill a small sample. These goals are then interactively refined at mission control with the help of a planning and scheduling system, adding constraints of rover motion, resources, and time to arrive at a final plan. Once the plan is ready, it is communicated to the rover.

On board the rover, the plan is executed by testing and monitoring conditions on time, resources, and rover and environmental state. The same plan executed multiple times may produce many different behaviors, based on the initial conditions and the variability of the rover's interactions with the environment.

The VIPER system allows users to simulate and visualize possible execution outcomes of a plan under development. We have developed a kinematically accurate simulator of the rover that allows the sci-

*NASA contractor with QSS.

†Author's current address is LightLogic, Inc., 8674 Thornton Avenue, Newark, CA 94560.

‡NASA contractor with RIACS.

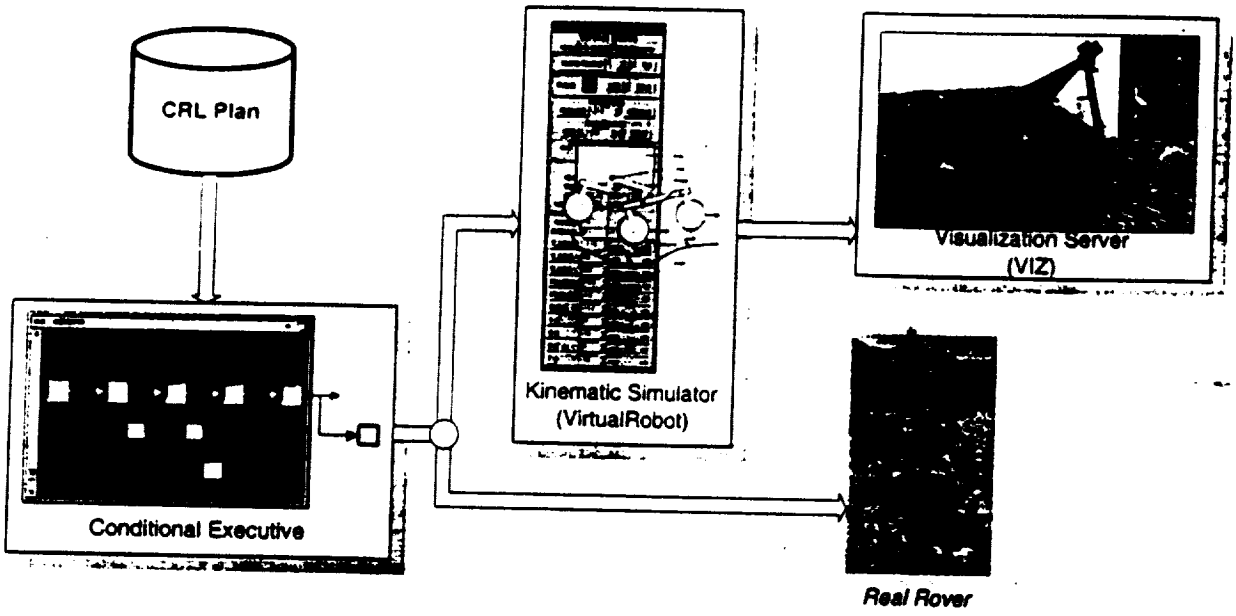

Figure 2: Overview of VIPER plan execution, simulation and visualization.

to MarsMap, initially deployed during the Mars Polar Lander mission [Nguyen *et al.*, 2001]. Viz implements an architecture that allows a flexibility and customizability similar in spirit to VEVI and presents the user with a highly interactive immersive environment as in MarsMap.

Robot Behavior Simulation The behavior of the mechanism is reproduced by the VirtualRobot simulator. VirtualRobot was initially developed at the Swiss Federal Institute of Technology, Lausanne (EPFL) by the Virtual Reality and Active Interfaces Group (VRAI) as an interactive tool to control and study any kind of robot manipulator [Flückiger *et al.*, 1998; Flückiger, 1998]. VirtualRobot was based on a generic kinematic generator. The collaboration of the VRAI Group with the Autonomy and Robotics Area of NASA Ames led to extensions of VirtualRobot that enable the simulation of rovers in addition to robot manipulators.

Plan Execution The plan execution component of this work was inspired in part by work on the Remote Agent (RA), an integrated agent architecture developed for spacecraft control and deployed as an experiment on the Deep Space One mission [Bernard *et al.*, 1998; Muscettola *et al.*, 1998]. The rover executive demonstrates advances in conditional execution compared to the RA executive: the language of the RA executive does not accept conditional sequences, which are critical to the success and effectiveness of a rover mission, given the highly variable interactions of the rover and the environment. The current implementation of the rover executive does not, however, attempt to reproduce all of the capabilities of the RA; in particular, multiple concurrent activities, model-based state reconfiguration, and run-time resource selection for state variables are not currently included in our executive.

VIPER system organization The VIPER system comprises the plan execution, simulation, and visualization subsystems (see Figure 2). These components allow the scientists to explore different possible plans and the expected behavior of the robot in the virtual environment.

The plan execution component interprets the command plan, checking conditions and monitoring runtime requirements of the plan. It sends commands to the rover simulation component and receives state information back. The rover simulation component, in turn, simulates the kinematics of the rover and its interactions with the terrain. The simulator sends pose information to the visualization component, which continually updates its environment model and renders the scene for the viewer.

3 Underlying Technologies

The VIPER system is built on generic technologies for each of its subsystems, which are specialized with data or configuration information to work with the particular robotic platform and environment. This allows the system to be used for visualization, test, and design of different, novel, and even imaginary robotic platforms. In particular, parts of this technology have been used to model and simulate the Pathfinder environment, the Mars Polar Lander robotic arm and camera, the NASA Ames

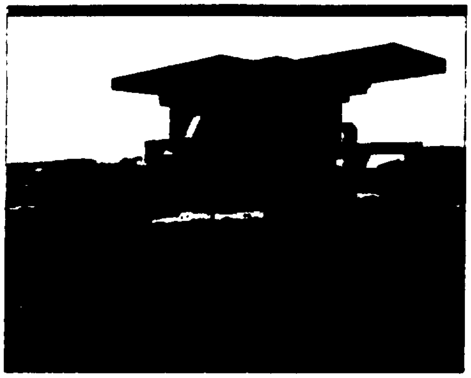

Figure 4: Simulation of K9 rover driving over a rock.

existing science toolset interface providing manual control of the rover and its image sensors — this is used for sequence construction. In addition, a button was added to launch the VirtualRobot simulator. The total effort required was a few days.

3.2 VirtualRobot Simulation

VirtualRobot was initially developed as an interactive tool to control and study any kind of robot manipulator [Flückiger *et al.*, 1998; Flückiger, 1998]. The design of VirtualRobot was driven by the following requirements:

- no code writing nor kinematic model is necessary to describe the kinematic behavior of the robot,
- the program is able to handle the kinematics of any robot manipulator structure in real-time[1],
- an intuitive user interface allows both novices and experts to easily manipulate robots in a virtual environment.

For these reasons VirtualRobot was based on a generic kinematic generator: after reading a text file describing the geometry of the robot, the program builds a numerical solver which computes the direct and inverse kinematics of the robot. The robot structure can be serial (as most industrial robots), parallel (as a Stewart platform) or hybrid (mix of the previous two). A robot model is created in a virtual environment and the user is able to interact with the robot using intuitive 3D input devices (like a Space-Mouse) or 6 degree-of-freedom force-feedback devices.

VirtualRobot has been extended to enable the simulation of rovers in addition to robot manipulators. The same kinematic solver can now also accommodate multi-wheeled rovers with passive suspensions driving on uneven terrains (see Figure 4). In addition, rather than being controlled by user input, the kinematic solver responds to inputs coming from any other program through the network.

[1] we consider real-time from a human point of view: refresh rate in the range of 20Hz to 200Hz

```
rover K9 {

  base {              // the original location of the rover
    pos ( -0 . 1 . . . )
    ori ( 0 0 0 . 90 )
  }

  iiodriver "333.5"        // how the rover is controlled

  frame 11 {          // position the first boogie
    pos ( -0.110 -0.316 0.205 )
    ori ( 90.0 0.0 0.0 )
    pred 99
  }

  link 1 {            // main right bogie
    parameters ( theta 0.0  r 0.0  alpha 0.0  d 0.0 )
    type 2            // revolute joint
    pred 11           // hierarchy ('1' is child of '11')
    range ( -45.0 45.0 )     // limits of the joint
                      // graphic object to represent this link
    filename ( "right_main_bogie.wrl" )
  }
...
  wheel 1 {           // rear right wheel
    type 1            // type of wheel (geometry)
    pos ( 0.0 0.0 0.0 )
    ori ( 90.0 0.0 0.0 )
    pred 13
    filename ( "right_wheel.wrl" )
  }
...
  constraint {        // the end-effector '5'
    frame 5           // must follow the input
    sensor ( 0 1 )    // described as sensor '1'
  }
...
} // end of rover description
```

Figure 5: Excerpts of a robot file description used by VirtualRobot to build and simulate a rover

The following two sections briefly review the robot description file and the kinematic solver used in VirtualRobot.

Robot description file

Any robot manipulator, rover or combination of both can be described in a human readable text file which is parsed by VirtualRobot. This file contains all the geometric properties of the mechanical structure to be simulated. The robot structure can be expressed with the Khalil-Kleinfinger formalism [Khalil and Kleinfinger, 1986] (which is also usable for multi-branch structures, unlike the well known Denavit-Hartenberg [Denavit and Hartenberg, 1955] notation) or by using regular reference frames (3 translations + 3 rotations). The robot is defined as a tree structure from the base up to each end-effector or wheel. For robots with kinematics loops, the desired chain is closed by defining an additional constraint between two branches of the tree. Similarly, declaring a constraint between any body of the structure and an input (3D device, network, etc.) will make the VirtualRobot program compute the appropriate inverse kinematic behavior of the structure to satisfy these constraints.

The use of a generic description file (see Figure 5 for an example), to define robots allows rapid simulation creation for new robot and rover structures

```
(block :id plan
       :node-list
       ((task :id drive-1a  :comment "drive 2.8m"
              :action baseDrive  :parameters (0.2 0.0 0 2.8))
        (task :id turn-1b  :comment "turn right 75 degs"
              :action baseDrive  :parameters (0.0 0.3 0 1.309))
        (task :id drive-1c  :comment "drive 0.5m"
              :action baseDrive  :parameters (0.2 0.0 0 0.5))
        (task :id mosaic-1  :comment "take mosaic of Yogi"
              :action icmosaic
              :parameters ("test1/images/pan1/p3_003"
                           3 0 0 640 480 3
                           0.1 -0.8 0.7 -0.4 0.2 0.3 1))
        (branch :id branch-on-time
                :comment "chooses route based on time"
                :options
                ((option :id opt1
                         :eligible-conditions ((time 0 60))
                         :utility 1.0
                         :node
                         (block :id branch-1
                                :node-list
                                ((task :id drive-2a
                                       :comment "drive -0.5m"
                                       :action baseDrive
                                       :parameters (-0.2 0.0 0 0.5))
                                 ...
                                 )))
                 (option :id opt2
                         :eligible-conditions
                         ((time 60 :plus-infinity))
                         :utility 0.5
                         :node
                         (block :id branch-2
                                :node-list
                                ( .
                                 ...
                                 )))
                ))))
```

Figure 6: Excerpts of a CRL plan.

tions are respected. CX can be instructed to wait for a subset of the preconditions (for example, the start time window) rather than failing the execution of the node. CX receives state information from the low-level rover control (or simulation). In the envisioned overall rover architecture, CX will also receive higher-level state information from a state-identification module and resource information from a resource manager. It uses this information to check preconditions and maintenance conditions, as well as to check the eligibility conditions of the plans in the alternate plan library.

At each point in time, CX may have multiple options, corresponding to the eligible options of a branch point and the enabled alternate plans. CX chooses the option with the highest estimated expected utility, computed over the remainder of the plan. In the current implementation, the utility of successfully completing an atomic action is fixed and set by operators at mission control. From this atomic utility and a model of the probabilities of various events (such as a traverse taking longer than anticipated), the expected utility of an entire branching plan can be calculated.

When plan execution fails, CX reacts as specified in the node, either ignoring the action or aborting the execution and checking for applicable alternate plans. If execution is aborted and no alternate plans apply, CX aborts the entire plan set and puts the rover into a stable standby mode; all operation is suspended and the rover awaits further plans from mission control.

CX and CRL were incorporated into the rover autonomy architecture used to control the Marsokhod rover during a February 1999 field test [Bresina *et al.*, 1999] and the K9 rover during a May 2000 field test [Bresina *et al.*, 2001].

Customization of CX and CRL for VIPER

The CRL grammar is generic; only the command and condition names change from one application to another. The CX execution semantics rely on the general CRL properties and not the command and condition names. As such, the central "execution engine" is completely generic. The only specializations needed are in terms of communication with the externally controlled rover (or simulation). These are accomplished with C++ subclassing that is completely transparent to the execution engine.

In order to control the K9 rover, the CRL "command dictionary" was defined, as well as routines that CX calls to translate CRL commands to messages to the rover. To incorporate CX and CRL into VIPER, the same command dictionary was used, but the translation routines were changed to communicate with the simulation rather than the actual rover. In all, the total effort was no more than a week, much of that to separate out the parts common to the actual and simulated rover to avoid code duplication.

4 Illustrative Example

Consider a simulation of a rover moving in the Mars Pathfinder environment. VIPER presents the viewer with three main windows that show: 1) a 3D visualization of a mobile robot at the Mars Pathfinder landing site executing a plan, 2) a display showing the VirtualRobot parameters, and 3) a 2D text display of the robot executive status. See Figure 2 for representative screen images.

4.1 Scenarios

Consider the scenario where the rover is located next to the lander, as in Figure 7. The next goals to achieve may include getting a close-up mosaic of Yogi, the largest rock in the environment, followed by a traverse around the lander to near the second ramp (Ramp2). Depending on the data storage available after the mosaic ends, the rover may decide to traverse via either side of the lander; one has more rocks that are interesting scientifically, the other has fewer. In either case images will be acquired of the targets during the traverse. If the time is too short, the rover will remain near the first lander ramp (Ramp1).

These three main scenarios, shown schematically in Figure 8 and graphically in Figure 7, are the result of data, power, and time limitations on plan execution.

The first scenario illustrates the effects of a time shortage in the absence of any data storage short-

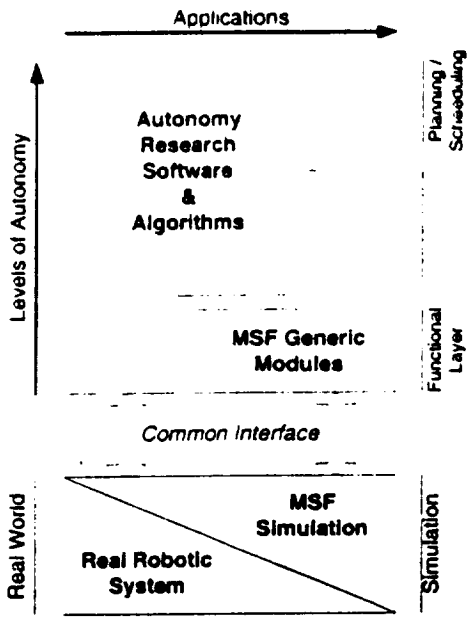

Figure 9: Scope of MSF.

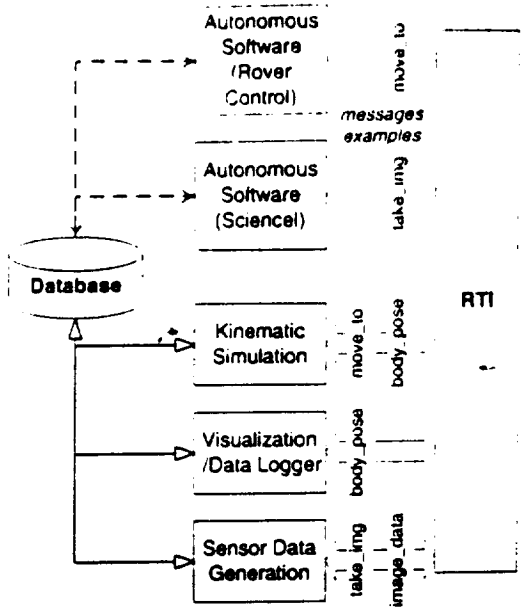

Figure 10: MSF Sample Simulation Instantiation.

time simplify their integration into a more reliable simulation.

As shown in Figure 9, MSF will provide simulation environments which could be used at different level of integration. For example a research lab could have a complete system from the top level autonomy to the low level hardware control: in this case MSF will only provide a replacement for the robotic hardware and the environment with the simulation. On the other hand, one could need to test a very high level autonomy component, without having the rest of the system: in this case, MSF will also provide generic components replacing the missing parts.

From an implementation point of view, MSF will rely on the publish/subscribe scheme used in HLA: each component of the simulation will communicate with the other components using a standard set of objects/messages defined for the purpose of MSF. The HLA Run-Time-Infrastructure (RTI) manages all the communications between the participants of the simulation. The RTI also provides facilities to address the problem of Time-Management which is a key point in a simulation like MSF because its different components are not necessarily designed to run in real time.

The Figure 10 shows a simple simulation example: it is composed of several components which are all connected to the RTI for communication. They also have access to a separate database to avoid overloading the network when accessing large objects like images. The autonomous software is decomposed into two distinct objects for clarity: each uses a different set of messages to interact with the simulation. Consider in this example that they send commands to a simulated robot. The kinematic simulator (like VirtualRobot) will then compute the behavior of the robot in response to the commands and return various sensor updates. If the autonomous software requests the acquisition of scientific data (like an image), then the sensor data generator will return either simulated data or real data extracted from a database. Visualization tools (like VIZ) could also be connected to the simulation to help the user evaluate the behavior of the autonomous system.

The development of MSF will provide a set of tools (simulation components / robotic systems library) that should speed up the testing process for the developers of autonomous systems. In addition, the proposed framework will help advance the development of standardization of communication between autonomous software and robotic hardware.

References

[Bares and Wettergreen. 1999] J. Bares and D. Wettergreen. Dante II: Technical description, results and lessons learned. *International Journal of Robotics Research*, 18(7):621–649, July 1999.

[Bernard *et al.*, 1998] Douglas E. Bernard, Gregory A. Dorais, Chuck Fry, Edward B. Gamble Jr., Bob Kanefsky, James Kurien, William Millar, Nicola Muscettola, P. Pandurang Nayak, Barney Pell, Kanna Rajan, Nicolas Rouquette, Benjamin Smith, and Brian C. Williams. Design of the remote agent experiment for spacecraft autonomy. In *Proceedings of the 1998 IEEE Aerospace Conference*, 1998.

[Bresina *et al.*, 1999] J. L. Bresina, K. Golden, D. E. Smith, and R. Washington. Increased flexibility