Simulation of a Rover and Display in a Virtual Environment

Adam Sweet (University of California, Berkeley), Theodore Blackmon (NASA Ames Research Center),
and Vineet Gupta (NASA Ames Research Center)
486 Minor Hall, University of California, Berkeley, CA 94720-2020 / USA
e-mail: sweet@newton.berkeley.edu
Tel: 1-510-642-3509

Abstract

This paper presents a dynamic simulation of a robotic vehicle operating in an unrestricted environment. The simulation program, based on a NASA planetary rover, is composed of two parts: a virtual reality interface and a simulation module. The virtual reality interface is used for creating a rover command sequence and for visualization of the simulation results; the dynamic simulation module takes the command list and uses it to predict the path and state of the rover. This paper explores each of these topics in detail and examines the simulation results.

1.0 Introduction

Our motivation for this study has evolved from the work on the Mars rover in 1997 and the continuing work on the NASA Ames Mars Mapping program. The goal in creating this rover simulation is to create a planning tool for mission controllers using supervisory control of a planetary rover. Supervisory control is the method of controlling virtually all planetary rovers, as neither direct manual control or autonomous control are feasible during a mission. Disadvantages of direct manual control are the continuing occupancy of the attention of the human operator and of the limited communication channel. On the other hand, the technical requirements for robust autonomous control of a rover in a complex environment have not yet been achieved. Communication with a planetary rover during a mission only takes place for small amounts of time at widely spaced intervals. It is therefore desirable to test the command sequences in a simulation on Earth before they are sent to a rover on another planet.

The present study has focused on the simulation of a rover vehicle and its kinematics and dynamics. To accomplish this, we have utilized maps and embedded organization of visual representations of terrain as the appropriate working environment for the simulated vehicle. The object of our simulation is slightly different from other recent simulations (Hacot, [1]): instead of analyzing the rover's ability to follow a particular path, we wish to begin from a list of rover commands and simulate the rover's response to the commands.

The rover that we are simulating is named the Marsokhod, and is kept at the NASA Ames Research Center. A picture of the rover is shown below in Figure 1. The Marsokhod was brought to Ames in late January of 1993 and has since been used in several Ames field tests on Earth.

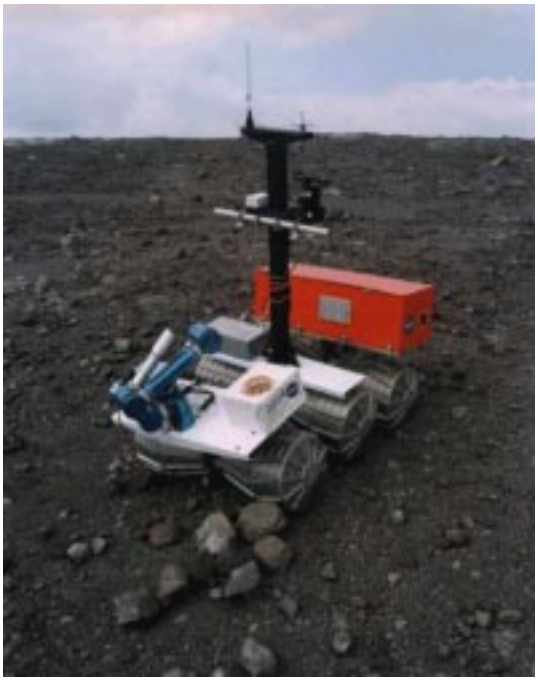


Figure 1: Picture of Marsokhod rover during NASA Lunar and Mars 1995 tests (conducted in Hawaii Volcanoes National Park on the Big Island of Hawaii)
Photo credit: Edward Schilling, NASA Ames Research Center

The simulation of the rover is a project in progress. The eventual goals are to develop dynamics of the rover travelling across terrain. Also, we will create models of the rover's power and control systems, to aid in modeling the response of the rover to the commands. Currently, we have worked out a set of simplified kinematics and dynamics for the rover, in order to test the software framework. As the project progresses, the kinematics and dynamics will be developed more fully.

2.0 Simulation Software

We have implemented the software framework that the simulation will be using. It consists of a combination of two previously existing programs, MarsMap and HCC. MarsMap is a virtual reality program developed by Theodore Blackmon (NASA Ames), using World Toolkit v. 8 (by Sense8 Corp.). It gets its name from its primary function, as a mapping and visualization tool for Mars rover missions. First, MarsMap loads in and displays 3-D models of the terrain surrounding the landing site. It also displays a reference grid above the terrain. The user can then perform mapping functions such as getting the coordinates of points on the Martian surface, finding the distances between points, and finding directions to objects. The user can also change the point of view in MarsMap, to see the terrain from different angles. Finally, MarsMap has the ability to display a rover and steer it across the terrain. A screen shot of MarsMap displaying the Marsokhod rover is shown below in Figure 2. For more information on MarsMap, see Stoker and Blackmon [4], [5].

Figure 2: Screen shot of MarsMap with Marsokhod rover

Source: Ted Blackmon, NASA Ames
(http://img.arc.nasa.gov/~blackmon/Tmp/marsoVR.gif)

The simulation is executed using a program called HCC. Simply put, HCC is a scripting language that allows for modeling of both continuous differential equations and discrete events. The language is object-oriented, and designed for easily read code. The basic flow of HCC is listed below in Figure 3:

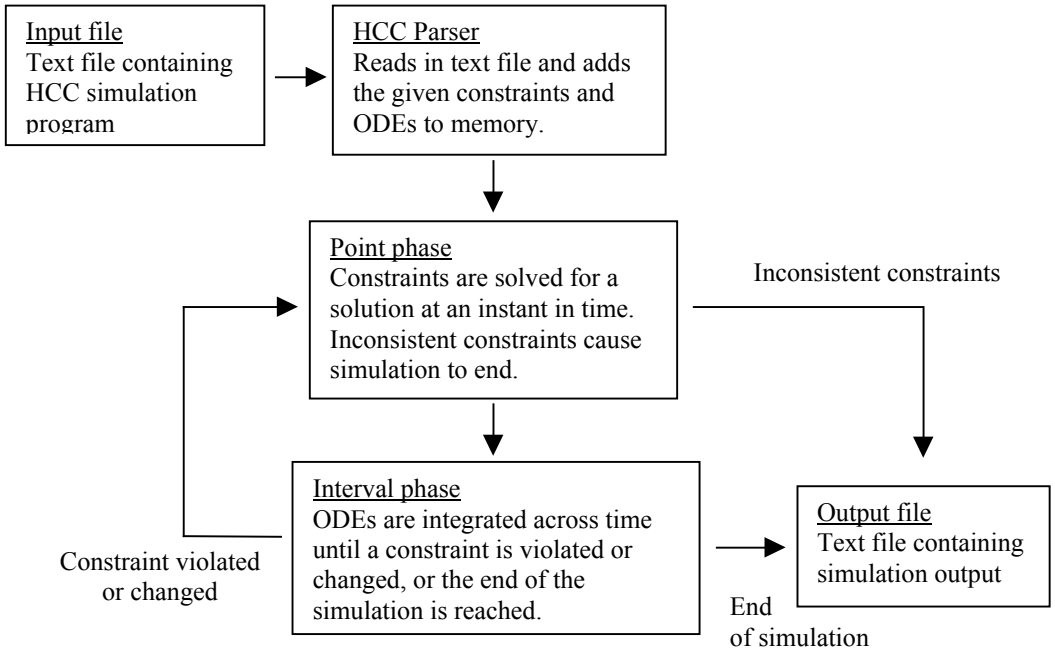


Figure 3: Flow of the HCC program

An HCC simulation is first created as a text file with the model to be simulated, written in the HCC scripting language. When HCC begins, it first reads in this file, and parses the given differential equations (ODEs) and constraints into the memory of the computer. It then begins to execute the

simulation. The execution of the simulation has two distinct modes: point phases and interval phases. In point phases, the equations and constraints are solved for solutions at a particular instant in time. Once a solution is found, HCC enters an interval phase. In the interval phase, time progresses, and the ODEs in the simulation are numerically integrated. The simulation remains in the interval phase until a constraint is violated or changed. This can happen as a result of either the variables reaching values that directly conflict with given constraints, or by the variables reaching values that trigger the simulation to change the equations and constraints. The program then reenters a point phase and attempts to solve the new constraints for a solution. This process is repeated until either a valid solution cannot be found, or the end of the simulation is reached. For a more in-depth discussion of HCC, see Carlson [2] or Gupta [3].

We created the simulation program by combining MarsMap and HCC. It uses the visualization tools of MarsMap, and also uses HCC as a simulation engine. In order to accomplish this, we compiled HCC as a set of external library functions. The simulation program is built from MarsMap, with the ability to call HCC in order to run a simulation. Also, during the simulation, several of the MarsMap functions are in turn called by HCC to complete the simulation, as indicated in the block diagram (Figure 4):

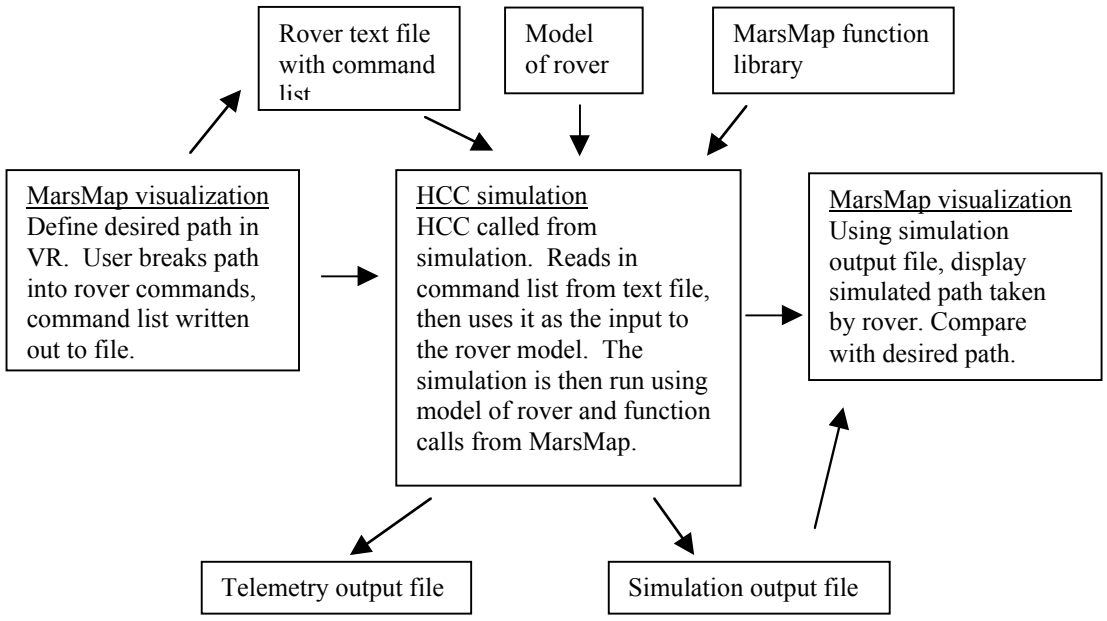


Figure 4: Block diagram of simulation program flow

3.0 Rover Model

3.1 Kinematics

In the simplified kinematic model of the rover (Figure 5, below), the vectors $\mathbf{E}_i$ represent the reference coordinate frame for the simulation, and the vectors $\mathbf{e}_i$ represent the coordinate frame relative to the rover. The model has two main rods that join three axles. The front and rear rods are free to rotate about $\mathbf{e}_2$: these angles are labeled θ_{13} and θ_{14}, and are defined as right-hand rotations about $\mathbf{e}_2$. Each of the three axles is free to rotate perpendicularly to the rods: these angles are labeled (from the front to the rear) as θ_{15}, θ_{16}, and θ_{17}, and are defined as right-handed rotations about $\mathbf{e}_1$. The variables x, y, and z represent the coordinates of the center of the rover on the surface, and ϕ denotes the heading angle of the rover (shown later in Figure 6).

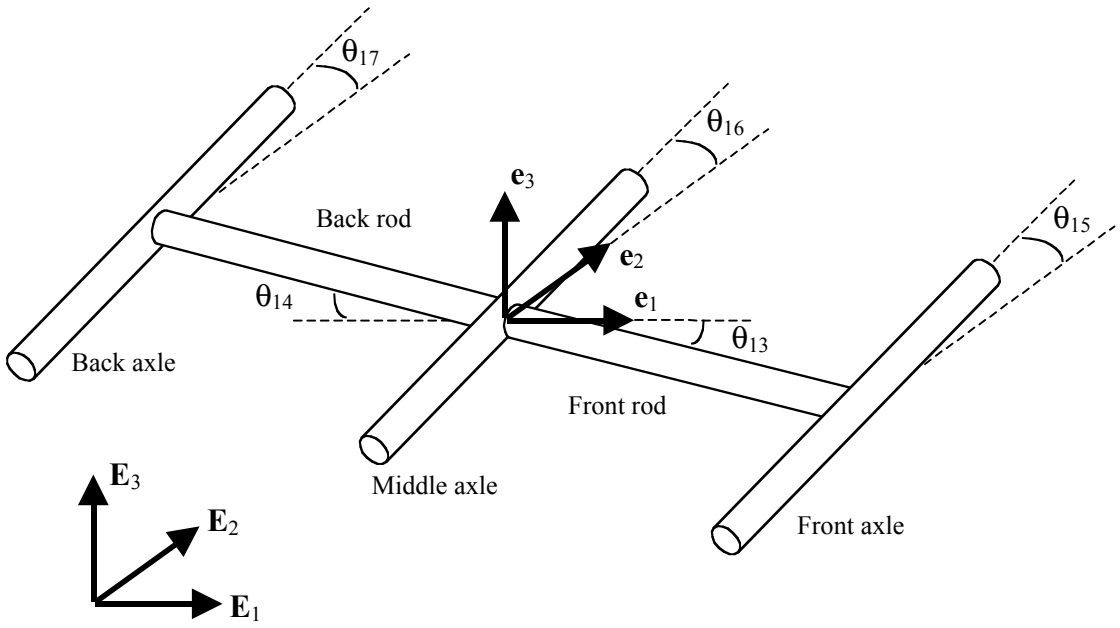


Figure 5: Simplified model of Marsokhod, with kinematic angles

The configuration of the rover is given by the five angles θ_{13}-θ_{17}, and is determined by the terrain beneath the rover. The equations used to calculate the rover configuration are given below, in (1). In those equations, FL.ht represents the height of the front left wheel, and similarly FR.ht = front right wheel, ML.ht = middle left wheel, MR.ht = middle right wheel, BL.ht = back left wheel, BR.ht = back right wheel. Also, front_length is the length of the front rod, back_length is the length of the back rod, and axle_length is the length of the axles.

$$
\begin{aligned}
z &= \frac{(ML.ht + MR.ht)}{2} & \theta_{15} &= \frac{FL.ht - FR.ht}{axle\_length} \\
\theta_{13} &= (z - \frac{FL.ht + FR.ht}{2}) / front\_length & \theta_{16} &= \frac{ML.ht - MR.ht}{axle\_length} \\
\theta_{14} &= (\frac{(BL.ht + BR.ht)}{2} - z) / back\_length & \theta_{17} &= \frac{BL.ht - BR.ht}{axle\_length}
\end{aligned}
\qquad (1)
$$

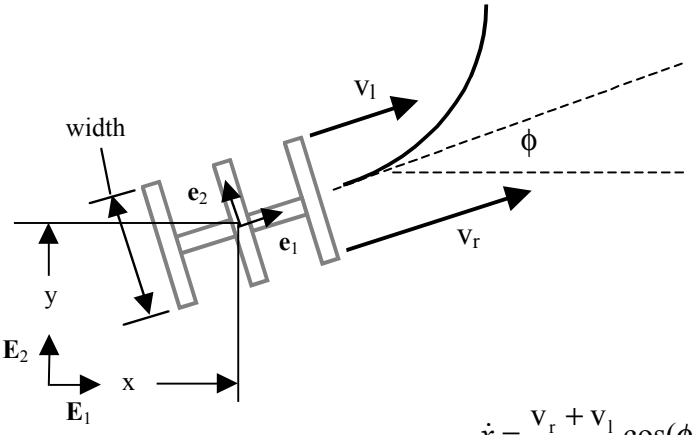
There are two basic simplifications made in writing (1). The first is that the angles θ_{13}-θ_{17} will remain small, so that the small angle sine approximation is valid. The second rises from the method of calculating the height of each wheel. First, the (x,y) coordinates of each wheel are calculated as if the rover was on level ground (all configuration angles equal to 0). Then, the height of the ground at each wheel location is measured. Of course, if the rover is actually settled on the terrain, the wheels will be at slightly different locations, and can have slightly different actual heights.

3.2 Dynamics

Our eventual goal is to develop a complete dynamical model of the rover. We wish to send the simulation the same commands that are given to the real rover. Based on these commands, a model of the rover's control system would calculate the torque output of each of the six wheel motors. Finally, this would be the input to the dynamics of the rover rolling across the given terrain. The dynamics would entail developing models for the rigid body dynamics of the rover as well as the soil-wheel interaction. The soil-tire interaction would likely include a random element to model slipping of the wheels. This would allow the user to run the simulation several times and see the possible range of paths the rover can take based on the commands sent. The amount of randomness added to the simulation would be an empirical value, determined by experiments done with the real rover.

We have implemented a set of simplified dynamics for the rover. The equations are based on the drive command that will be given to the rover. In the drive command, the user defines a desired translational speed and rotational speed for the rover, to be executed simultaneously. The translational speed is the rate at which the rover moves along the path, and the rotational speed is the rate of change of the heading angle. By adjusting these two parameters, the rover can travel along a circular arc of any radius. For example, setting the translational speed to zero with a nonzero rotational speed will cause the rover to rotate in place; contrawise, setting the rotational speed to zero with a nonzero translational speed will cause the rover to travel in a straight line. When both the translational and rotational speeds are set to nonzero values, the rover will travel along a circular arc.

Based on the translational and rotational speeds given in the command, the simulation assigns each wheel a nominal speed according to Figure 6(a). The left wheels are each assigned the speed $\mathbf{v}_l$, and the right wheels are each assigned the speed $\mathbf{v}_r$. Then, the speed of the rover in the x and y directions is calculated according to Figure 6(b).



$$v_l = translation - \frac{width}{2}(rotation)$$

$$v_r = translation + \frac{width}{2}(rotation)$$

(a)

$$\dot{x} = \frac{v_r + v_l}{2}\cos(\phi)$$

$$\dot{y} = \frac{v_r + v_l}{2}\sin(\phi)$$

$$\dot{\phi} = \frac{v_r - v_l}{width}$$

(b)

Figure 6: Basic relations between (translation, rotation) of drive command and (x, y, ϕ) used for dynamics.

The actual expressions used for $\mathbf{v}_l$ and $\mathbf{v}_r$ in Figure 6(b) are slightly more complicated than those given in Figure 6(a). They are shown below in Eq. 2. Each wheel is set to a nominal value of $\mathbf{v}_l$ or $\mathbf{v}_r$. However, the simulation also allows a wheel to be stuck. In that case, the velocity of the stuck wheel is set to 0, regardless of the drive command. The wheel velocities on each side of the rover are averaged to find a net velocity of that side. Then, if one of the wheels is set to be stuck, it will be dragging and will tend to pull the rover to that side. The wheel speeds are also adjusted by the pitch angles of the rover, θ_{13} and θ_{14}. This means that only the velocity component in the x-y plane is used in the equations, which would come into account if the rover is climbing a hill.

$$\begin{aligned}
\mathbf{v}_l &= \text{ML.linear_velocity}\,?\cos((\theta_{13}+\theta_{14})/2) + \text{FL.linear_velocity}\,?\cos(\theta_{13}) \\
&\quad + \text{BL.linear_velocity}\,?\cos(\theta_{14}) \\
\mathbf{v}_r &= \text{MR.linear_velocity}\,?\cos((\theta_{13}+\theta_{14})/2) + \text{FR.linear_velocity}\,?\cos(\theta_{13}) \\
&\quad + \text{BR.linear_velocity}\,?\cos(\theta_{14}) \\
&\text{ML} = \text{middle left},\ \text{MR} = \text{middle right}, \text{FL} = \text{front left}, \text{FR} = \text{front right}, \\
&\text{BL} = \text{back left}, \text{BR} = \text{back right}
\end{aligned} \tag{2}$$

3.3 Power

We have implemented a simple model of the rover's power systems. It includes a battery containing a certain amount of charge, and the drive motor on each wheel consuming power. The total power consumed is the sum of the power consumed by each wheel, and the rate of change of the battery charge is equal to the total power consumed. Their equations are:

$$\begin{aligned} \text{total\_power\_drain} = {} & \text{FL.power\_drain} + \text{FR.power\_drain} + \\ & \text{ML.power\_drain} + \text{MR.power\_drain} + \\ & \text{BL.power\_drain} + \text{BR.power\_drain} \end{aligned} \tag{3}$$

$$\frac{d}{dt}(\text{battery\_charge}) = \text{total\_power\_drain}$$

4.0 Implementation and Results

We implemented the dynamics described above into an HCC program. Each part of the physical rover is defined as an object in the simulation. Each object has member functions that perform tasks related to the physical object. A number of rover commands have been implemented (Table 1).

Finally, there are several faults that can be modeled by the simulation. A fault is a general term used to describe a problem with a rover. For our purposes, it means that a part of the rover's hardware is malfunctioning. We have implemented simple models for three different faults: a stuck wheel, a broken wheel encoder, and a skipping wheel encoder. In each case, the normal operation of the simulation is modified to reflect the fault. Also, each of the faults can be created and cancelled by the user dynamically during the simulation.

Table 1: Rover commands implemented into simulation program

Module	Command	Arguments	Description
MarsoBaseController	BaseDrive	(*CmdID*, *translation*, *rotation*, *time*, *distance*)	Instructs the rover to drive with a translational velocity of *translation* and a rotational velocity of *rotation*. This will continue for *time* seconds, or until the rover has traveled *distance* meters.
	BaseAbort	(*CmdID*)	Aborts the current driving command.
	BaseWaitDone	(*CmdID*)	Toggles the flag that indicates if the next BaseDrive command should wait until the current command finishes.
	BaseDelay	(*CmdID*, *time*)	Sets the time delay between command executions, in seconds.
MarsoWheelClass	ResetEncoder	( )	Resets the encoder value for the wheel to 0.
	SetWheelStuck	( )	Sets the wheel to be in a stuck mode – it will not turn, but will still consume power.
	SetEncoderBroke	( )	Sets the encoder value to 0 regardless of the rotation of the wheel
	SetWheelSlip	( )	Sets the wheel encoder to be a number smaller than its correct value
	ClearWheelFault	( )	Clears the wheel fault, for the wheel to become unstuck.

Below are some results from the simulation. The initial position and heading of the rover were

$$x_{init} = -4.5,\ y_{init} = 3.5,\ \phi_{init} = 0.0$$

The commands sent to the rover were:

wait 1 do M.Base.BaseController.BaseDrive (11, 2.0, 0.0, 2, 5),
wait 5 do M.Base.BaseController.BaseDrive (15, 3.0, 1.0, 2, 5)

The first command will instruct the rover to drive straight in the positive x direction at a speed of 2.0; the second command will instruct the rover to drive in an arc with a translational speed of 3.0 and a rotational speed of 1.0. (Because of the nature of the current dynamics, the units on these values are scaleable). The simulated path of the rover is compared with the commanded path in Figure 7:

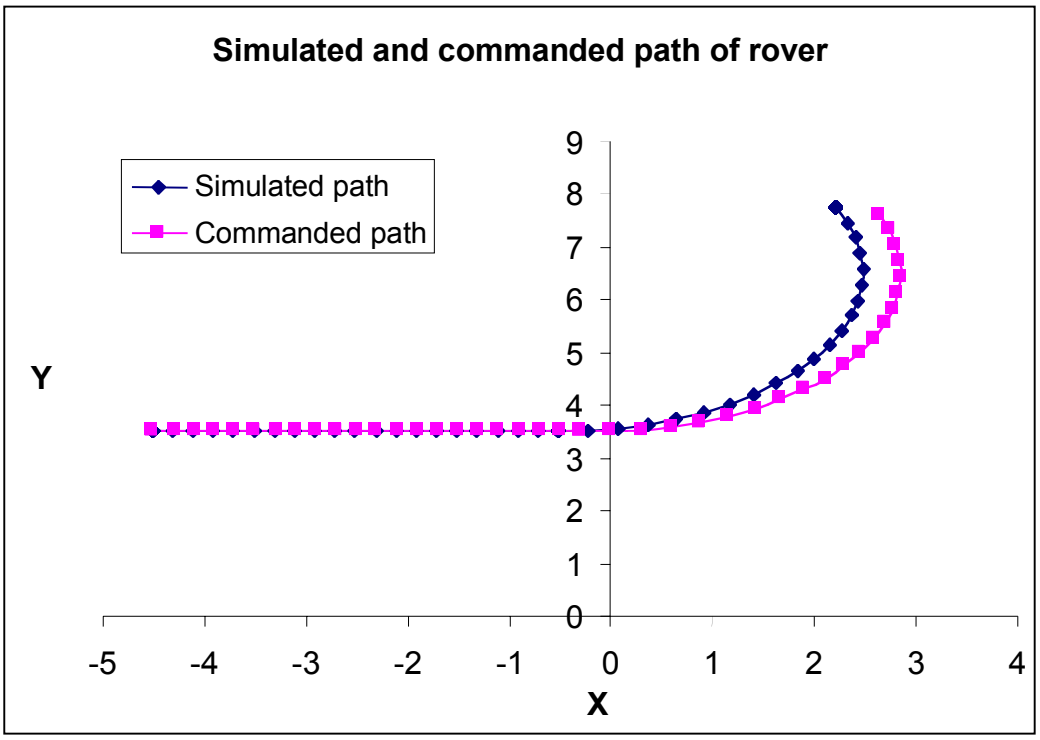


Figure 7: **Comparison between commanded path and simulated path of rover traversing over terrain**

There is variation between the simulated path and the commanded path. The difference is a result of the terrain defined beneath the rover. In this case, the simulation was run with a small slope defined in the right half of Figure 7. When the rover was travelling on the slope, it followed the commanded distance along the slope: this results in some error when the path is projected down on to the x-y plane. Hence, this example shows that the simulation is returning results that make physical sense.

5.0 Conclusion

The rover simulation is a project in progress. The main task accomplished to date is the creation of the software framework used to run the simulation. We accomplished this by merging two existing programs, MarsMap and HCC, and utilizing components of both to create a rover simulation in a virtual environment. Also, we tested the simulation by implementing a simple set of kinematics and dynamics based on rover commands. The planned future work is to more fully model the dynamics of a rover, including the rigid-body dynamics, power systems, and soil-tire interactions. At the conclusion of the project, the rover simulation will be a useful planning tool for mission controllers exploring remote sites and other planets.

Acknowledgements

We would like to thank Dr. Lawrence Stark for his advice on creating this paper, and Debbie George and Michelle Lent for their proofreading and insightful comments.

References

[1] Herve Hacot. "Analysis and Traction Control of a Rocker-Bogie Planetary Rover." Department of Mechanical Engineering, Massachusetts Institute of Technology. Published by MIT Document Services, 77 Massachusetts Ave, Cambridge, MA 02139-4307, June 1998.

[2] Bjorn Carlson, Vineet Gupta. "Hybrid CC and interval constraints." Proceedings of the International Workshop on Hybrid Systems: Computation and Control HSCC'98, Berkeley, April 1998, edited by Tom Henzinger and Sankar Sastry, Springer Verlag, LNCS 1386, pgs 80-94.

[3] Vineet Gupta, Radha Jagadeesan, Vijay Saraswat. "Computing with Continuous Change." Science of Computer Programming, Vol 30, No 1-2, pages 3-50, 1998.

[4] Stoker, C., Blackmon, T.T., Hagan, J. Kanefsky, B. and others. "MarsMap: Analyzing Pathfinder Data Using Virtual Reality" IN: 1997 Fall Meeting, American Geophysical Union, San Francisco, CA, Dec. 8-12 1997. vol. 78, no. 46. p. F403.

[5] Stoker, C., Zbinden, E., Blackmon, T.T. et. al. "Analyzing Pathfinder data using virtual reality and super-resolved imaging." In Press: Journal of Geophysical Research. Special Issue on Mars Pathfinder, 1999.