MISSION PLANNING AND NAVIGATION SUPPORT FOR LUNAR AND PLANETARY EXPLORATION

by

JOSEPH R. ESSENBURG

B.S. Engineering Physics, Taylor University, 2002

B.A. Mathematics, Taylor University, 2002

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2008

Signature of Author _____
Department of Mechanical Engineering
September 28, 2008

Certified by _____
Dava J. Newman, Thesis Supervisor
Professor of Aeronautics and Astronautics & Engineering Systems
Director, Technology and Policy Program

Certified by _____
Steven Dubowsky, Thesis Supervisor
Professor of Mechanical Engineering
Director, Field and Space Robotics Laboratory

Accepted by _____
Lallit Anand
Professor of Mechanical Engineering
Chairman, Department Committee on Graduate Students

MISSION PLANNING AND NAVIGATION SUPPORT FOR LUNAR AND PLANETARY EXPLORATION

by

JOSEPH R. ESSENBURG

Submitted to the Department of Mechanical Engineering on August 28, 2008, in partial fulfillment of the requirements for the Degree of Master of Science in Mechanical Engineering

ABSTRACT

When mankind returns to the moon and eventually voyages to Mars, the ability to effectively carry out surface extra-vehicular activities (EVAs) will be critical to overall mission success. This thesis investigates improving planetary EVAs via a support system to enable optimized mission operations. In order to develop a robustly effective aid capable of performing under the high time pressure, risk, and uncertainty inherent in space exploration, key surface operation factors are examined to understand to best fit role of automated support within complex, changing exploration situations.

A detailed characterization of the makeup and challenges of planetary surface EVAs was used to establish a specific framework for maximizing the productivity of these missions. Recognizing the need for automated support in achieving such optimal performance, the presentation of methods by which all pertinent mission factors may be quantitatively modeled led to creation of a comprehensive automated mission support architecture.

Based on this analysis and motivated by ongoing field testing, a prototype mission support system was developed with twofold intent: both for pre-mission planning and simulation as well as for real-time explorer navigation and re-planning. The prototype presents an intuitive interface where controllers may quickly represent a broad range of mission parameters and scenarios in order to determine a best course of action for immediate execution. Specifically, this system optimizes explorer traverses with respect to given cost functions via a novel implementation of the A* search algorithm. Developed plans may further be linked to a global positioning system to empower real-time team navigation.

Through the completion of experimental EVA simulations involving physical explorers on a remote terrain jointly controlled by a multi-university team, the developed system was shown to robustly respond to situational updates and contingencies to maintain optimal mission performance in near real-time, offering enhanced functionality where preceding systems fell short. The analysis closes with a discussion on the opportunities for such a system as well as potential areas for further improvement.

Thesis Supervisor: Dava J. Newman
Professor of Aeronautics and Astronautics & Engineering Systems
Director, Technology and Policy Program

ACKNOWLEDGEMENTS

For a Midwestern kid who grew up playing with LEGOs, making it through here at MIT has been a dream come true. There are so many people I need to thank who have helped me along the way.

First and foremost, I would like to thank God who has lit the path here and faithfully guided me through every opportunity. I would also like to thank my family for supporting me since the beginning, helping me keep my sanity, occasionally doing my laundry, and making the best meals I've had in the past too many years. Thanks mom and dad for always being there!

Thanks to Dava Newman, Win Burleson, and Kip Hodges for helping give this broad project a clear direction, constantly motivating the work, and coming up with (excessively) abundant ideas for improvement. Also thanks to Steve Dubowsky whose constructive criticism ended up being instrumental in shaping a specific focus for this work.

A huge thanks goes to Jessica Márquez for her support and enthusiasm in answering all of my questions, and to Lasse Lindqvist for making clear sense of all the previous work. Thanks to Daniel Sheehan, without whom this project would not have been possible, for his help with ArcGIS and setting up the navigation system.

Thanks to Uday Kumar and the team at ASU as well as Chris Assad and the team at JPL for their excellent work in helping set up the joint EVA simulations. While I'm thinking of it, thanks also to these teams for being located where they are, providing my best chance to get a tan in the past year.

Another big thanks goes to Brandon Johnson for his terrific work coding A* in Matlab. Thanks to Kate Clopeck for her help in setting up early experiments, and to Sally Chapman for helping keep track of Dava and answering many questions. Most recently, thanks to Marcelo Vazquez for the opportunity to send the system up for testing at Devon Island this summer.

I have major appreciation for my second monitor, upon which this thesis was written, that has saved me from countless hours of minimizing and maximizing windows. I suppose appreciation is also due to Matlab, LabVIEW, ArcGIS, and the Seamless map server; give me a couple months to recover and I'll get back to you on that one. Google Maps is still cool with me though, thanks guys for understanding latitude and longitude, and keeping me from getting lost and letting me see where I'm headed, and even see the people on the street who happened to be walking by that day.

Clearly a massive thanks goes out to Leslie Regan for all her support and especially for still accepting this thesis for Summer graduation despite the current date (which shall remain nameless in this work).

Beyond academics, thanks to the folks at Warwick, Taylor, Ibanez, Fender, Harmony, and Michael Kelly Guitars for giving me something to do in my free time. Thanks also to Sportcraft dartboards in that department.

A huge thank you goes to Alex Edmans, Andrej Kosmrlj, and all of Tang and MIT intramurals. You've kept me young, relaxed, occasionally limping, and relatively in shape over these years. On a related note, thanks also to Michael Collins for remaining cool after a soccer ball made its way through the glass of my 19th story window. And thanks for deeming my room not a fire hazard despite containing a large combustible structure bearing guitar cases, a ceiling fan, and even a TV, carrying live electrical lines and partially blocking the fire sprinkler.

Thanks to the Thirsty Ear for providing employment so that I didn't starve before I had funding, and thanks to Subway, Shinkansen, and Anna's Taqueria for keeping me fed throughout this process. Thanks also to Gatorade, Kellogg's, and Quaker Oats for helping me make it through many sleepless nights.

A great big thanks to Austin Marks and the MIT Gosple Choir, playing bass for you guys was exciting to say the least. Also, thanks to Warren Rohsenow for having a lab named after him in Building 7, being my first cousin twice removed, and letting me crash in Maine, because that's just awesome.

Thank you to Specialized bikes for getting me around in the city, and to Southwest airlines for whenever I left. For in between, thanks to Stafford loans and Honda Financial Services for enabling a completely broke grad student to afford a 2008 V6.

Thanks to the Chicago Cubs for always giving me something to hope for, and also to Boston sports for at least keeping things interesting while I was here. Note this sentiment does not extend to "New England" sports.

I must thank Mark Swanson at Goddard and Bob Davis and Jeremy Case at Taylor for writing what must have been some amazing letters of recommendation to get me here. Thanks to everyone at the MVL for your help and support. Finally, thanks to all my friends here at MIT! Don't worry, I'm still kicking, I'll see you as soon as this gets printed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 MOTIVATION

On January 14, 2004, President George W. Bush announced a new national focus for future space operations. A primary goal of this Vision for Space Exploration is to return manned crews to the lunar surface by 2020, with the purpose of "establishing an extended human presence on the moon" (Bush, 2004). Such an undertaking will be the first of its kind since the Apollo era, and it will serve as a stepping stone for human exploration to Mars and beyond.

The task of developing an outpost on the moon is extraordinary. Of the massive obstacles to be overcome is the imposing extent of extra-vehicular activities (EVAs) necessitated for construction and field exploration. By comparison, the extent of EVAs required to assemble the International Space Station (ISS) became known as the "wall of EVA," as this undertaking significantly overshadowed the total previous experiences from the Gemini program through the initial Space Shuttle missions (Ney & Looper, 2005). In turn, the projected EVA hours required to establish an occupied lunar base has been coined "the mountain of EVA," as this easily dwarfs even the ISS construction (Figure 1.1).

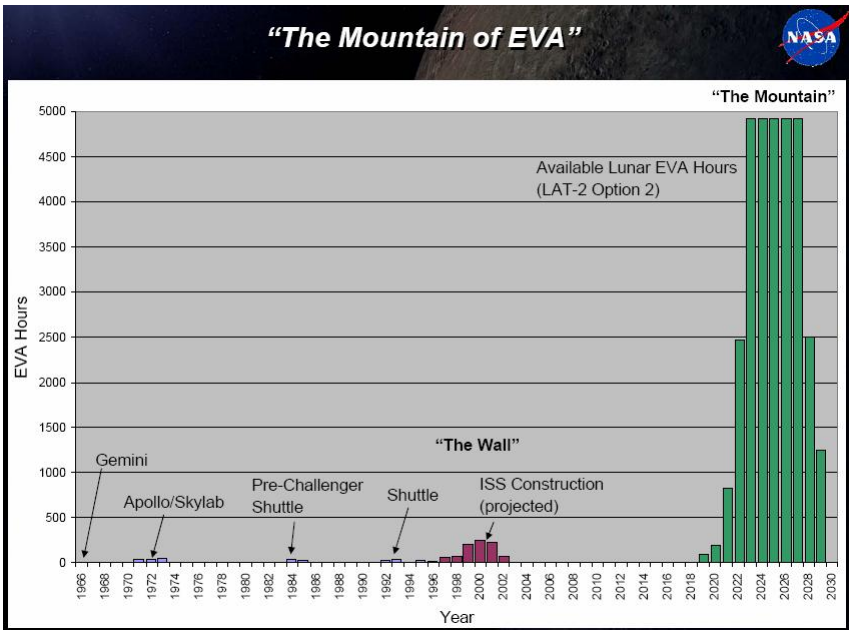


Figure 1.1 EVA hours to establish a Lunar base, "The Mountain of EVA" (Cooke et al., 2007)

To develop a successful planetary operations program, we must leverage our experience from the Apollo era along with nearly half a century of technological advancements. The Apollo EVAs were conducted with minimal support for the explorers in the field. On the traverse of Apollo 14 to Cone Crater, the astronauts experienced fatigue and disorientation as they climbed unexpectedly steep terrain with only a paper map to guide them (Márquez, 2007). The addition of the Lunar Roving Vehicle in later missions enabled broader travel, yet the explorers still lacked support in distinguishing the varying terrain and locating objectives. The Apollo experience clearly demonstrates both the theme of expanding EVA ambition and, in turn, the need for improved planning and support systems (Figure 1.2). On the future lunar surface, EVAs will become routine, daily tasks. Hence, our ability to complete them safely and efficiently will be critical to overall mission success.

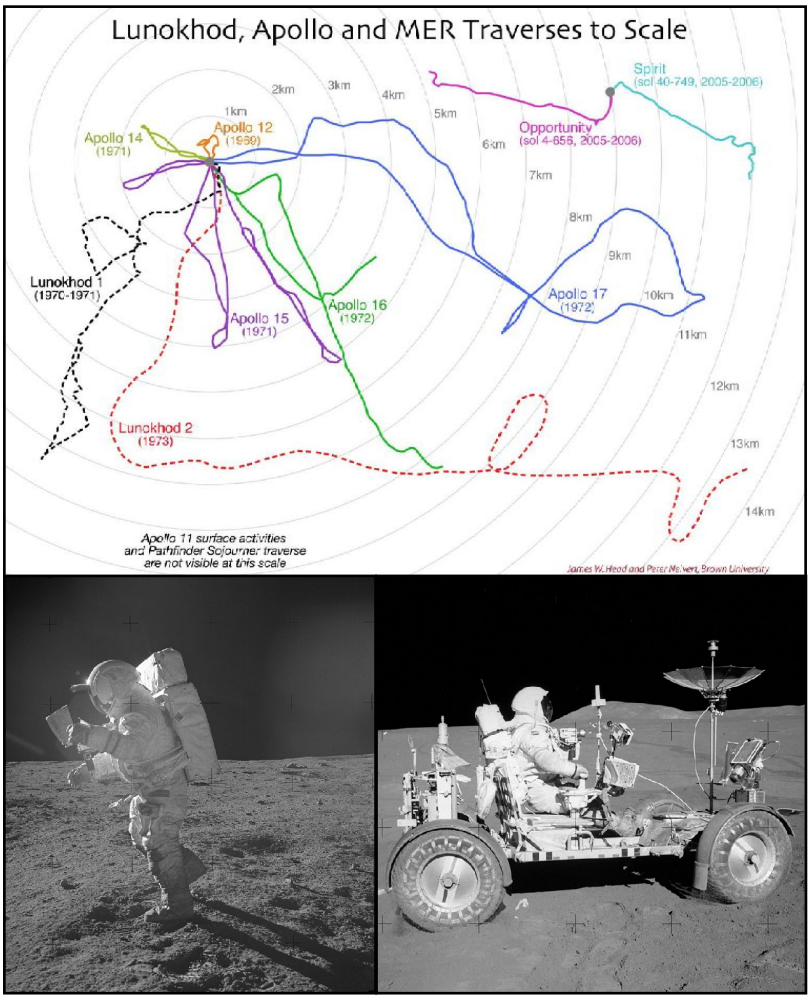


Figure 1.2 Past EVA experiences. Top: Various planetary traverses to scale (Eppler, 2004)
Bottom Left: Apollo 14 astronaut with lunar map (NASA image, AS14-64-9089)
Bottom Right: Apollo 15 astronaut on the lunar rover (NASA image, AS15-85-11471)

As the number and scope of surface operations grow much greater, the ability to maximize crew safety and productivity while minimizing costs becomes essential. This thesis focuses on improving planetary EVAs via optimal mission operations. The task of optimizing planetary operations begins with mission planning. Planners must establish clear objectives and lay out a course of action for achieving them while keeping within a set of constraints. In the case of running an EVA with maximized productivity, mission planners require foreknowledge of the terrain and the ability to quantitatively estimate costs and returns. Especially when several astronauts or robots may be used within a mission, planners need to be able to compare various scenarios and strategies in order to determine the best option. In turn, a traversing astronaut or robot must continuously manage mission information, exploration activities, navigation, safety, and constraints under time pressure and in a hostile, unfamiliar environment. Creating an ideal plan is futile if the field explorers cannot effectively understand and follow it. Furthermore, since surface teams must react to situations in real-time, mission control must also be capable of responding in real-time to offer support. Inherent to exploration are contingencies and unexpected discoveries, and in these cases a new plan of action is required. In order to maintain productivity and safety, all proceeding mission re-planning must be optimized within operational constraints as well. Hence, a complete planning and navigation support system capable of adapting to changing situations in real-time is essential to optimizing EVA performance. The scope of such a system extends to any remote excursion with a team of explorers whether on Earth, Moon, Mars, or beyond.

1.2 OBJECTIVE

Planning an optimally productive EVA mission requires the quantitative representation of mission costs, returns, and constraints coupled with the ability to compare the results of various operation strategies and situations. When shifted to real-time during a mission, optimized re-planning further necessitates the capacity to quickly analyze changing situations and seamlessly update the mission plan with the best course of action. The knowledge base upon which these decisions are made must continuously be updated with feedback from the field explorers as the mission is carried out. Finally, productive operation further demands effective support for the surface explorers in navigating along planned paths to objective sites and carrying out the desired activities.

The primary goal of this thesis is to develop a model automated support system for optimizing planetary EVA operations. This system's use will be twofold: both for pre-mission planning and simulation as well as for real-time explorer navigation and re-planning. To this end, surface EVA missions will be characterized and the optimization process examined. All operation factors relevant to mission support will be identified, and automated system performance evaluated. The prototype mission support system will then be presented, followed by field tests involving key system components. The results will be discussed along with several design recommendations for enhancing mission fidelity.

1.3 THESIS OUTLINE

Chapter 2 begins by establishing the general makeup and especially challenges of planetary surface missions through a pragmatic overview of past, present, and projected future EVAs. Mission operations are further classified into a set of specific interactions between the environment, field explorers, and mission control. From here, the methods by which these interactions, and hence mission performance, can be optimized in terms of productivity is clearly specified. In particular, the framework for providing robust real-time support is developed.

Chapter 3 further breaks down EVA operations into a set of specific factors pertinent to automated mission support, and the functionality of a comprehensive automated support system is presented. The aid provided by such a system is necessary to enable optimal performance of controllers and explorers given the high time pressure of making decisions and completing planned objectives on schedule. In particular, the distinct support system functions of optimal mission planning, surface explorer activity support, and real-time re-planning in response to uncertain situations are explained in detail.

In Chapter 4, a subset of the established factors and functionality of automated support are implemented in the development of a prototype mission operation support system, named Pathmaster. All features of this prototype system are presented in detail, outlining the near real-time process by which users may represent a mission situation and develop an optimal plan of explorer traverses to be executed. The generation of predicted explorer costs and the subsequent traverse optimization routine are explained as well.

Chapter 5 summarizes the field testing that has motivated the development of the automated support system into its current state. The setup and operation of each experiment is presented, and the key system components addressed are identified. Results are discussed in terms of the intended and actual performance of the system and emergent desire for improved functionality.

Chapter 6 concludes by summarizing the contributions of the current work and discussing the formation of an ideal mission support system. To close, numerous design recommendations to improve the fidelity and utility of the Pathmaster system are provided along with opportunities for continued research in EVA operational support.

2 CHARACTERIZING PLANETARY SURFACE MISSIONS

2.1 EXTRAVEHICULAR ACTIVITIES

On July 21, 1969, man first landed on the Moon. While Neil Armstrong and Buzz Aldrin spent 21 hours on the lunar surface that day, they are eternally remembered for the two and a half hours that they ventured outside the Lunar Module carrying out the first extravehicular activity (EVA) on the surface of another world (NASA, 2004; BBC, 2008). Every subsequent manned mission to land on the moon has included a set of EVAs as well, with ever broadening objectives and ambitions. In order to design a planning and support system for these sorties, the first step is to characterize the makeup, expectations, operation, and challenges of such missions.

2.1.1 PAST AND PRESENT SURFACE EVAS

Our experience with planetary EVA operations begins with the Apollo program, where each mission was completed by a team of two suited astronauts. Sticking together, the team would travel to pre-planned sites and complete various activities such as drilling and collecting samples. Although extensive preparation involving scientists, engineers, astronauts, and mission planners would be undertaken to maximize scientific return of each mission, the resulting traverse routes and estimated travel times were established based upon low-resolution photographic images and crude topographic maps (Muehlberger, 1981). The Apollo 11 and 12 EVAs focused mostly on engineering testing, and it wasn't until Apollo 14 that mission objectives shifted more toward surface exploration and scientific advancement (Márquez, 2007). In these missions, where traverses grew along with the demand on crews, several common problems emerged.

On the second EVA of Apollo 14, the astronauts were provided with only a paper map as a guide to locate the edge of Cone Crater (NASA, 1971). Wearing bulky space suits on the unfamiliar vast monochromatic terrain, the astronauts became unsure of their position and began climbing unexpectedly steep slopes which were obscured by low sun angles (Figure 2.1). The astronauts began pushing the limits of exertion with elevated breathing and heart rates, forcing them to periodically stop and rest. Falling behind schedule, fatigued, and unable to accurately determine

their objective, the planned destination was abandoned and the crew had to settle for another site. As a result, the crew recommended incorporating a 30% safety margin in future mission scheduling (Engle, 2004).

Figure 2.1 Low sun angles on the flank of cone crater (NASA image, AS14-64-9099)

Apollo 15 – 17 saw the addition of a Lunar Roving Vehicle (LRV), and with it an inertial "dead-reckoning" navigation system (Figure 2.2). This onboard system provided the range and bearing back to the Lunar Module, at least to within 600 meters (LaPiana, 1971; Wade, 2008). Despite enabling astronauts to travel considerably farther with much less effort, neither the rover nor the navigation system provided astronaut assistance in distinguishing objective sites or judging

terrain slopes and feature sizes. Crew members reported consciously driving slower as a result of not being able to accurately anticipate the upcoming terrain (Jones, 1995). The introduction of the LRV also imposed a strict safety constraint known as the "walk-back" requirement on all EVAs. By this rule, the astronauts were never allowed to venture a distance farther from the Lunar Module than they would be capable of walking back with the remaining oxygen. This way, should the LRV fail, the astronauts could still make it back safely on foot (Jones, 2006). As a result of this constraint, exploration to farther sites came with a high time pressure so as not to exhaust undue oxygen and violate the "walk-back" requirement. Any delays or unexpected findings jeopardized the completion of all planned objectives. Such a situation occurred during Apollo 17, when astronauts found "orange soil" and had to quickly assess whether they could collect unplanned core samples given the resource limitations (Jones, 1995; Márquez, 2007).

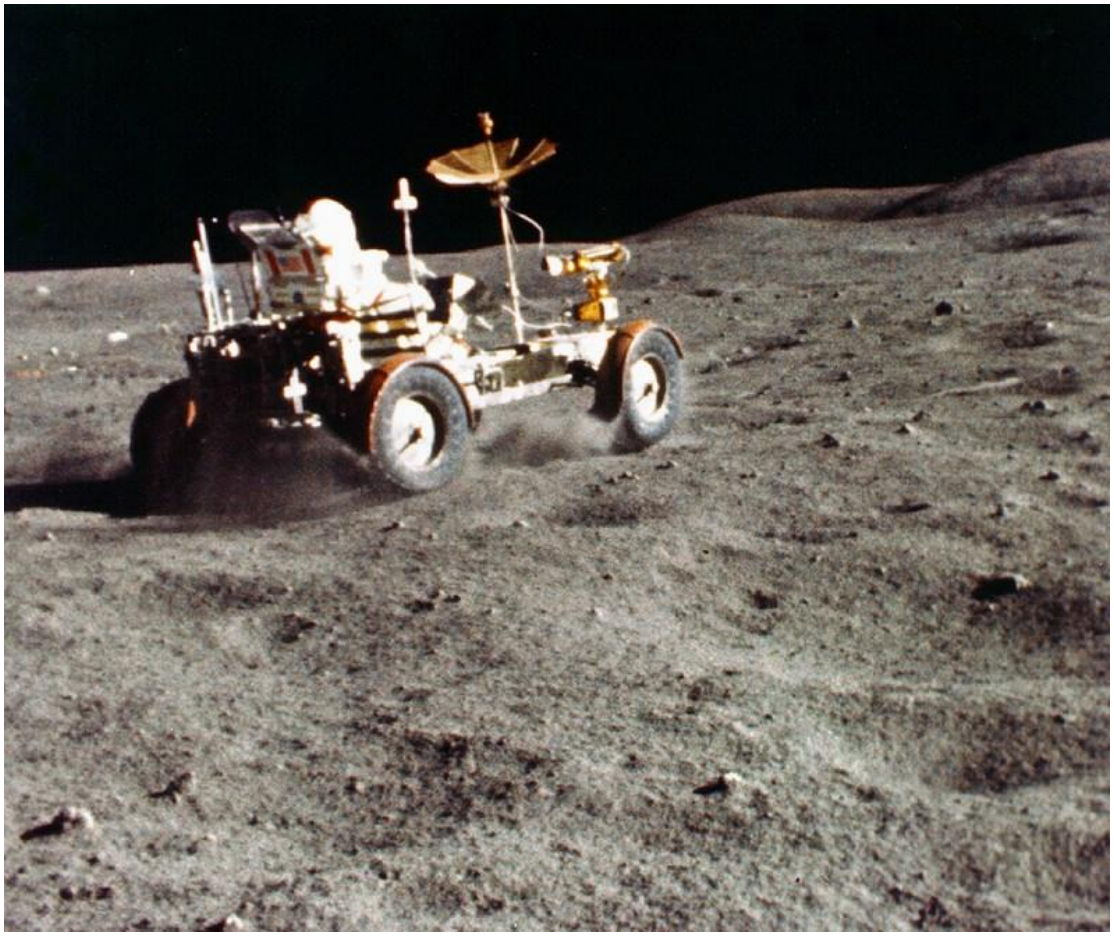


Figure 2.2 Apollo 16 astronaut driving the Lunar Roving Vehicle (NASA image, S72-37002)

In addition to issues with navigation and unplanned findings, technical difficulties also caused crews to fall behind schedule. The lunar surface dust in particular became a considerable problem. This very fine, unweathered dust "can adhere to every object and penetrate very small openings…the dust permeated the cabin, covered the EVA suits, and soiled the field experiment hardware" (Figure 2.3) (Lindqvist, 2008). The dust impeded the performance of instruments and forced crews to take extra time to clean equipment. Each crew also had to deal with occasionally malfunctioning equipment, including the LRV. A recurring theme of EVAs on the moon was a general lack of time to complete all planned activities (Márquez, 2007). All lunar missions sustained considerable time delays or contingencies in one form or another. Faced with resource limitations, the explorers and mission control were required to perform real-time re-planning of each lunar EVA to salvage the objectives of highest priority and return the crew home safely.

Figure 2.3 Apollo 17 astronaut covered with lunar dust (NASA image, AS17-145-22157)

When the Apollo 14 crew landed in 1971, another somewhat less known explorer from earth was also roaming the moon. Launched by the Soviet Union, Lunokhod 1 was the first remote controlled robot to land on another world. Along with its successor Lunokhod 2, these 8 wheeled vehicles slowly rolled alone across the lunar terrain performing soil analyses and capturing thousands of images (Christy, 2008). Such EVAs were an early analog to our present exploration of Mars (Figure 2.4). The Mars Exploration Rovers (MER), Spirit and Opportunity, are currently surveying Mars on a daily basis carrying out various scientific goals. Unlike the Apollo missions,

these long-term unmanned EVAs did not originate with a set of pre-selected objective locations. Instead, all investigation sites have been determined in-situ by planetary scientists based upon imagery and spectroscopy data taken by the rover (Márquez, 2007).

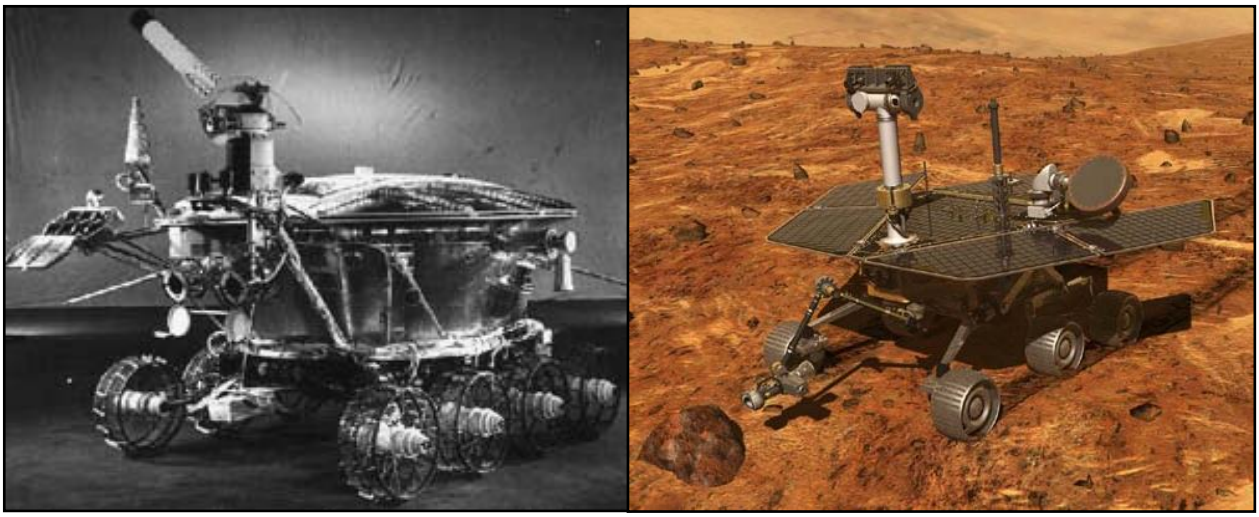


Figure 2.4 Past and present exploration robots. Left: Lunokhod 1 (Christy, 2008) Right: Artist's rendering of a Mars Exploration Rover on the Martian surface (JPL image)

The real-time planning of traverses and tasks for the MER takes place every Martian day (sol) by a team of scientists and engineers. This group interfaces with the rovers using the Scientific Activity Planner (SAP) (Norris, et al., 2005). The SAP processes data received from the rover and produces terrain maps detailing slopes, solar energy, and instrument reachability (Leger, Deen, & Bonitz, 2005). The limiting aspect of the rovers' exploration capacity is their navigation ability. The topography and soil mechanics of the Martian terrain vary considerably and have a large impact on rover traverse speed, slippage, and power requirements (Iagnemma et al., 2004; Márquez, 2007). Once waypoints and end goal states are chosen, a mission simulation is run that predicts power consumed, time required, data volume, and final position (Norris, et al., 2005). The physical rover then assesses the mission plan and determines the actual traverse path itself, avoiding any obstacles (Márquez, 2007).

Despite this highly sophisticated and meticulous planning routine, the MER missions have revealed EVA problems and concerns in addition to those encountered during the Apollo era. While the Apollo astronauts may have fallen behind schedule and experienced difficulties with navigation, they never had to deal with becoming altogether immobilized like on Mars when a

rover got stuck in a sand dune (Biesiadecki, Leger, & Maimone, 2005). This contingency highlights the great sensitivity that robots particularly have to traverse path planning, which is ultimately based upon predictive modeling of the upcoming terrain characteristics. It also exposes the limits of a robot in recovering from difficulties during an EVA. A human explorer has the inherent ability to cope with uncertainty and make real-time judgments in response to the unexpected, whereas a robot becomes dependent upon operator intervention when things do not go as planned. The Apollo and MER programs have provided a fair, though limited, understanding of the requirements and challenges incorporated in planetary surface EVA operations. We must leverage this experience as we look toward the future of EVA exploration.

2.1.2 FUTURE VISION OF EVAS

When we return to the moon, mission operations will be far more complex and demanding than experienced before. EVA traverses will likely be conducted in a fashion similar to the MER program, where sites of interest are chosen in real-time based upon constantly updating terrain data and imagery. However, unlike MER or Apollo, these missions will involve a greater number of explorers, both human and robot, working cooperatively. The goal of establishing an extended presence on the moon also requires humans to remain on the lunar surface for much longer durations and endure a significantly higher number of EVA missions than ever before. EVA activities will expand from simple testing and sample collection to daily construction, maintenance, and extended exploration sorties (Figure 2.5).

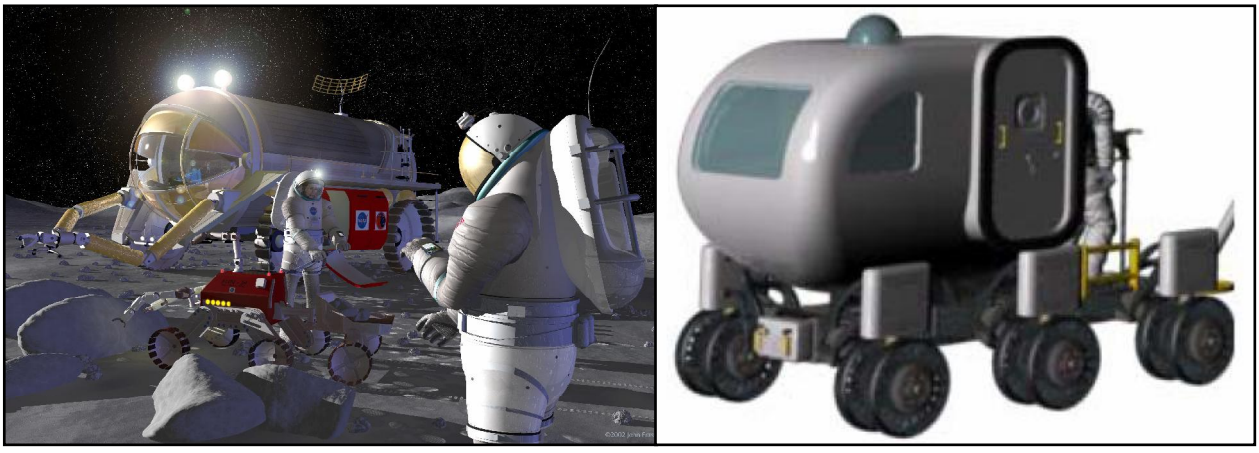


Figure 2.5 Future Lunar EVA systems and operations
Left: Artist's rendering of humans and robots working together on the moon (NASA image)
Right: Model of a prototype pressurized lunar rover (Cooke et al., 2007)

Missions such as these bring a host of additional EVA concerns that must be addressed. First, supporting human life on the surface of another world for an extended period has yet to be accomplished. To ensure crew safety, new restrictions limiting the total astronaut work performed, acceptable radiation exposure, and nighttime traversals will need to be enacted (Márquez, 2007). Second, since science in the vicinity of an outpost can be quickly exhausted, extended range surface mobility becomes essential (Cooke et al., 2007). Such expansive exploration necessitates improved surface navigation. The Apollo 17 crew travelled just over 11 kilometers from the Lunar Module at farthest (Eppler, 2004). In contrast, pressurized rover missions have been proposed with ranges exceeding 900 kilometers from an outpost (Cooke et al., 2007)). Crews must not only be able to traverse the upcoming local terrain robustly and with ease, but also require the capacity to accurately locate objective sites and precisely navigate back to base. Lastly, any robots joining an EVA must augment the capabilities of a team, not burden them. This demands that robots in the field be capable of keeping up with human explorers in terms of physical travel, power life, and data processing. Developing effective systems able to perform these tasks on a daily basis is a formidable obstacle that engineers and mission planners will need to surmount.

2.2 MISSION INTERACTIONS

In an abstract sense, EVA operations can be thought of as a set of interactions between the environment, the field explorers, and mission control. The management of these interactions is paramount in promoting EVA productivity

2.2.1 ENVIRONMENT INTERACTIONS

Environmental factors that determine the execution of an EVA are terrain properties, sun lighting, gravity, atmospheric characteristics, radiation, space suit or robot capabilities, support equipment (rovers, tools, etc.), consumable resource supplies, and safety constraints. Barring contingencies, all of these factors remain constant over the course of a planetary mission except for the remaining resources, sun lighting, and local terrain. For EVA planning to occur in real-time, these variable parameters must be continuously monitored and updated. Measuring remaining consumables is relatively simple, and has been accomplished routinely in past and present EVAs both on the moon and in orbit. In turn, sun illumination may be determined

mathematically given the current time and global position of the surface team. A detailed analysis on the interaction of sun position and EVA performance is provided in Márquez, 2007. We lastly focus on characterizing and interacting with the planetary terrain.

Conducting a surface EVA begins with the ability to successfully traverse the terrain and access sites of interest. This means that explorers must be able to robustly maneuver across varying features and negotiate any obstacles in order to reach objective destinations. Any hiker knows that increased surface slopes correlate to greater exertion and slower speeds. Differing soil types further have a large impact on the ease of explorer mobility within an area (Iagnemma et al., 2004). Unpredicted terrain makeup or topography as well as poor visibility may also significantly affect travel routes and times. Finally, during a traverse explorers may encounter a feature or region of unexpected apparent interest. Travel to or through this territory hence becomes desirable considering the potential scientific gain. On earth, humans manage interacting with various surface properties and shifting immediate objectives quite intuitively on a daily basis. However, wearing a confining space suit and faced with an inhospitable terrain with no familiar references under high time pressure, these tasks become significantly formidable. Instructing a robot to do the same is even more troublesome.

Beyond merely crossing a terrain, explorers must also accurately locate and distinguish objective sites. This means that one must not only be able to dependably navigate areas of traversability and avoid obstacles, but also pinpoint the arrival at a destination. Geological research additionally requires constant monitoring of explorer location in respect to a geographic database (Eppler, 2004). For positioning on earth, we have the Global Positioning System (GPS) as well as a global magnetic field that enables the use of a compass. No such conveniences exist on the moon or Mars (Arnett, 2005; Acuña, 2003). The Apollo experience has shown us that unaided astronauts dealt rather poorly with navigating unfamiliar lunar territory, and MER has shown us that navigating a lone robot across the Martian surface is quite painstaking. Properly executing all of these terrain interactions, though, makes the difference between successfully reaching destinations on schedule and being forced to abandon mission objectives or even getting stuck.

2.2.2 ASTRONAUT-ROBOT INTERACTIONS

Future planetary missions will likely be conducted in teams of multiple astronauts and robots. Synergy in the field will rely upon explorer to explorer interactions. Human to human interaction is intuitive (at least for most) and has been exhaustively studied. Allocating tasks between humans is also relatively routine. During Apollo, there were no significant issues with astronauts working together. Human-robot interaction, on the other hand, is a relatively new and much less well defined field. Taking full advantage of the diverse capabilities of both astronauts and robots begins with allocating mission tasks to the best suited team members. Astronauts hold advantages in ease of mobility, dexterity, reasoning, improvisation, and exercising judgment. Meanwhile, robots have advantages in precision, repetitiveness, computation capacity, quantitative data collection, and multitasking (Márquez, 2007). The use of robots is also relatively cheap, and it eliminates human risk (Squires, 2008).

In past experiences with humans and robots working cooperatively, two general strategies have emerged. The first is to treat the robots as a separate unit, while the second is to use the robots as a technical tool (Casper & Murphy, 2003; Cabrol et al., 1999). Treating the robot as a separate unit generally involves sending the robot alone to complete tasks that are either significantly costly or impossible for a human to execute. A common such assignment involves sending the robot as a scout ahead of an astronaut team to characterize environmental parameters (Cooke et al., 2007). Another example is sending astronauts to quickly traverse an area and flag sites of interest, while a trailing robot visits each flagged site and performs a longer and more tedious analysis (Cabrol et al., 1999). Finally, a smaller robot can be called in to access regions that are either inaccessible or too dangerous for astronauts. This general strategy enables mission planners to employ the fast human understanding of the environment and main mission objectives when most advantageous, while capitalizing on the low cost and low risk of using robots to complete lengthy repetitive tasks in a hostile environment (Cabrol et al., 1999).

In turn, robots have also been used as a technical tool travelling along with a human team. Associated tasks generally include surveillance of the human team, use as a search camera, transporting tools and samples, or use as a computational analysis instrument (Cabrol et al., 1999; Casper & Murphy, 2003). The major advantage here is that the robot replaces a

crewmember in completing mundane tasks, which frees the human to perform other tasks. Also, sophisticated onboard systems could aid astronauts in identifying and mapping terrain features and sites of interest as well as quickly analyzing samples. A considerable drawback to this strategy, though, is the limited mobility of robots on rough terrain. Robots supporting astronauts in the course of doing field work "must be able to go up the hills, over the rocks, everywhere the human goes, at the same speed" (Eppler, 2004). Indeed, "what [MER style] rovers can do in a day, humans could do in a minute" (Squires, 2008). The theme of robots falling behind humans on a joint mission is clearly observed in the field tests presented in Chapter 5 as well. Robot locomotive technology will need to make great strides before robots are ready to keep up with traversing astronauts.

Both of these strategies, while highly useful in certain situations, fall short of fully incorporating robots as team members. Robots have the potential to be far more than a modest tool or instrument. While astronauts are readily able to assist robots in dealing with uncertainty today, future robotic systems could be capable of mitigating human EVA uncertainty and errors. There is great need for advancement both in the field of human-robot interactions and in robotic technology before the vision of astronauts and robots working cohesively on the surface of another world becomes a reality.

2.2.3 MISSION CONTROL INTERACTIONS

All major EVA decisions are made via a team on earth overseeing the entire mission, known as *mission control* (Figure 2.6). In order to make decisions and re-plan activities in real time, mission control must gather all data taken by the surface team, assess the current situation, develop a plan, and relay the new course of action back to the explorers. This must be accomplished quickly and seamlessly, so as not to waste valuable consumables while re-planning. Achieving this necessitates mission control to maintain updating models of the local environment and the estimated explorer costs for completing each task. All resources and constraints must be monitored as well. Essentially, mission control must remotely interact with all aspects of an EVA in order to provide effective support.

Figure 2.6 Mission control for the Phoenix Mars Lander at the Jet Propulsion Laboratory

Mission control interacts with the EVA environment by modeling it within a database employed for making decisions. Most environmental parameters are known beforehand or, as in the case of sun lighting and consumables, may be calculated or measured directly. Terrain characteristics and explorer costs, however, must be represented as modeled estimates. A traversing explorer may find new terrain to be substantially easier or more difficult to negotiate than expected. The soil mechanics may vary drastically, obstacles may emerge, or a new interesting features may become apparent. In these cases, mission control must document the new explorer feedback within the environment models. In addition, explorer costs such as time or metabolic expenditure required to perform certain EVA tasks may also deviate from the predicted values. Again, this feedback must be incorporated into the activity cost models as fit. Once updated models are generated, a new mission plan based upon the latest data may be developed.

Interactions between mission control and the physical explorers involve relaying information and commands. Explorers actively supply mission control with relevant EVA data, and passively transmit consumable resource measures. In turn, mission control conveys the latest commands

and objectives back to the surface. The process of monitoring explorers and revising commanded activities in response to feedback represents a continuous support cycle. Astronauts may intuitively send and understand relayed information audibly or visually. They require little aid in quickly adapting to understood commands. In the case of a robot, though, these interactions are potentially much more involved. A human operator at mission control is responsible for directly assisting a robot whenever situation uncertainty or unexpected scenarios cannot be immediately resolved (Figure 2.7). The controller, in turn, must receive all possible data from the robot in order to accurately assess the situation and make appropriate judgments. The value of this operator-robot interaction has been highlighted in the MER missions (Márquez, 2007). To summarize, when mission uncertainty cannot be mediated by the surface team alone, mission control intervenes and determines the next course of action.

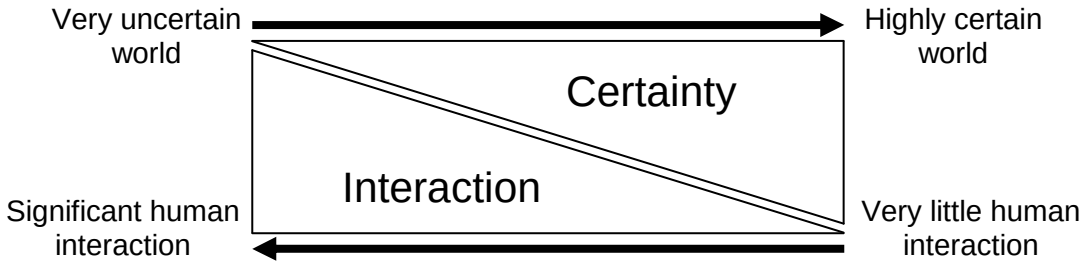


Figure 2.7 Human interaction with automation as a function of certainty (Cummings, 2006)

Finally, mission control is not necessarily a single localized entity. In fact, the majority of current NASA missions are operated by multi-organizational teams which are dispersed across various locations (Clement et al., 2007). Effective collaboration between institutions is dependent upon site to site interaction. Mission control locations must have mutual access to all databases as well as open inter-site audio and visual communication. Interaction with the surface team, however, must not be convoluted, stemming from multiple disconnected sources. Instead, a small contingent should be dedicated to communication with the surface team and operation of robots. All mission updates would be sent via this team.

2.3 OPTIMIZING MISSIONS

Now that we have a clear picture of the general makeup and operation of surface EVAs, we focus on the key aspects which will enable maximization of mission productivity. Achieving

productivity may in an abstract sense be thought of as optimizing each of the mission interactions detailed in the previous section. In another sense, fostering productivity involves mitigating and working through the challenges encountered on an EVA. Either way, the efficacy of mission operation is ultimately determined by the ability to develop an optimal mission plan, the subsequent ability to carry that plan out, and the robust ability to re-plan in the face of uncertainty.

2.3.1 PLANNING: INPUTS AND OUTPUTS

The goal of mission control when planning a mission is to maximize EVA return and minimize costs while remaining within all operational constraints. Numerous factors are incorporated into the planning of an EVA. To begin, the mission environment is modeled. This representation includes the terrain properties as well as sun lighting, gravity, atmospheric makeup, and radiation. Next, the resources available to a mission are identified. Resources include astronauts, robots, and equipment as well as consumable supplies such as oxygen or battery power. Furthermore, each distinct explorer must be modeled in terms of the time and consumables required to perform EVA tasks. Lastly, constraints are applied to the system. Constraints include limits on resource consumption and elapsed time as well as restrictions on terrain areas which may be traversed. Once the EVA environment has been modeled, mission objectives are identified. Objectives are driven by scientific return, and may include both destination sites and specific activities. Collectively, with the environment, resources, constraints, and objectives clearly defined, we achieve a complete representation of the EVA situation.

Once all situational inputs are entered, an optimization is applied to determine the best course of action. This optimization ensures that the greatest possible extent of mission objectives are met while incurring the least possible cost. Essentially, this routine optimizes the travel and activity plans in terms of favorable explorer-environment interactions, as well as the overall mission strategy in terms of the most advantageous astronaut-robot interaction schemes. The output of such mission planning is a well defined set of EVA destination sites, traverse routes, desired activities, and time and cost schedules. This plan is relayed to the surface team for immediate execution. The overall planetary EVA planning framework is summarized in Figure 2.8.

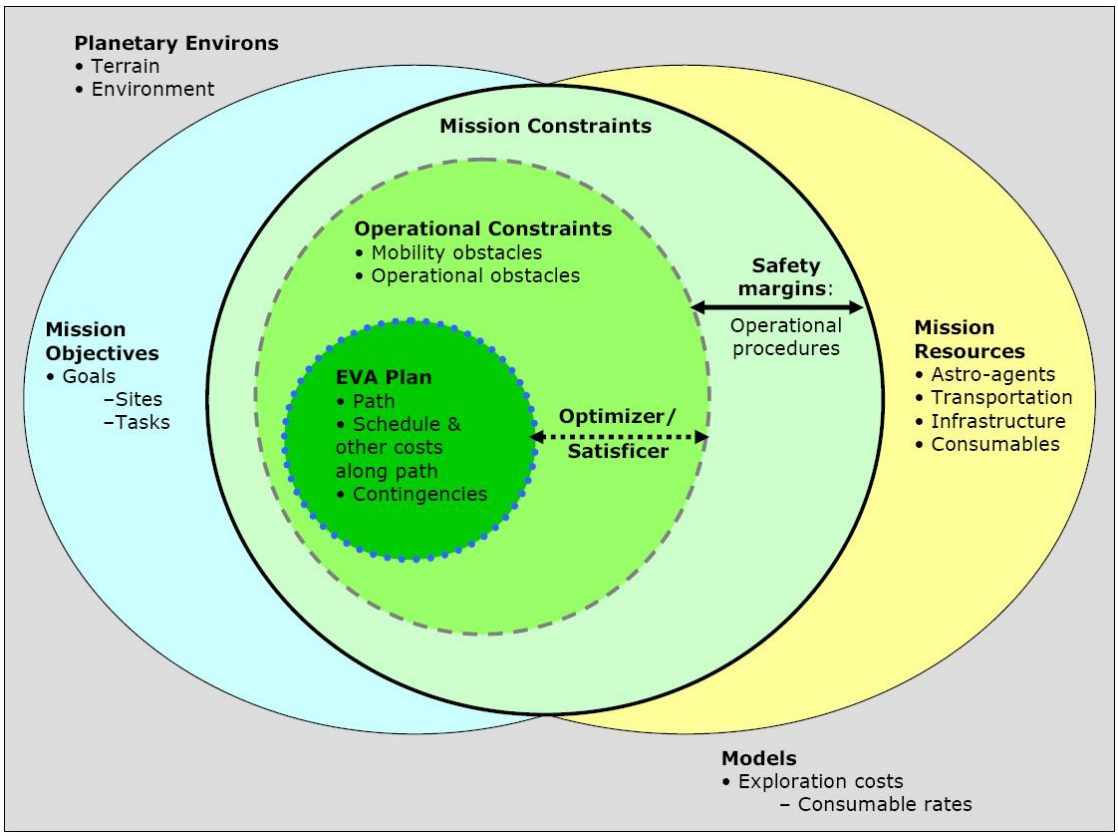


Figure 2.8 Planetary EVA planning framework (Márquez, 2007)

Viewed as a functional block diagram, the task of EVA mission planning is shown in Figure 2.9.

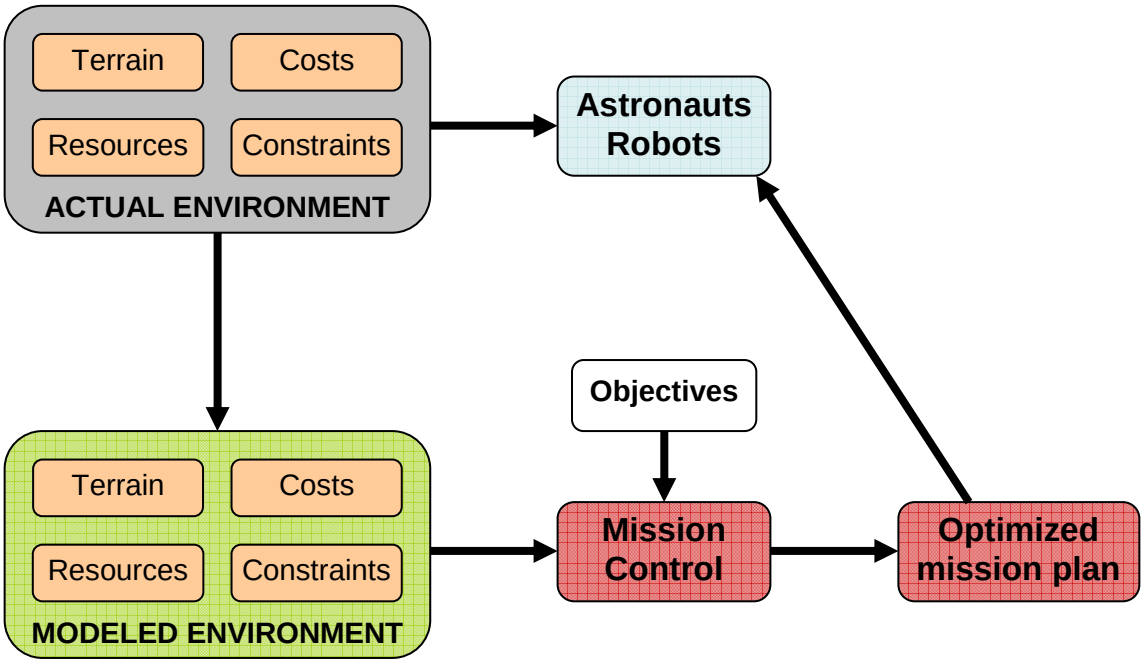


Figure 2.9 Block diagram of planetary EVA mission planning

2.3.2 MAXIMIZING PRODUCTIVITY

Maximizing EVA productivity begins with generating an optimal mission plan based upon the best available data. However, providing an ideal plan is irrelevant if the physical explorers are unable to accurately follow it. The primary operational challenges identified in the Apollo and MER experiences include difficulties in terrain navigation and falling significantly behind schedule. The associated missions had been carefully planned, yet in many cases the explorers were unable to complete desired tasks or keep up with the desired timing. Due to uncertainty, error is induced into the mission planning framework. This error results in discrepancies between the mission plan and the actual EVA performance (Figure 2.10). These discrepancies potentially correlate to substantial degradation in operation productivity and even the abandonment of mission objectives.

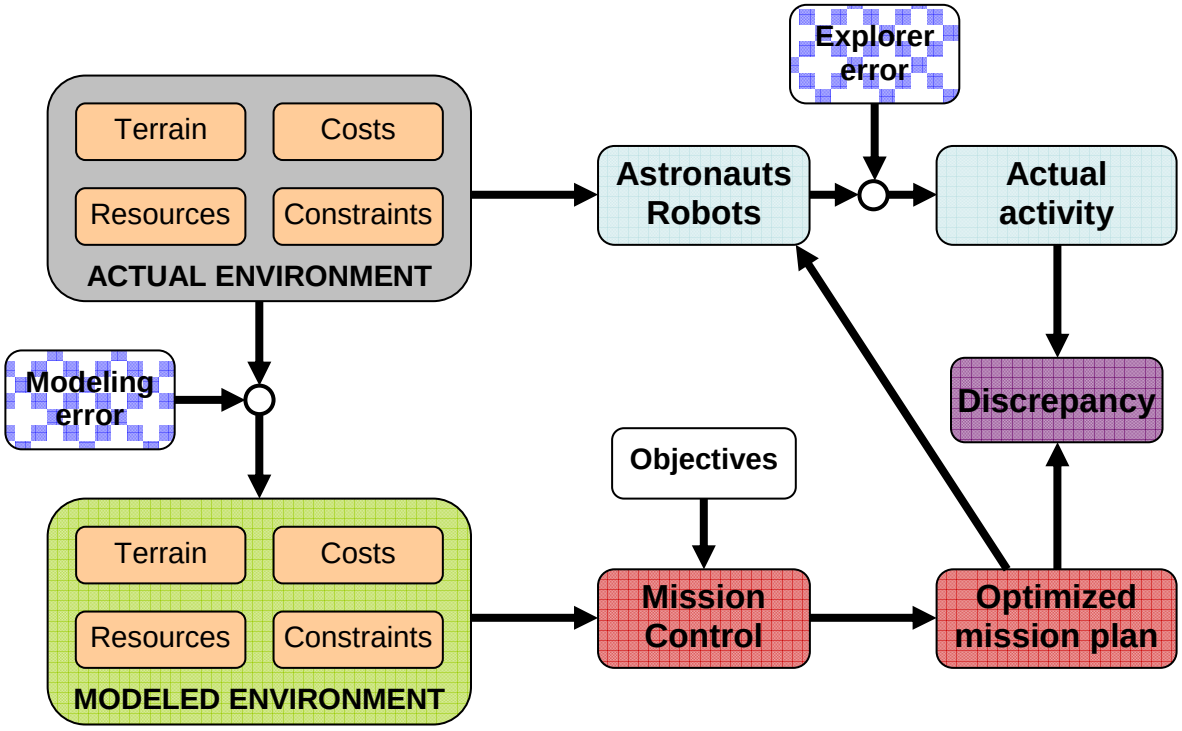


Figure 2.10 Block diagram of EVA mission planning, error, and actual activity

Assuming that the plan developed by mission control is in fact optimal, maximizing EVA productivity becomes a matter of eliminating the errors which result in operational discrepancies. There are two primary sources of error: modeling error and explorer error. Modeling error primarily results from uncertainties in terrain property and explorer cost estimates. Apollo astronauts struggled to assess slopes and terrain features, while mission planners significantly

underestimated the time and exertion levels required to complete certain tasks on the moon. In turn, MER engineers still struggle to predict varying Martian soil mechanics and the associated energy required for robot motion (Perko, Nelson, & Green, 2006). Constantly updating a model database with the most recent data, ideally in real-time, is perhaps the most effective method for mitigating modeling error. The best models are not based upon a priori estimates, but upon actual experience.

Explorer error is primarily a product of navigational difficulties. In cases where modeling error is not to blame, explorer error results from general disorientation, deviation from a planned route, inability to locate or recognize objective sites, lack of visibility, or failure to distinguish samples of interest. Dependable navigation support is a critical aspect in future planetary missions, not only to promote productivity by reducing explorer error, but also to ensure explorer safety in returning to shelter. Navigation support in the form of a display detailing the terrain, traverse routes, objectives, and current position would greatly enhance a human explorer's interaction with the unfamiliar environment. For robots, offering mission data along with closed-loop position feedback would alleviate situation uncertainty. Current research in the automotive industry with "smart windshields" is a simplified analog to heads-up navigational assistance which would be highly beneficial on the moon and Mars (Figure 2.11) (GM, 2008). Equipped with an optimized mission plan based upon reliable models and coupled with accurate navigation support, future astronauts stand to be far more productive than their Apollo counterparts.

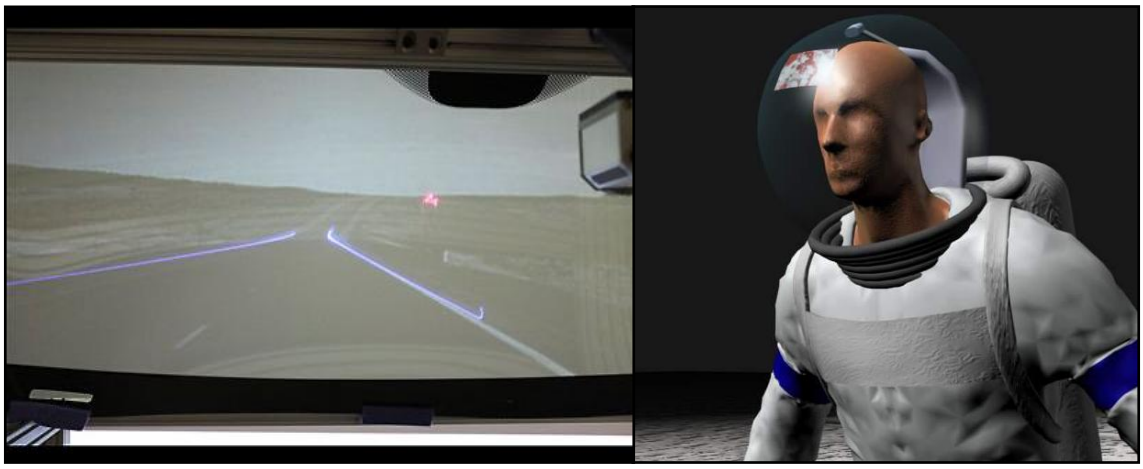


Figure 2.11 Heads-up navigation assistance concepts
Left: General Motors "smart windshield" to enhance the upcoming view (GM, 2008)
Right: Mission support system integrated with a space suit helmet (Lindqvist, 2008)

2.3.3 CONTINGENCIES AND RE-PLANNING

Error in modeling and navigation is not the only factor that can cause significant deviations between a mission plan and the actual situation. Contingencies are prevalent in all past surface EVA experiences, and they are typically even more disruptive to a mission than general planning uncertainty. Developments such as equipment failure, explorer health concerns, unexpected discoveries, or environmental emergencies can drastically alter the available mission resources, immediate objectives, and activity constraints. Inherent in exploration is the unexpected. In order to maintain productivity, EVA operations must be robustly adept in handling multifarious unpredicted scenarios.

By incorporating any errors or contingencies into the mission model database, an accurate representation of the current situation is attained. From here, a new mission optimization can be performed to generate a revised plan which outlines the best operational response for the remainder of the mission. This closes the loop of planning, activity performance, coping with uncertainty, and re-planning with updated information (Figure 2.12). This process repeats itself whenever the surface team deviates from the established plan, either by choice, mistake, intervention from mission control, or contingency. Maintaining this cycle is essential to ensuring productivity in the face of uncertainty. By updating the mission model database and re-planning accordingly, the surface team may at all times act according to an optimal plan based upon the best available data. Coupled with perhaps a heads-up display, explorers could seamlessly receive new mission information and begin implementing it in the blink of an eye. Such functionality, though, imposes a high pressure on mission control to be capable of updating models and robustly re-planning in real-time. Mission planners will require high-fidelity, automated support to meet the optimal operational demands of these future EVAs.

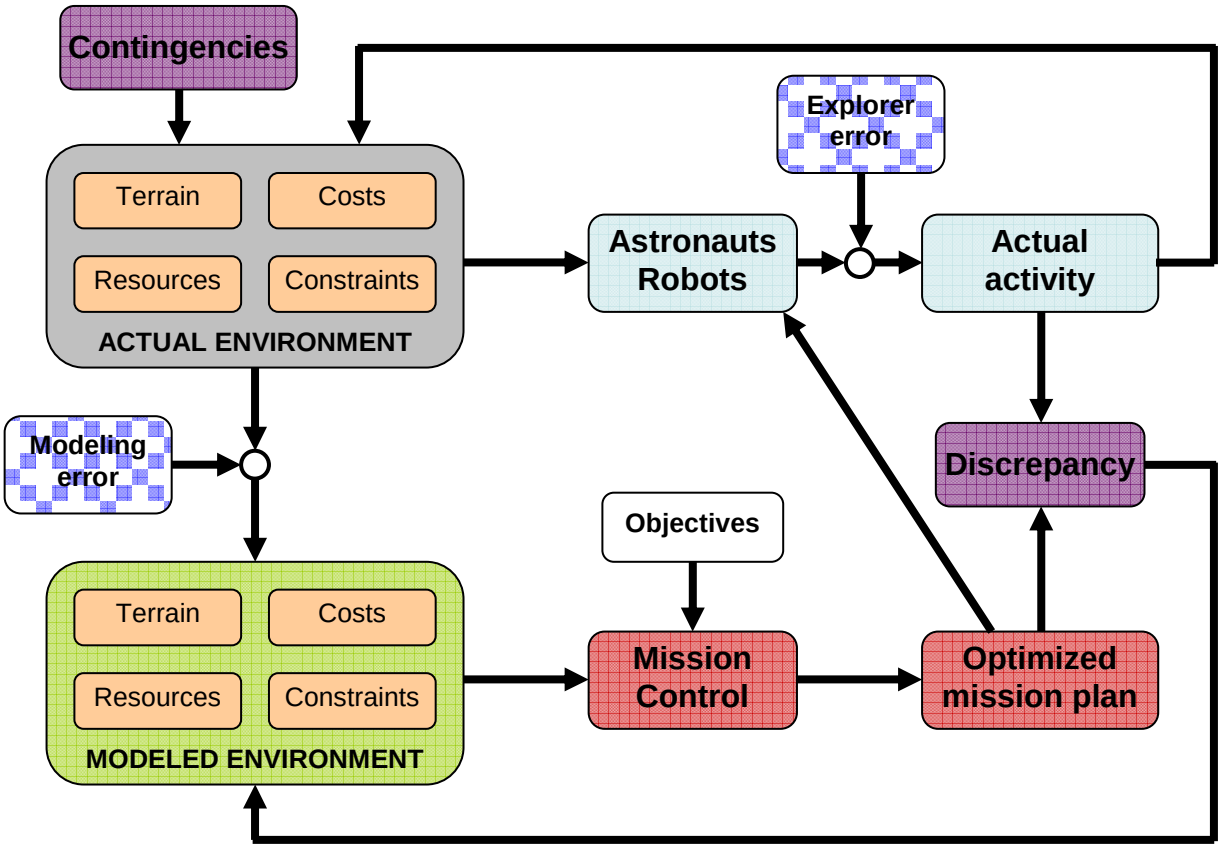


Figure 2.12 Block diagram of complete EVA planning, activity, and re-planning cycle

3 MISSION OPERATION FACTORS FOR AN AUTOMATED SUPPORT SYSTEM

3.1 MISSION PLANNING

The goal in developing an EVA plan is to maximize mission return while minimizing explorer costs within all constraints. To accomplish this, mission planners need organized methods for analyzing all factors of EVA performance, comparing various potential scenarios, and making optimal decisions. Considering the likely future EVA architecture where sites of interest are determined in real-time, there arises a high pressure to develop new plans quickly. This means that all aspects of the decision making process need to function seamlessly, from updating planning inputs with the latest feedback to clearly conveying the new mission information for immediate understanding and execution. A support system must be developed that relieves any computational burdens and enables mission planners to rapidly evaluate EVA situations and determine optimal courses of action. This way, human controllers can focus on promptly making high level decisions and relegate the tedious details to the support system.

The planning support system should perform automatically with minimal need for human mediation. Specific levels of automation (LOA) for such a system, listed from 1 to 10, are defined in Table 3.1. Due to the high-risk nature of planetary exploration, no mission plan should be executed without ultimate mission control approval. This limits the support system's permissible LOA range from 2 to 5. Automated systems depend upon quantitative representations of inputs and outputs in order to function. Hence, all relevant EVA mission operation factors including the explorers, environment, objectives, and constraints must be expressed as quantitative models to provide the support system with situation awareness. Automated assessment of potential operation scenarios further requires a framework for representing the relative cost and return of each task. In this way, given a mission situation, various activity scenarios can be directly compared and an optimal plan identified.

Table 3.1 Levels of automation (Parasuraman et al., 2000)

Automation Level	Automation description: The computer…
1	offers no assistance: human must take all decisions and actions
2	offers a complete set of decision/action alternatives
3	narrows the selection down to a few
4	suggests one alternative
5	executes the suggestion if the human approves
6	allows the human a restricted time to veto before automatic execution
7	executes automatically, then necessarily informs the human
8	informs the human only if asked
9	informs the human only if it, the computer, decides to
10	decides everything and acts autonomously, ignoring the human

3.1.1 DEFINING OBJECTIVES

The primary goal of surface EVAs, aside from any construction or maintenance activities, is scientific return. Scientific return is a broad term denoting all interesting data or samples gathered as a result of general activities including engineering trials, environmental analysis, and surface exploration. These operations feed the overarching space exploration goals of uniting and expanding human civilization (NASA, 2007). Specific mission objectives typically include performing desired activities at a site of interest, and in turn venturing to successive destination sites. On-site objectives, such as collecting samples or performing analyses, may simply be verbally or textually dictated to astronauts and even pre-programmed for robots. Performing these localized activities requires minimal support from mission control. In terms of a mission plan, it suffices to simply list these objectives along with appropriate time and cost scheduling. Objectives that involve voyaging over unfamiliar terrain to sites of interest, however, have proven to be more involved. Defining these destination sites, or *waypoints*, entails distinguishing their location with respect to a known position, or better yet, with respect to an established positioning system.

Mission waypoints may be conveniently represented in terms of global position coordinates via latitude and longitude or via a Cartesian projection comparable to Earth's Universal Transverse Mercator system (Riesterer, 2008). Employing such coordinate systems enables precise, quantitative definition of locational objectives in a manner that is universally understood (Figure 3.1).

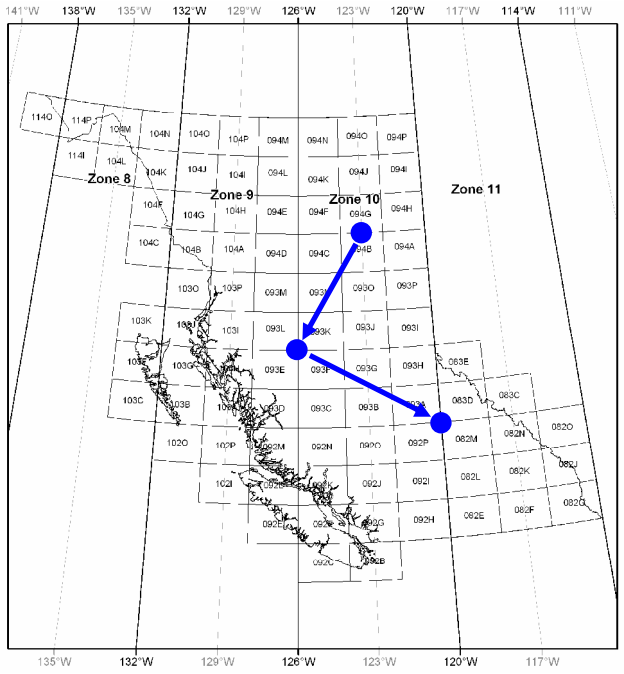


Figure 3.1 Defining waypoints with respect to global coordinate systems (modified from UNBC, 2008)

3.1.2 EXPLORATION COSTS AND CONSTRAINTS

All activities within an EVA incur a measurable cost taken against limited resources. Activity costs are different for each explorer, and are highly subject to local environmental parameters. Predicted cost values for each potential activity are the paramount factor in scheduling objectives given operational constraints. Hence, accurate cost models greatly facilitate the establishment of realistic mission expectations.

3.1.2.1 DEFINING COST FACTORS

The total physical cost of performing an exploration activity may be simplified into three fundamental factors: distance, time, and energy. Distance, determined geometrically, refers to the physical length travelled by an explorer during an activity. Time refers to the elapsed time required to complete an activity. Finally, energy refers to the net energy expended by an explorer in completing an activity. By associating explorer activities with distinct values for each of these factors, a cost profile for each task is expressed numerically. In this manner, assessing the relative total cost of various activities becomes a simple matter of comparing the associated numeric cost profiles. Explorer activity models must also incorporate any applicable constraints.

Operational constraints exist due to limited resources, and these restrictions generally include a safety margin. Constraints limit the set of permissible activity scenarios, and they may be expressed as a maximum bound imposed on each cost factor. In this way, explorer capabilities can be fully characterized in terms of comprehensive cost profiles and factor limits.

3.1.2.2 EXPLORER MODELING

Planetary EVA surface teams are comprised of three general types of explorers: suited astronauts on foot, unmanned robots, and transportation rovers (Figure 3.2). Although these explorers represent drastically varying systems and operation, they may be uniformly modeled in terms of the same cost factor framework. This involves identifying specific parameters which characterize and restrict explorer activity, and then representing those in terms of cost profiles and limits.

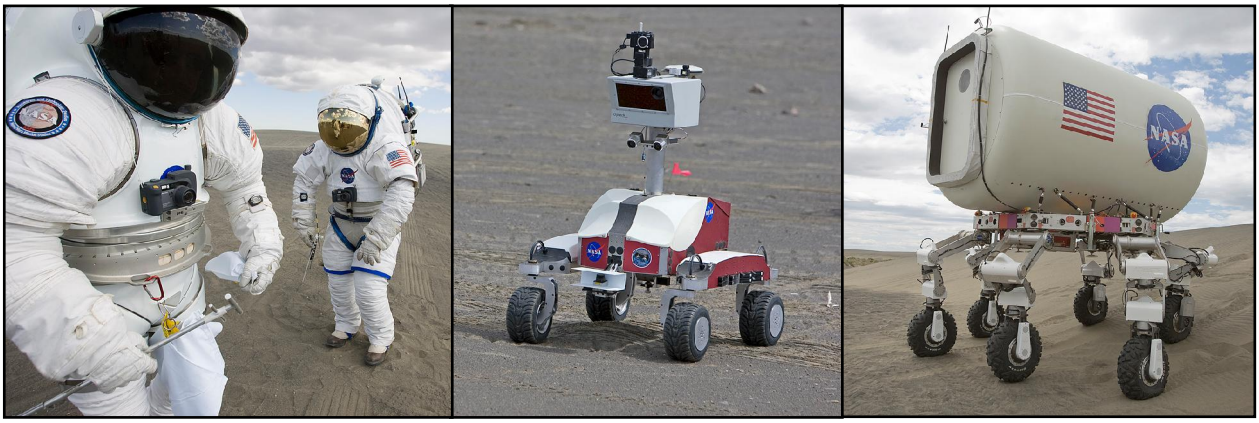


Figure 3.2 Planetary EVA explorer types. From left to right: suited astronauts on foot, unmanned robot, pressurized transport rover (NASA images)

Astronauts on an EVA rely entirely upon their space suit for life support. Modern space suits are comprised of two assemblies: pressurized garments referred to as the space suit assembly, and the life support system commonly recognized as a mounted "back-pack" (Lindqvist, 2008). The suit provides a miniaturized earth-like environment for the occupant, enabling several consecutive hours of activity. However, it significantly restricts the natural mobility and dexterity of the crewmember. Simple activities such as walking or bending limbs, which require little effort on earth, demand a substantially greater exertion when in a pressurized suit. As a result, traverse range and performance capabilities are notably bounded due to fatigue. Stated more precisely, the total work output, or energy, that can be demanded from an astronaut on an EVA is limited by fatigue. In addition to the total energy expenditure, momentary human

exertion is limited as well by health and safety concerns. Astronaut activities are further constrained by the total time permitted on the surface due to finite oxygen supplies and, although the space suit does provide some protection, radiation exposure. Lastly, since astronauts must always return to shelter at the end of an EVA, the allowed distance away from base (outpost, pressurized rover, etc.) is at all times restricted by the remaining resources and cost allowances. Collectively, astronaut exploration cost parameters of concern include momentary and total energy expenditure, oxygen supply, radiation exposure, and distance from base. In terms of our cost factors, the energy and distance constraints may be translated directly. Remaining oxygen and radiation exposure, in turn, set constraints on the permissible time remaining for surface activity before returning to shelter.

Predicting astronaut activity cost values involves characterizing the specific demands of each activity. EVA operations may be broken down into two classes: localized on-site activities, and traversals. Although traversals, which involve travelling considerable lengths across the surface, are highly subject to numerous terrain and environmental attributes, developing a general framework for determining each cost factor is relatively straightforward. To begin, distance is trivially understood as the length of travel along the surface. Next, given a set of terrain properties and knowing the capabilities of a suited astronaut, both the instantaneous velocity and exertion of the moving astronaut can be estimated. Required time to traverse a distinct segment is subsequently found as the quotient of distance over velocity. Finally, total energy expenditure is evaluated as the modeled exertion integrated over required time. Upon completing a traverse and arriving at a destination, on-site activities ensue. These localized, repetitive tasks, such as sample collection or even construction, present a much smaller degree of modeling complexity. For example, crossing a plain will have a drastically different cost than scaling a mountain, but lifting a sample along either route requires essentially the same effort. On-site activity costs may be determined from testing or previous field experience, and can be modeled directly in terms of required time and energy (distance cost is effectively zero). These activity factors should remain relatively consistent from site to site over the course of a mission.

Robots on an EVA may function autonomously or be remotely controlled by mission control or by humans on the surface. The primary factor limiting robot activity is simply stored power,

which in terms of our costs is the total energy available. Ensuring that a robot does not deplete its electrical reserves involves detailed power budgeting of all onboard science, maintenance, and mobility systems (Bagherzadeh et al., 2001). Because robots are not bound by a life-support system, total time spent on the planetary surface is less an issue. However, under pressured situations, robot time delays indirectly amount to substantial costs if they cause mission schedule delays, especially when working along with astronauts. Hence, although surface activity time might not impose a direct constraint on robots, it potentially represents a significant cost within an overall EVA mission.

Robot activity cost values may be formulated in a similar fashion as done for astronauts. On-site activities again incorporate repetitive, consistent costs that can be straightforwardly modeled in terms of required time and energy. In turn, estimating traverse costs requires a more complex model. General robot traverse capabilities are determined by the robot size and equipped locomotion system (wheel configuration, degrees of freedom, motor power, etc.). Actual robot mobility, though, is highly sensitive to the roughness and soil characteristics of the local terrain. Furthermore, terrain uncertainty can be significantly detrimental to robot performance. While an astronaut may be able to maintain standard velocities along poorly mapped terrain with intuitive on-the-fly judgment, robot traverse planning over unfamiliar terrain is a heavily time consuming process by current methods (Biesiadecki et al., 2005). Hence, robot traverse cost models are highly contingent upon accurate terrain models. That said, the general framework for estimating traverse costs is largely the same. Given a specific robot and a set of terrain properties, the traverse velocity and power requirements may be calculated. Coupled with surface distance, the total required time and energy are found in the same manner as with an astronaut traverse.

Transportation rovers, such as the Apollo LRV or pressurized vehicles on future EVAs, offer a highly advantageous tradeoff between astronaut exploration costs and electrical power consumption (or fuel consumption in certain cases on earth). Rovers benefit from the situation awareness offered by a human operator coupled with high velocity and power capabilities. In turn, while any onboard astronauts still incur life-support related costs, their energy expenditure is at a minimum and radiation exposure can be nearly eliminated with onboard shielding. Rovers may also be used to transport robots, which can be powered off while travelling. Rover activity is

primarily limited by stored power and onboard oxygen supplies, which respectively represent energy and time constraints. The overall cost of utilizing a rover includes the distance, time, and electrical energy required for transportation over the terrain, which may be modeled similar to a robot traverse, plus the base support costs for any onboard explorers.

There are certain additional parameters affecting all explorers which must be incorporated into the formulation of cost estimates. The first of these is explorer mass. Stated succinctly, a heavier explorer requires more energy to move. The mass of an astronaut or robot may vary with differing outfitted equipment, while the total mass of a rover is dependent upon onboard explorers or collected samples. Aside from energy differences, changes in mass may also impact traverse velocities. The second mutual cost parameter, planetary gravity, has a profound impact on explorer activity. With respect to earth, gravity on the surface of the moon is approximately one-sixth, while on Mars it is approximately one-third. These substantial differences play a major role in determining explorer power expenditure and traverse velocities. For astronauts in low gravity, loping at a relatively higher velocity is actually advantageous over walking in terms of energy expenditure, contrary to what is experienced on earth (Rader, Newman, & Carr, 2007). Meanwhile robots and rovers, while steady on earth, may be subject to instability and increased slippage in lower gravity.

The final parameter presented which mutually affects exploration costs is the sun lighting. For humans, walking into the sun produces unwanted glare, and low lighting angles obscure the perception of terrain features (Márquez, 2007). This can adversely affect astronaut traverse velocities due to increased terrain uncertainty. Robots and rovers, on the other hand, can greatly benefit from maximized sun exposure via mounted solar cells that replenish electrical supplies. A robot working in direct sunlight can potentially gain net electrical energy, or effectively incur a negative energy cost. Modeling sun illumination can be considerably complex since, unlike explorer mass and gravity, surface lighting varies with respect to time, planetary position, and even local terrain slopes and hill shade. For local short-term activities on the moon, sun lighting remains relatively constant as the moon only revolves once per month. This convenience is not shared on the earth or Mars, which have comparable day lengths. Nevertheless, incorporating sun lighting into the explorer activity costs is essential in developing high-fidelity models.

3.1.2.3 TERRAIN CHARACTERIZATION

Explorer models provide a computational framework for estimating the specific costs of a traversal. In turn, predicted cost values are ultimately determined by the makeup of the terrain being crossed. Incorporating terrain characteristics into the modeling of a traverse involves describing the terrain in terms of distinct representative parameters that can be fed as inputs into the formulation of explorer costs.

Terrain modeling begins with portraying the general surface topography. This is typically accomplished by collecting elevation measurements throughout a region. Such a mapping is known as a digital elevation model (DEM) (Figure 3.3). A DEM commonly projects the terrain surface over a uniform two-dimensional grid, and the elevation at each grid point is recorded. The horizontal spacing between adjacent grid points is known as the map resolution. As this distance becomes shorter, the resolution is said to increase, and data points become more densely packed. At low resolutions, terrain details of a scale shorter than the grid spacing will not be depicted in the model. Instead, these features will be smoothed out, and only the mean local elevation will be expressed. Hence, a higher resolution enables a more precise representation of the terrain and its finer details. High resolution DEM models, though, can correspond to burdensomely large data sets.

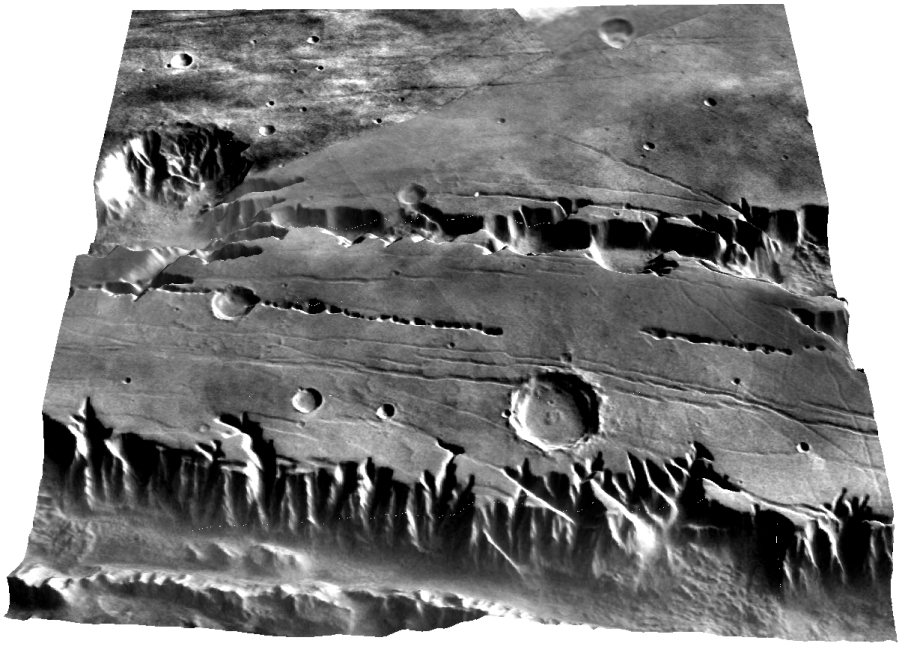


Figure 3.3 Rendering of a digital elevation model (DEM) of Martian terrain (USGS image)

A routine gradient operation may be performed upon the elevation data to furnish the approximate local slope at all points in the model. Surface slopes are a critical first parameter in determining the cost of traversing a given region. Mimicking the modeling of elevations, slope data may be stored as an identically sized matrix in the same orientation. Hence, for each elevation data point in the model, there is now a corresponding terrain slope value.

The next step is to distinguish regions of the terrain that are both traversable and those that are non-traversable. Areas which explorers are unable to cross are defined as obstacles. Obstacles are typically large boulders or locations of increased slope such as steep hills, crater walls, ravines, cliffs, or exceptionally rough patches. Explorers attempting to cross such areas would be dangerously prone to sliding, falling over, or getting stuck. Hence, explorers are required to navigate around obstacle regions. Stated precisely, obstacles represent constraints on the permissible position and planned trajectory of an explorer within the terrain. Obstacle data may be represented in logical terms: true if an area is an obstacle, otherwise false if the area is traversable.

Accessible terrain regions further span a broad spectrum of surface characteristics. The properties that distinguish differing terrain types are in this analysis collectively referred to as *soil mechanics*. Such parameters relevant to EVA operations include rockiness and rock distribution, firmness, strength, stability, and homogeneity (Perko, Nelson, & Green, 2006). Each of these has a specific impact on explorer stability, traction, and slippage. Considered as a whole, these parameters define the overall ease of traversability of a terrain, from which predicted traverse velocities and power requirements may be calculated (Iagnemma et al., 2004).

Following the established modeling scheme, soil mechanics, obstacles, and any other data may be stored as corresponding matrices along with the elevation and slope data. The concept of "layering" various collective data within a terrain model is illustrated in Figure 3.4.

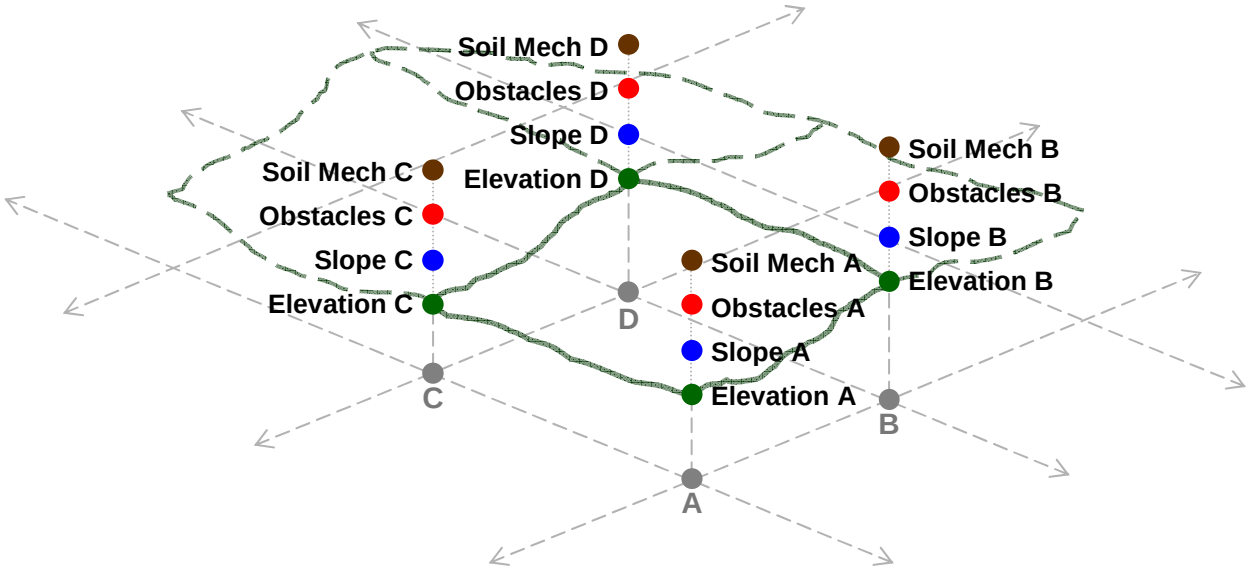


Figure 3.4 "Layering" terrain data at each digital elevation model (DEM) grid point

Taking advantage of global positioning, a DEM and accompanying terrain data may be oriented within the existing coordinate system already employed for locating objective destinations. In this way, each data point along the grid of the terrain model is matched with its physical location, given in terms of global positioning coordinates. Hence, distances, headings, and locations in the terrain model now correspond to real-world values. Moreover, mission waypoints may now be precisely identified and rendered within the terrain model (Figure 3.5).

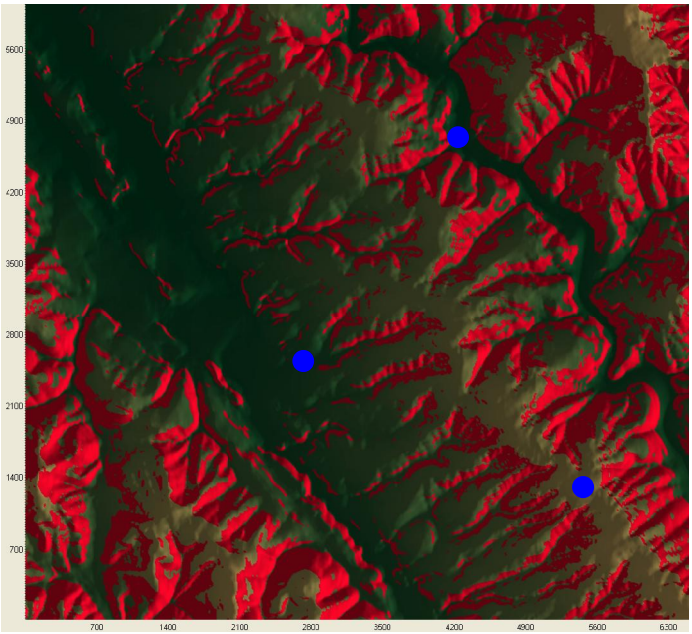


Figure 3.5 Mission waypoint positions (blue) overlaid on a terrain model with obstacles (red)

3.1.2.4 COST FUNCTIONS

The explorer and terrain models together specify all parameters necessary to determine the costs of each activity within an EVA. These numerous parameters serve as variables within a set of *cost functions* that calculate the predicted values for each necessary cost factor associated with an activity. From here, the explorer models produce the ultimate cost profile for that activity in terms of the fundamental cost factors: distance, time, and energy. The cost functions represent the final link in determining physical activity costs given the explorer and environment. Each explorer will have a unique set of these functions, formulated to provide accurately predicted activity cost values.

As an example, consider an astronaut traversal between two arbitrary points. From the explorer model, the unknown factors in this case are instantaneous velocity and power. A pair of cost functions will input all applicable parameters (human energetics, suit mobility, mass, gravity, sun lighting, terrain slopes, soil mechanics, etc.) and output the velocity and power estimates. Again per the explorer model, distance over velocity gives required time, and power integrated over time gives energy. Hence, the cost profile for this activity is fully determined.

In culmination, a conclusive cost function may operate upon the cost profile factors and any other relevant parameters in order to formulate an ultimate cost index for each activity, denoted the *exploration cost* (Table 3.2). While perhaps without a physical interpretation, this single value represents all EVA cost concerns weighed cumulatively for a given activity. It enables complete assessment of the relative costs of various activities effectively at a glance. More importantly, such functionality permits fully automated comparison of the total cost of all potential activities.

Table 3.2 Cost functions: input parameters from models and output cost factors

	Explorer	**Environment**	**Terrain**
Inputs	- Type - Weight - Activities	- Gravity - Sun position - Radiation	- Slopes - Obstacles - Soil mechanics
Outputs	**Exploration cost**, distance, time, energy		

3.1.3 EXPLORATION RETURN

As stated earlier, the primary goal of planetary exploration is scientific return. This is gained through the successful completion of objective activities at sites of interest. While all stated objectives are desirable, certain activities or sites may be more interesting or have a higher priority within the mission than others. Analogous to exploration cost, exploration return must also be expressed in measurable terms to enable comparison of activity scenarios. Modeling the exploration return involves associating relative return values to all EVA objective activities. Objectives with a higher prioritization or interest will correspond to respectively higher values of return. Moreover, the incremental gain in return with respect to activity duration at a site is by no means constant. This may be expressed by assigning respectively higher or lower return values to activities performed at different times throughout the stay at a site. Lastly, in the same manner as work is limited by cost constraints, scientific return is also bounded. That is, only a limited amount of interesting information can be gained from a region before it is exhausted and further work is fruitless.

The terrain itself can be characterized in terms of relative interest as well. Local terrain features such as craters or rilles, as well as distinct terrain properties such as chemical composition or radioactivity, can make certain areas of the terrain far more interesting, or in other words have a higher potential scientific return, than others. Traversal to or through these regions is preferential, as this promotes increased overall scientific return from the EVA. In the same manner as storing terrain cost parameter data such as obstacles or soil mechanics, scientific return data may be modeled as a corresponding matrix with relative values associated to each point in the terrain DEM.

Once quantitative scientific return data has been established, a “return function” may be defined which computes an ultimate scientific return index for each activity, denoted the *exploration return*. This single value represents the cumulatively weighed overall mission gain for a given activity. Analogous to the exploration cost, this enables quick and even automated numeric comparison of the projected scientific return for various potential EVA scenarios.

3.1.4 CREATING AN OPTIMIZED MISSION PLAN

Mission optimization denotes the process of maximizing return while minimizing cost and remaining within all constraints. Explorer and terrain models coupled with cost and return functions provide numerical estimates for the comprehensive exploration cost and return of each potential activity. These models further describe mission constraints in terms of upper bounds on the cost and return factors. This enables the actual optimization process to operate entirely numerically, which moreover permits fully automated performance.

Recall that on a particular mission, exploration activity may ultimately be limited either by costs due to constraints, or by return due to the exhaustion of interesting science. Automated mission optimization performs distinct functions in each of these cases.

Case 1: EVA limited by scientific return

In the case where interesting science is exhausted before reaching any operational constraints, the exploration return is fixed at a limiting value. Here, the function of mission optimization is to minimize the exploration cost given this fixed return value. This physically translates to completing all possible objective activities in the most cost efficient manner.

Case 2: EVA limited by operational constraints

The scenario where activity is limited by operational constraints is far more common in the real world. In this case, one or more cost factors reach their upper bound before all possible science has been conducted. Here, the function of mission optimization is to maximize the achievable scientific return given the limited permissible cost. This process is more complicated, and relies upon highly detailed models of incremental scientific return as well as clear prioritization of objectives. In this mode, the system must be capable of resolving issues such as whether it is more beneficial to remain at a current site, or to traverse to a new site and spend the remaining time and energy there.

The numeric optimization routine computes the best-case scenario of exploration cost and return values given a mission situation. A mission plan, in turn, details the actual physical activities associated with these optimal values. Mission plans explicitly describe task scheduling and

division of labor among team members. They also provide detailed explorer traverse routes between objective locations. Finally, they give the estimated physical costs incurred with each included activity in terms of desired cost parameters. An optimized mission plan clearly defines the set of activities that are maximally productive given the EVA situation. Once this plan has been developed, it is sent to the surface team for immediate execution.

3.2 REAL-TIME MISSION SUPPORT

Regardless of planning, mission productivity is ultimately determined by the actual activities carried out by the surface team of explorers. Once an optimized mission plan has been created, the primary goal of mission control shifts to providing support that enables the field explorers to accurately complete the planned activities on schedule. Crew members on the surface must manage mission information, exploration activities, navigation, safety, and constraints all in real-time. Providing support that relieves the burdens of comprehending mission information, navigating the surface, and monitoring constraints enables crew members to clearly focus on the specific task at hand. This promotes safer and more efficient performance of activities. Furthermore, this support aids in eliminating wasted time and energy due to deviations from the optimal plan.

As explorers react to uncertainty and contingencies in real-time, so must mission control. Hence, an effective mission support system must also enable planners and explorers to respond to the unexpected quickly and in a continuously optimal manner. This involves gathering feedback from the surface team and promptly generating a revised optimal mission plan in light of the current situation. As with all other mission aid, explorer support and activity re-planning should perform automatically with minimal need for human mediation. Such a complete support system not only manages burdensome reevaluation and decision making details, but also empowers consistently optimal explorer performance. Again, the distinct LOA involved in real-time support and re-planning must be considered (Table 3.1). As stated earlier, the system as a whole should maintain an LOA range from 2 to 5.

3.2.1 EXPLORER NAVIGATION

Traverses are most efficient when the explorers follow paths of least cost and highest interest, arriving at clearly distinguished destination sites. Determination of best-case traverse routes is a function of the mission plan optimization. Physically following a planned path, in turn, requires easy recognition of route locations coupled with a clear understanding of current position and heading, all in respect to the actual terrain and all in real-time. Providing this necessary support alleviates explorer disorientation and facilitates awareness and smooth correction when beginning to deviate from a planned route.

3.2.1.1 PATH MODELING

A traverse path may be expressed as a series of line segments along the terrain, connecting a starting point to a destination location. Hence, the only requisite to fully define a planned route is the set of endpoints of all such segments, with straight line travel assumed between consecutive points. In the same manner as mission waypoint positions are overlaid on an oriented terrain model, such traverse path points may be located and overlaid as well. Animating the interconnecting line segments produces a continuous rendering of the traverse route from start to finish. On a mission, the objective waypoints serve as the destination sites between which individual traverse paths are developed. By distinctly marking the waypoints along each route, all traverse paths and destination sites for an entire mission may be clearly identified within the terrain model.

3.2.1.2 POSITIONING AND MOTION CAPTURE

Depicting traverse paths and waypoints on a terrain model essentially yields a detailed map which the explorers are to follow. However, this alone offers little aid in dealing with uncertainty and, as known from the Apollo experience, still leaves explorers prone to disorientation and ambiguity in distinguishing current position and destinations (Márquez, 2007). In order to effectively follow a defined path, explorers must constantly assess their current location with respect to the path trajectory. Accurately determining explorer position, however, presents a significant challenge.

On earth, explorer location may be ascertained directly via the Global Positioning System (GPS). GPS receivers, worn or mounted on explorers, read various satellite signals to provide real-time surface position and heading to within less than ten meters (in some cases, less than three meters). This convenience does not extend to the moon or Mars, although such systems have been proposed (O'Keefe, Lachapelle, & Skone, 2004; Carney et al., 2005). Without a satellite-based positioning system, future EVA crews will have to rely upon inertial navigation technology, positioning with respect to a spread of surface beacons, or perhaps a hybrid of these systems (Titterton & Weston, 2004; Gorder, 2008). While this is a crucial area of open research in preparing for future moon and Mars missions, the development of these technologies is beyond the scope of this analysis.

Once an explorer position is known, it may be represented within the terrain model in the same fashion as all other spatial data. However, unlike planned waypoints and routes, explorer position is not static. An effective positioning system must continuously capture and express the motion of each explorer in real-time. Motion may be assessed indirectly by periodically sampling the explorer position. In this manner, differences in consecutive position readings specify the current heading and velocity. This continuous sampling forces the terrain model to become interactive, with the explorer position constantly updating as the explorer moves.

3.2.1.2 FOLLOWING A PLANNED PATH

Navigation support now becomes a matter of equipping surface team members with a complete model of the terrain, waypoints, traverse paths, and interactive current position. This enables explorers to clearly associate a planned traverse with the actual physical surroundings. For humans, a visual rendering is most beneficial. With such a display, astronauts may follow a planned path, recognize and correct any deviations, and pinpoint arrival at each waypoint simply by ensuring that the rendering of their position at all times coincides with the rendering of the determined route as they travel across the surface (Figure 3.6). Robots may automatically follow a planned path in the same manner, though instead of a visual interpretation, direct numeric positional data for paths, waypoints, and location feedback is best suited. Such a system greatly facilitates the implementation of an optimized traverse.

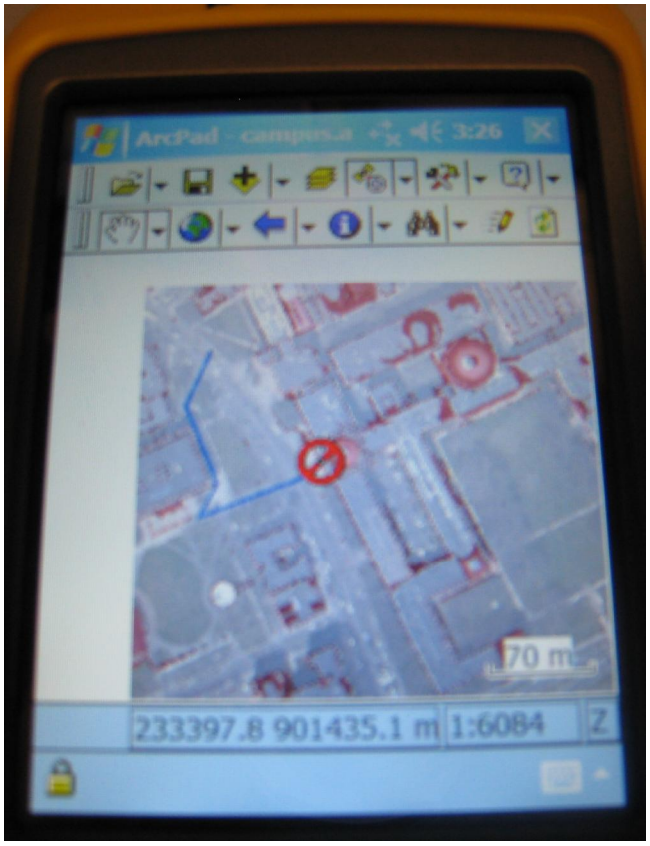


Figure 3.6 Handheld display showing terrain rendering with current explorer position (red circle) along a planned traverse path (blue lines)

3.2.2 MONITORING EXPLORER ENERGETICS

During a mission, both the astronaut physiological signs and robot energy levels must be constantly monitored to ensure that the explorers remain safely fit for activity and that no operational constraints are violated. Significant astronaut signals include heart rate, breathing rate, oxygen consumption, carbon-dioxide production, blood pressure, and body temperature. If any of these spike too high, activity may be suspended and the crewmember ordered to temporarily rest until admissible levels are restored. Such monitoring is relatively common, and occurs routinely for those operating in extreme conditions such as deep sea diving or present manned orbital operations (Asaravala, 2004). For a robot, general diagnostic factors include the electrical power drain, allocation of that power through the various robot systems, and operating temperatures. Monitoring these signals is likewise a standard process in present robots. All energetic data must automatically be transmitted to mission control in real-time so that current explorer conditions may continually be assessed and activities regulated (Figure 3.7).

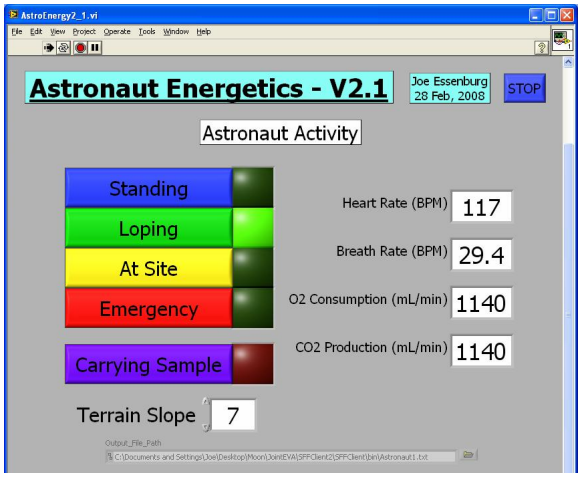


Figure 3.7 Example interface for monitoring astronaut energetics signals

The explorer cost models predict values for the energy expenditure of each activity. In turn, the streaming energetics data may be used to estimate the actual energy expenditure levels for each explorer. Comparing the predicted and actual values allows planners to gauge the accuracy of the cost models. While an isolated discrepancy may be due to a contingency factor or deviation from the plan, systematic differences would suggest an error in modeling. Revising the cost models according to incoming actual data promotes high-fidelity predictions for the costs of upcoming similar activities.

3.2.3 MISSION ALTERATIONS AND RE-PLANNING

Despite careful planning, unexpected developments are inherent in exploration. Crucial to preserving EVA productivity is the ability to robustly maintain optimal operation as mission scenarios change. Situation awareness is provided to mission control via information relayed from the surface explorers. In addition to passively transmitting physiological and energetics signals, explorers actively return scores of real-time data through regular observation and analysis. When discrepancies between a planned scenario and apparent reality accrue to warrant a response, the primary goal of mission control momentarily shifts from supporting the current explorer activities back to assessing the situation and developing a plan. This entails a reevaluation of the mission models and, in turn, optimizing a new set of activities in light of the latest explorer feedback. As a revised mission plan is established, the surface activity must adapt accordingly to maintain optimal productivity. This entire process must be fully streamlined to prevent wasted time and effort.

3.2.3.1 MODIFYING MISSION MODELS

The need to amend missions in response to new information demands that all aspects of the planning process become fully interactive. This means that every feature within the mission models must be made capable of regularly receiving updates to reflect new data. Completing revisions in real-time further stresses that processes be automated wherever possible, employing human involvement only when necessary. The general desired interactivity of each model aspect parallels that already established for explorer position (which automatically updates as the explorer moves), only now it is applied to substantially more complex elements comprising the mission cost, return, and objectives.

Revising explorer costs entails adjusting the cost profiles for each activity. For on-site activities, modeled only in terms of required time and energy, the editing process is quite straightforward. If a specific activity, for instance drilling, consistently requires a different time or effort than predicted, then the associated time or energy factor should be altered as fit. Traversal modeling is notably more elaborate, but the general editing process remains the same. Here, cost factor values are not assigned directly but rather are calculated via cost functions. In this case, the actual formulation of any discrepant parameter is what must be altered in order to accurately match reality. For example, if a robot consistently ascends hills faster than expected, then the cost function used in estimating the robot's traverse velocities should be amended as fit.

Terrain models are the easiest to update. Since all data (elevations, slopes, obstacles, soil mechanics, etc.) are stored as corresponding matrices, editing parameters entails simply entering new values at specified individual indices. With explorer position along these oriented matrices known, the specific data points corresponding to any physically observed terrain may quickly be determined. As explorers encounter a region with unpredicted properties, the entries at associated data points may be directly updated with the observed appropriate values.

Redressing scientific return models involves a hybrid approach. For on-site activities, the return value may be edited in the same manner as if editing a cost factor. For instance, if a specific activity begins to produce far more interesting results than expected, then the corresponding scientific return value for that activity should be raised. Conversely, if an activity is not

producing meaningful results, then the return value for that activity should decrease. In turn, the potential scientific return offered from various surface regions is modeled as a matrix in the same manner as the terrain parameters. Unexpected developments may be portrayed by editing all associated entries in the terrain scientific return matrix. For example, if the chemical composition of samples in a region was apparently interesting, but upon examination is not, then the scientific return values for all data points in that region should be appropriately lowered.

Beyond amending cost and return factors, refining the conclusive functions used to determine overall activity exploration cost and return indices is a more involved process requiring additional human reasoning. These functions weigh the relative importance of every aspect of an activity in order to assign an ultimate cost or return value. Hence, revising them involves reevaluating the significance of specific activity factors in terms of the overall mission. For example, if differences in soil properties are causing a greater impact on traverses than expected, this increased relative importance should be reflected in the cost function that determines the exploration cost of traversals.

Lastly we consider modifying objective destination sites as a whole. Editing mission waypoints is relatively simple, and involves selecting new objective locations as well as clearing existing waypoints as desired. The relative priorities or projected return of each waypoint in the group may be amended as well. The updated set of waypoints is then represented as usual within the terrain model.

3.2.3.2 IMPLEMENTING AN UPDATED MISSION

Once all mission models have been updated, the best course of action may be determined through the same process as the original mission plan. Incorporating the latest information, the overall exploration cost and return values are optimized within all constraints. Because the optimization routine is purely numeric, it may be performed automatically and extremely rapidly by computer, which is ideal considering the time pressure faced by mission control. The optimal operation scenario is expressed as a new mission plan. As before, the plan details activity schedules, division of labor, traverse routes, and estimated physical costs.

This revised mission information is then relayed to the surface team for immediate execution. As the explorers receive the new plan and promptly adapt their activities to match, mission control shifts its primary focus back to providing explorer activity support. Maintaining this process of updating mission models, optimally re-planning, and carrying out the revised plan ensures that the explorers in the field are at all times performing maximally productive activities based upon the best available data. As uncertainties impact each successive plan, the cycle repeats itself and the optimal plan adapts. This consistent support framework enables robust optimization of surface EVA operations.

3.2.4 CONTINGENCIES

Emergencies, accidents, discoveries, and a host of other unexpected events can drastically alter the makeup of an EVA. Developing a full set of contingency plans encompassing every possible scenario, from equipment failure and health concerns to unprecedented discoveries and emergency walk-backs, is a daunting task. However, this is a compulsory responsibility of mission control to ensure crew safety and productivity. Fortunately, contingency situations may be managed in a manner consistent with all other mission reevaluation and re-planning, thus taking advantage of the automated support framework already developed.

Mission contingencies induce an abrupt shift in the current situation facing the explorers. As in the case of nominal mission uncertainty, the consequence of a contingency scenario may be represented as a quantitative change in one or more factors within the mission models. As an example, suppose a robot traversing some distance from a team of astronauts malfunctions and becomes immobile. Although the astronauts could continue their planned activity, it may be most beneficial overall to recover the stuck robot. This scenario could be modeled by adding a new astronaut mission waypoint at the location of the robot, and further setting the exploration return value respectively high for attending to the robot. Another example is a high radiation event that forces explorers to immediately seek shelter. This contingency may be modeled as a prohibitively low constraint on the permissible surface activity time.

By representing the contingency situation in terms of the mission models, the best course of action may be determined by invoking the same optimization routine used in all mission re-

planning. The output will be a new mission plan which incorporates the contingency and provides the optimal operational response, while still satisfying all constraints and safety requirements. Relaying the new plan to the surface team enables them to immediately execute the best course of action and salvage as much EVA productivity as possible. Hence, the support framework is also robust in optimally handling mission contingencies. This provides considerable aid to human controllers in dealing with unexpected events.

3.2.5 RELAYING MISSION INFORMATION

Effective communication between the remote surface team and mission control is the final crucial link in accomplishing a maximally productive EVA. Data must continuously be sent from the surface to mission control for situation assessment, and in turn mission information must be sent from controllers back to the explorers. These exchanges need to happen seamlessly and without confusion to prevent wasted resources.

Communications from the surface team to mission control are relatively straightforward. Audio and video links provide direct verbal and visual assessment of EVA operations, while instrument readings, energetics signals, and even current position coordinates may be sent as routine data streams. This information grants mission control with full situation awareness, and can be used to update mission models as necessary.

When a new mission plan is made, the associated activities must be mutually understood by controllers and explorers. For astronauts, although verbal communication is readily available, it is insufficient for providing full mission comprehension, in particular navigational requirements. This is further impractical on Mars considering the time delay of more than three minutes in transmitting data to and from earth. Instead, traversing astronauts could greatly benefit from a simple display that clearly depicts the local terrain, current position, and the locations of traverse routes and waypoints. Mission control already has such a display available in the form of the rendered terrain model with overlaid mission data. However, remotely loading this full model as a display presents a challenge since astronauts on the surface lack the computing power and resources available to mission control. Fortunately, astronauts are not concerned with the actual

model numeric data or editing capabilities. Instead, they require only an image of the model rendering.

Hence, relaying traversal data that can be immediately understood and carried out by an astronaut becomes a simple matter of sending a "picture" of the terrain model with clearly indicated traverse routes and waypoints. Providing real-time navigation support, as explained earlier, further requires only including a simple interactive rendering of current astronaut position and heading within this display. For such a system, motion in reality will correspond to motion of the explorer position within the display. Moreover, ensuring that the displayed position follows the illustrated route corresponds to physically following the planned path in reality. To complete the mission plan description, activity schedules may be provided as simple text lists appended to the mission display. Example 2D and 3D mission information displays detailing the terrain, astronaut position, and a planned traverse route are shown in Figures 3.6 and 3.8, respectively.

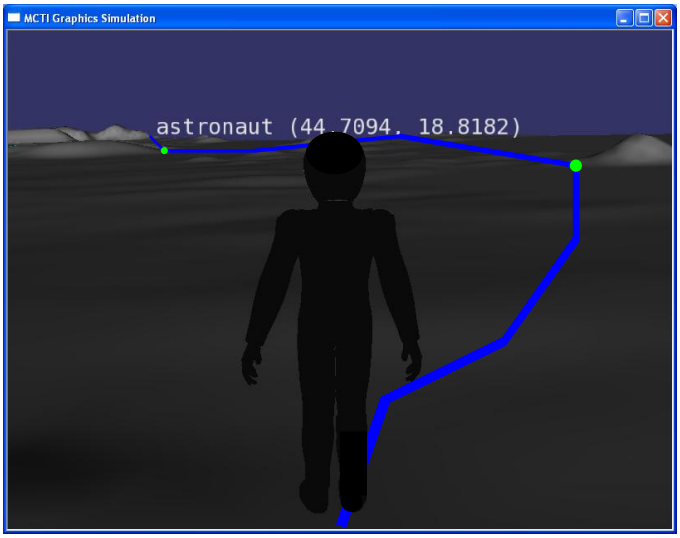


Figure 3.8 Concept 3D mission information display showing terrain rendering with astronaut position along a planned traverse route

Various astronaut information display concepts are shown in Figure 3.9. Perhaps the most favorable of these is the hands-free heads-up display depicted at left. Here, an image is projected within the space suit helmet near the top of the astronaut's field of vision. Such a system would enable seamless and intuitive astronaut interaction with both a newly received mission plan and the upcoming physical terrain.

Figure 3.9 Mission information display concepts
From left to right: heads-up display, space suit imbedded screen, computer screen
(NASA images s99_04197, jsc2004e18850, jsc2004e18859)

Robots may also automatically update their activity via the models available from mission control. Conversely to astronauts, robots have no use for a visual display. Instead, controllers would transmit the relevant numerical mission data. In particular, sending robots the DEM, planned traverse route and waypoint coordinates, and specific activity commands equips them with a complete understanding of the mission situation and objectives. Coupling this information with real-time positional feedback potentially enables fully automated execution of the planned EVA. Adaptation to mission revisions is further a simple matter of downloading the new mission information and implementing it in place of the previous plan. Hence, the complete support system can permit automated and continuously optimal robot operation.

4 PATHMASTER: A MISSION PLANNING AND SUPPORT PROTOTYPE

4.1 DEVELOPING A MISSION SUPPORT SYSTEM

Ensuring optimal EVA performance throughout all situations necessitates comprehensive automated operational support. A prototype support system has been developed to aid in fulfilling the crucial mission productivity criteria of planning and physical execution. This system enables the planning of optimized explorer traversals, operation scenario comparison, limited field navigation, and mission re-planning. Hence, it is designed to be utilized both beforehand by mission planners, as well as in real-time by explorers for navigation support and mission control for decision making.

This prototype was developed to implement a subset of the factors identified in Chapter 3. Planning begins by loading an elevation map of the physical mission terrain and providing orientation information. General EVA parameters that function as inputs for the eventual determination of activity costs are given next. These include the number, type, and mass of field explorers, planet and time of the mission, and the maximum traversable surface slope. A scaled terrain interface next allows planners to locate mission waypoints for each explorer as well as enter terrain data parameters along the surface including obstacles, soil mechanics, scientific return, and other potential options. Waypoints define the mission objectives, and terrain obstacles represent a sole operational constraint. Collectively, this information forms the characterization of a mission situation.

Once all mission inputs have been entered, traverse paths for each explorer are found by invoking an optimization routine. This process computes a specific numeric cost for each incremental step along the surface and works to minimize that cost while avoiding any obstacles in determining a route. The final output hence delineates valid paths of minimal total cost from waypoint to waypoint. The predicted physical costs for these traverses are presented as well in terms of distance, required time, and energetic expenditure. A comprehensive dataset and visual

display depicting the terrain, objective waypoints, optimized route trajectories, and associated costs constitute an EVA mission plan.

The surface team may use the portrayal of the mission plan alone as a detailed map for guidance across the terrain. Interactive navigation support is additionally available by feeding the mission information to certain separate systems. An audio/video link with the explorers enables them to continually provide feedback while performing activities. As missions develop or contingencies arise, mission control revisits the terrain interface and updates data parameters as fit. The optimization process is repeated, a revised plan is generated, and the EVA operation cycle proceeds.

In order to promote real-time EVA situational response, crucial elements of the support system should be automated to the highest practical degree. Recall that there are ten distinct levels of automation (LOA), presented again in Table 4.1. As stated earlier, the overall LOA range for the system is limited from 2 to 5.

Table 4.1 Levels of automation (Parasuraman et al., 2000)

Automation Level	Automation description: The computer…
1	offers no assistance: human must take all decisions and actions
2	offers a complete set of decision/action alternatives
3	narrows the selection down to a few
4	suggests one alternative
5	executes the suggestion if the human approves
6	allows the human a restricted time to veto before automatic execution
7	executes automatically, then necessarily informs the human
8	informs the human only if asked
9	informs the human only if it, the computer, decides to
10	decides everything and acts autonomously, ignoring the human

There are three distinct time-pressured, labor intensive functions demanded of mission control: updating mission models, generating an optimized plan, and conveying the new information. The majority of real-time feedback from the surface team comes as verbal reports or visual images. Automated interpretation of this information would be a daunting task; instead, human reasoning is well suited for quickly translating such qualitative data into distinct parameter values. That said, the support system can mitigate this task by streamlining the editing process for these parameters and, when applicable, simplifying their representation into a limited set of discrete

values. For instance, subjective local soil mechanics feedback, while interpreted by a human controller, can be classified into a discrete numeric index and instantly entered into the terrain model with the click of a mouse. This desirable model updating functionality represents an LOA of between 2 and 3.

When creating a subsequent mission plan, we are only interested in one set of paths: the optimal ones. Since the optimization routine is entirely numeric, it requires no human involvement. The LOA of mission plan generation, hence, is limited only due to the high risk nature of the domain in which we are operating (Sarter & Schroeder, 2001). For safety, human controllers must ultimately assess, amend, and approve any plan before it is commanded to the field team. Here, the scenario generated by the support system serves as a nominally optimized suggestion for mission control. Therefore, mission re-planning functions with an LOA of 4.

Once a plan is decided upon, it is relayed to all parties. This involves loading mission information as a display image and transferring all necessary data between remote hardware systems. This functionality ideally occurs with an LOA of 5, where specified mission data is automatically interpreted and transmitted as soon as human controllers approve. Currently, though, these tasks are currently performed mostly manually through standard computer procedures, pending further system development.

While this implementation may be a somewhat limited subset of the comprehensive functionality presented in Chapter 3, it correlates well with the operation of any remote geological excursion where exploration costs and scheduling are primarily determined by the traversals between sites of interest. This is a reasonable analogue of the Apollo missions and, perhaps, the first manned missions back on the moon. Moreover, this prototype highlights the general architecture by which more versatile, higher-fidelity mission support systems can be developed.

This chapter provides a comprehensive overview of the developed mission support system, named Pathmaster. Pathmaster is written as a single file entirely in MATLAB, called an m-file. It runs as a series of GUIs where users may quickly and intuitively enter mission information and generate optimized mission plans in near real-time. All features are explained in detail here. The

Pathmaster User Manual is presented in Appendix B, and the MATLAB code is recorded in Appendix C; these are also included on the enclosed DVD-ROM.

4.2 OPENING PATHMASTER

Pathmaster is written for both Windows and Mac OS X. It is intended to be run in MATLAB R2007a or later. A minimum monitor resolution setting of 1024 by 768 pixels is recommended. Upon opening MATLAB and setting the file search path appropriately, Pathmaster is called directly from the command line. There are four general options when opening the program:

>> **pathmaster**

The command "pathmaster" alone will initialize a prompt to load elevation data from file. This is the normal method of running Pathmaster.

```
>> pathmaster
```

>> **pathmaster(Elevmap)**

Calling Pathmaster with a matrix argument loads that matrix as the elevation map.

```
>> [a,b] = meshgrid(0:.1:10);
>> elev = sin(a+b);
>> pathmaster(elev)
```

>> **pathmaster('lite')**

Calling Pathmaster with the 'lite' option employs simpler surface rendering. This speeds plotting time and prevents problems on some machines, and will be discussed later.

```
>> pathmaster('lite')
```

>> **pathmaster(Elevmap,'lite')** **OR** **>> pathmaster('lite',Elevmap)**

Calling Pathmaster with both a matrix argument and the 'lite' option does both of the above. The arguments may be entered in any order.

```
>> pathmaster(elev,'lite')
>> pathmaster('lite',elev)
```

4.3 PLANNING A MISSION

Pathmaster is currently most functional as a mission traverse planning and re-planning tool. Upon opening, a terrain elevation map is loaded and all general EVA input parameters are subsequently entered through program menus. The main mission planning GUI, complete with a scaled interactive terrain rendering, next enables point-and-click editing of mission waypoints and terrain characteristics. When finished, an optimization routine determines paths of minimized cost between successive waypoints, avoiding any obstacles. These paths are depicted

within the display along with predicted physical cost data, which together comprise a mission plan. For making higher level strategic mission decisions, various separate mission scenarios and their respective optimized plans may be compared side-by-side.

4.3.1 LOADING ELEVATION MAPS

After being called from the Matlab command line, Pathmaster will open a prompt allowing the user to select an elevation map to be loaded from file (Figure 4.1). In the case where a matrix argument was entered at the command line, that matrix is loaded and this step is bypassed. Elevation data may come via either a text file or a MATLAB data file. MATLAB data files are used to save workspace variables, which are stored under individual "fields". If such a file with multiple stored fields is selected, a subsequent prompt will ask to specify the elevation data (Figure 4.1). Chosen files may also contain a host of additional data which Pathmaster will automatically recognize and load. This can include map information parameters, additional terrain data maps, and even pre-defined mission waypoints.

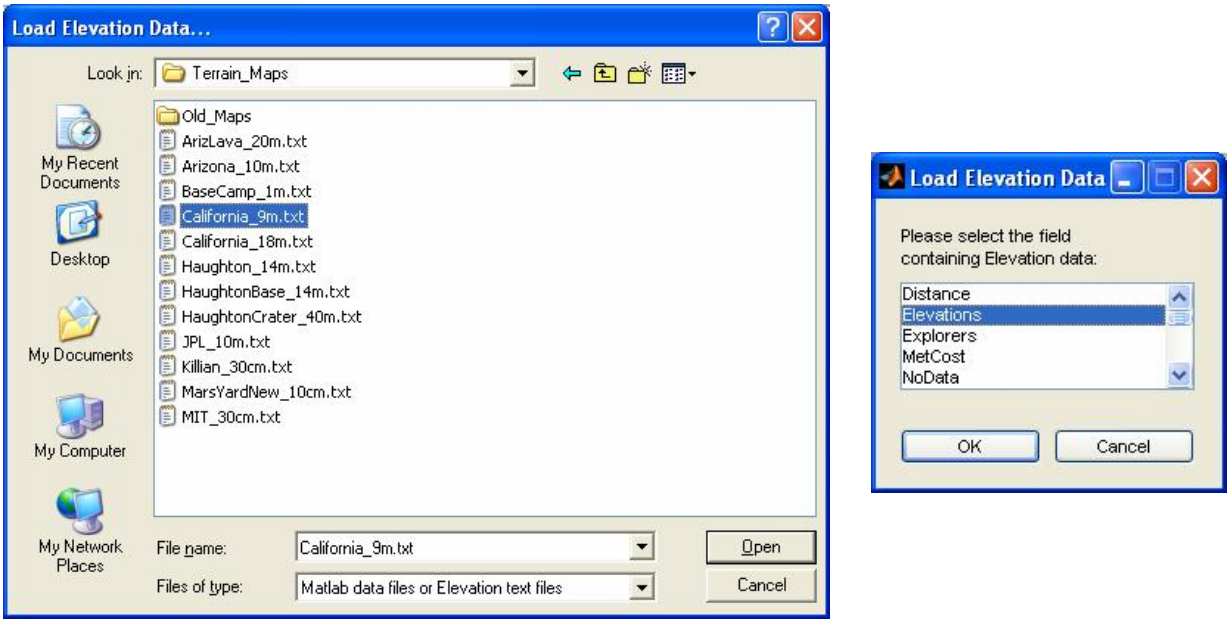


Figure 4.1 Elevation data file prompts

The elevation maps used by Pathmaster are arranged as a rectangular matrix. Matrix indices correspond to a regular grid projected horizontally across the physical terrain being modeled. The data stored at each point represents the relative terrain elevation, in meters, at the corresponding physical location.

4.3.2 ENTERING MAP INFORMATION

Once an elevation map has been loaded, Pathmaster will open the Map Information menu (Figure 4.2). Here, the size of the elevation map matrix is given in rows and columns, and the user may enter the map sizing and, if applicable, positioning data. In most cases, this data will already exist within the selected file. Text files store map information as a set of header lines before the elevation matrix begins. Matlab data files, in turn, store additional information as separate data fields. All existing data is automatically recognized and displayed in the corresponding data fields.

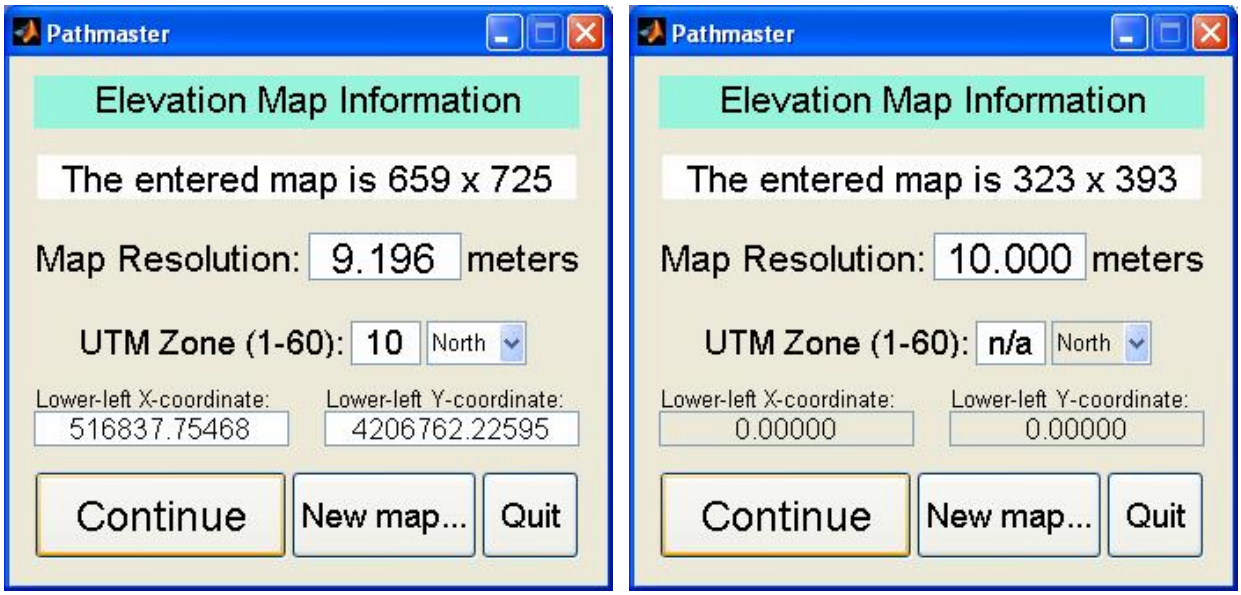


Figure 4.2 Map Information Menu with global positioning active (left) and inactive (right)

4.3.2.1 MAP RESOLUTION

As explained, the elevation map corresponds to a regular grid of data points projected over the physical terrain. Map resolution denotes the uniform horizontal spacing between adjacent data points, given in meters. This value is entered in the first data field on the Map Information menu. A map with a smaller such distance is said to have a higher resolution since data points are more densely recorded along the terrain.

4.3.2.2 GLOBAL POSITIONING

The remaining data fields in this menu are optional, and they are used to identify the global position of the mapped terrain. This system, currently applicable only on earth, enables the calculation of latitude and longitude coordinates for any point in the map. When enlisting this functionality, Pathmaster necessarily assumes that north is in the upward direction, or topmost row, of the loaded map matrix. Positioning is given in terms of the Universal Transverse Mercator (UTM) system, which comprehensively divides the earth into distinct zones each with independent Cartesian-based surface coordinate projections (Riesterer, 2008). The first field, UTM zone, represents the east-west zone within which the map is located, numbered 1 through 60. The neighboring dropdown menu then specifies whether the map is in the northern or southern hemisphere. The lower-left X and Y coordinates finally denote the exact location of the southwest corner of the mapped terrain within the UTM zone. The X-coordinate specifies the "easting", or meters east of the zone origin, while the Y-coordinate specifies the "northing", or meters north of the zone origin. The positioning feature may be deactivated by entering a zero into the UTM zone field.

4.3.3 ENTERING EVA INPUTS

Upon pressing "Continue" in the Map Information menu, Pathmaster opens the EVA Input menu (Figure 4.3). Here the user enters all general parameters of the EVA, including a designated name, the number and character of explorers, the planet upon which the mission is run, the time at which the mission begins, maximum traversable surface slope, and the directory to which certain output files are written. In addition, if any existing terrain map data or mission waypoints were stored along with the elevation file chosen when opening Pathmaster, then options to load this data will appear as a series of check-boxes near the top of this menu. All of these parameters are used as inputs in determining the costs of any subsequent mission traversals. Aside from individually defined explorer type and mass, these parameters uniformly apply to all explorers.

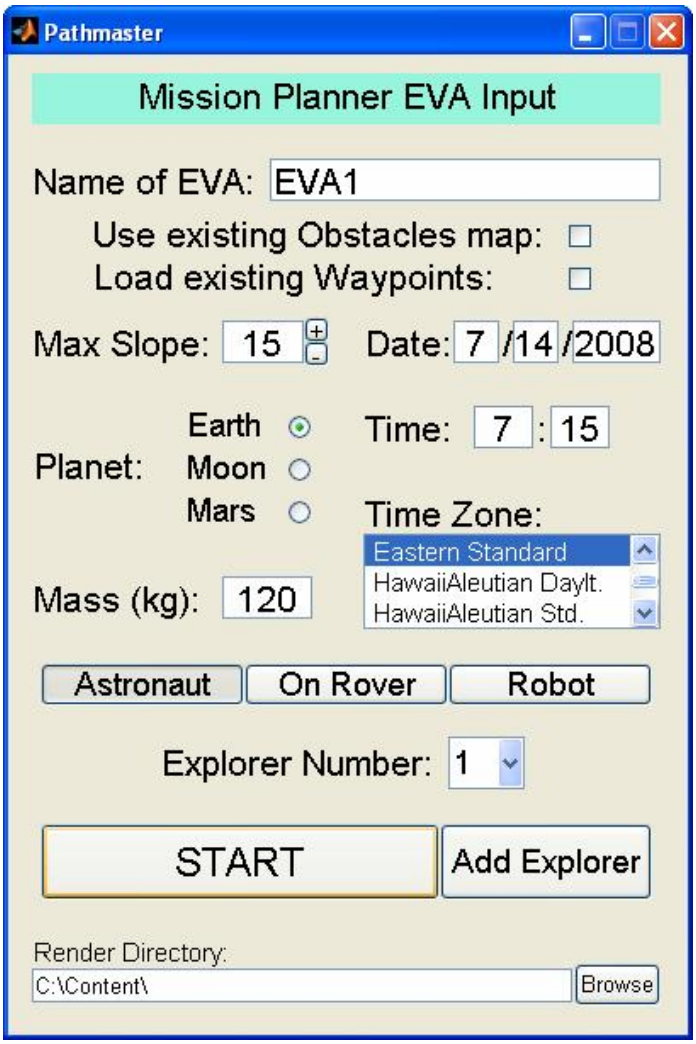


Figure 4.3 EVA Input menu

4.3.3.1 MAXIMUM TRAVERSABLE SLOPE

The "Max Slope" field in the EVA Input menu denotes the maximum permissible surface slope over which explorers may cross, given in degrees. Any areas of the terrain with a local slope greater than this value will be presented as terrain obstacles, which explorers must avoid and navigate around. In this way, the slope value represents an operational constraint on mission traverses. This constraint exists both to spare heavy exertion by the explorers in crossing these difficult areas as well as for safety to keep away from areas where they may be prone to sliding or falling over. Typical values for the maximum slope are between 10 and 20 degrees. By their nature, terrain features such as boulders, crater walls, ravines, cliffs, and rough patches involve particularly steep changes in elevation, and hence they will appropriately appear as obstacles.

4.3.3.2 PLANET SELECTION

Pathmaster accommodates missions on the surface of earth, the moon, and Mars. Selecting the planet upon which a mission is to take place determines the assumed environmental gravity. Gravity on earth is assumed to be 9.8 meters per second squared; it is approximated as one-sixth of that value on the moon, and one-third of that value on Mars. The planet selection also sets the default render mode for the terrain display, discussed later.

4.3.3.3 TIME OF MISSION AND SUN POSITION

The date, time, and time zone precisely define the point at which a mission begins. By default, the current computer time is entered in these fields. Time of day is recognized as military time, with hours ranging from 0 to 23. These values are used to determine the sun illumination on the mission surface. This data can be converted into Coordinated Universal Time (UTC). Coupled with known planetary locations provided by Pathmaster's positioning feature, the relative sun position in terms of azimuth and elevation angles may be mathematically determined.

4.3.3.4 CHARACTERIZING EXPLORERS

New explorers may be added to a mission team by clicking on the "Add Explorer" button in the EVA Input menu. Explorers are individually characterized by their general type and mass. There are three types of explorers recognized in Pathmaster: astronauts, rovers, and robots. Astronauts are suited humans on foot. Rovers refer to transportation vehicles which carry astronauts and robots, such as the LRV, and all-terrain vehicle on earth, or eventually pressurized rovers. Lastly, robots are unmanned surface exploration machines. Under these criteria, certain systems which may commonly be referred to as a "rover", such as a MER style explorer, are classified in Pathmaster as a robot. An explorer's type is selected by clicking on the corresponding buttons in the menu, and the appropriate mass in kilograms is entered in the "Mass" field. This information is used in determining distinct activity costs.

4.3.3.5 DATA OUTPUT

Pathmaster stores all mission data and parameters to a series of output files while running the program. The name specified for an EVA is shared by all corresponding files to enable easy recognition. Data for a new mission is first generated when the user clicks "START" in the EVA

Input menu. All saved mission information is automatically written to a Matlab data file located in the same directory as the Pathmaster m-file. In addition, a separate "Render" directory receives specially formatted text files containing the mission data. This directory is specified in a field at the bottom of the EVA Input menu. The files written to this location may be employed by an independent render engine in order to create additional simulated mission displays. Such systems will be discussed later.

4.3.4 TERRAIN DISPLAY

Once all map information and EVA input parameters have been entered, the Mission Planner GUI opens (Figure 4.4). This is Pathmaster's main interface, where users may view the terrain rendering, edit waypoints, edit terrain characteristics, find traverse paths, and display all mission information. The GUI includes an interactive terrain display accompanied by a menu of controls and data fields at the top.

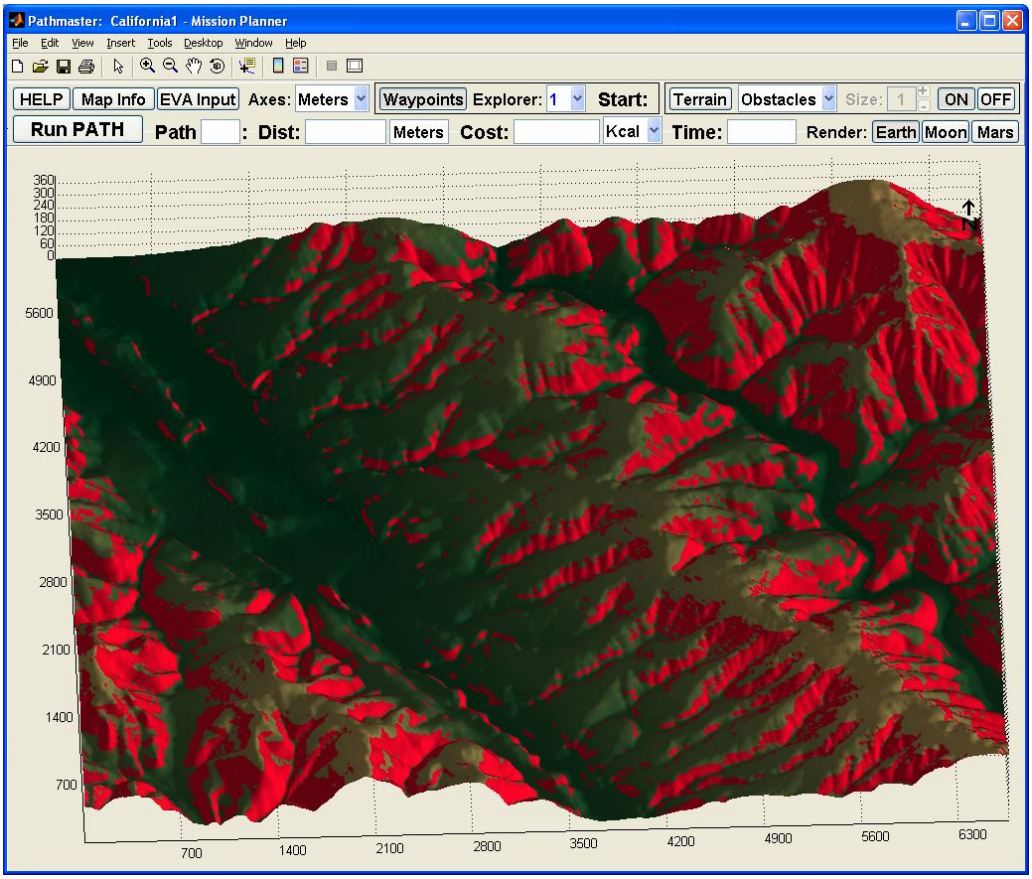


Figure 4.4 Mission Planner GUI

4.3.4.1 SURFACE APPEARANCE

The terrain is rendered as a 2.5D surface plot oriented within a scaled XYZ-coordinate system. Recall that map data is loaded as a matrix. Pathmaster assumes that north is in the direction of the topmost row of the map. In this intuitive manner, the X-axis is oriented west to east and corresponds to differing matrix columns, and the Y-axis is oriented south to north and corresponds to differing matrix rows. The origin is defined in the southwest corner. Map data points are plotted in this corresponding XY-orientation, maintaining a uniform spacing as defined by the map resolution distance. The Z-value for each data point is the recorded physical elevation. In this way, the topography visually appears as it would from an aerial view. If the global positioning feature is active, a compass will appear in the northeast corner of the map indicating the implied northern direction.

The scaling of the terrain axes corresponds to actual physical distances. Scales may be displayed in units of meters, kilometers, feet, or miles. The desired units are chosen through the "Axes" drop-down menu (Figure 4.5). When a new selection is made, the surface axes and gridlines will automatically update with new spacing and tick marks as fit.

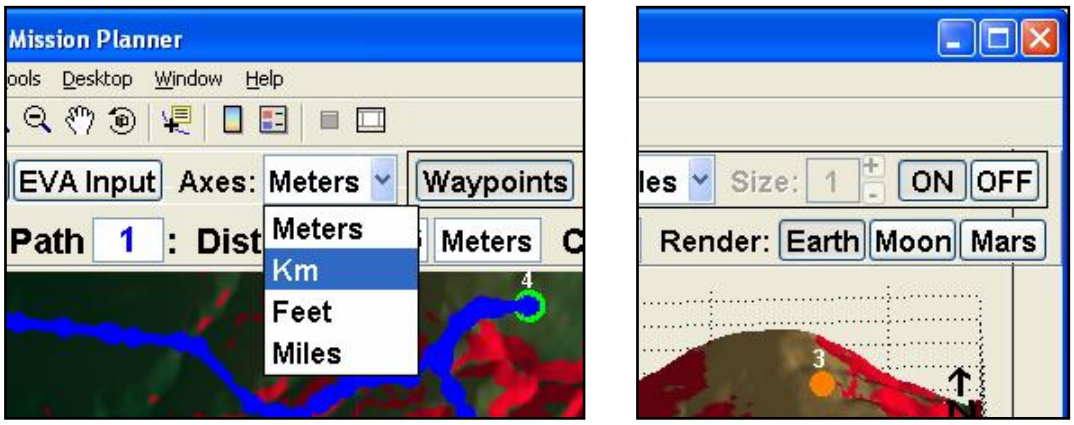


Figure 4.5 Terrain surface appearance options
Left: axes scaling selection; Right: render mode options

The coloring of the terrain surface is adjustable, and can be set to mimic a chosen planet. Buttons at the right of the menu allow users to select between distinct earth, moon, or Mars representative render modes (Figure 4.5). The initial render mode is determined by the entered mission planet. Changing the render mode, in turn, affects only the display and does not alter the stored planet or gravity. The available render modes are portrayed in Figure 4.6.

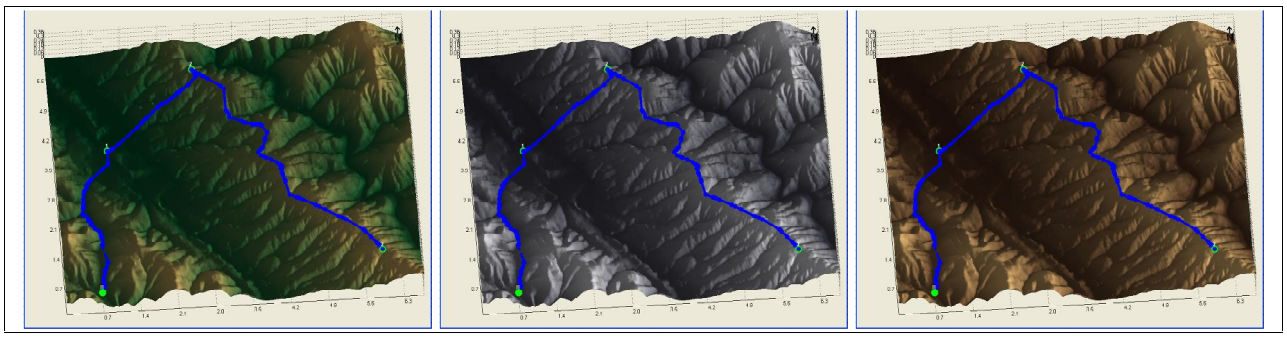


Figure 4.6 Surface render modes. From left to right: Earth, Moon, Mars

4.3.4.2 DATA LAYERS

Beyond the elevation data, additional terrain data parameters employed by Pathmaster include obstacles, soil mechanics, scientific return, and possibly other information. This data is stored as a series of corresponding matrices, which are "layered" in the sense that a distinct value for each terrain parameter is specified at each point in the elevation map. Pathmaster enables the data for each individual parameter, or data layer, to be visualized as a colored rendering across the surface. The current layer is chosen in the terrain drop down menu, and its display may be turned on or off with the toggle buttons to the right (Figure 4.7). If no other display is active, the elevation rendering will show.

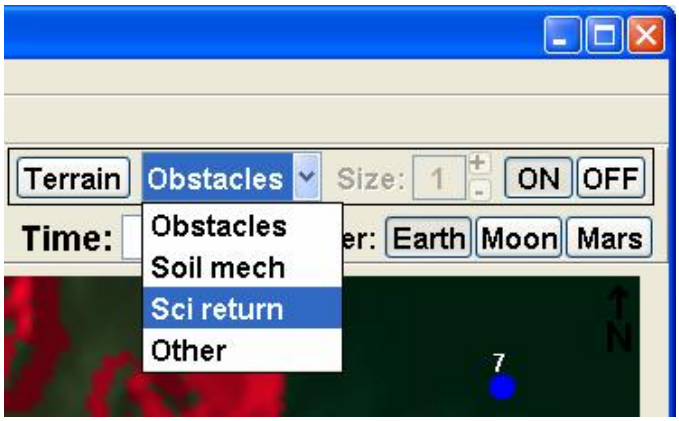


Figure 4.7 Terrain data layer display options

By default, the obstacles are displayed when the Mission Planner GUI opens. Unless a custom obstacles map is loaded, the initial obstacles represent all regions of the terrain where the local slope, found via a gradient operation on the elevation data, is greater than the defined maximum

traversable slope. This layer is clearly distinguished as bright red areas superimposed on the elevation rendering. Every location colored in red represents an obstacle. Data corresponding to soil mechanics and scientific return, on the other hand, are not binary. These layers are presented in a gray to maroon or gray to purple rendering, respectively. Areas colored in gray represent negligible significance in terms of the parameter, whereas darker areas denote a high significance. Specific parameter values will be discussed later. Example data layer displays are shown in Figure 4.8.

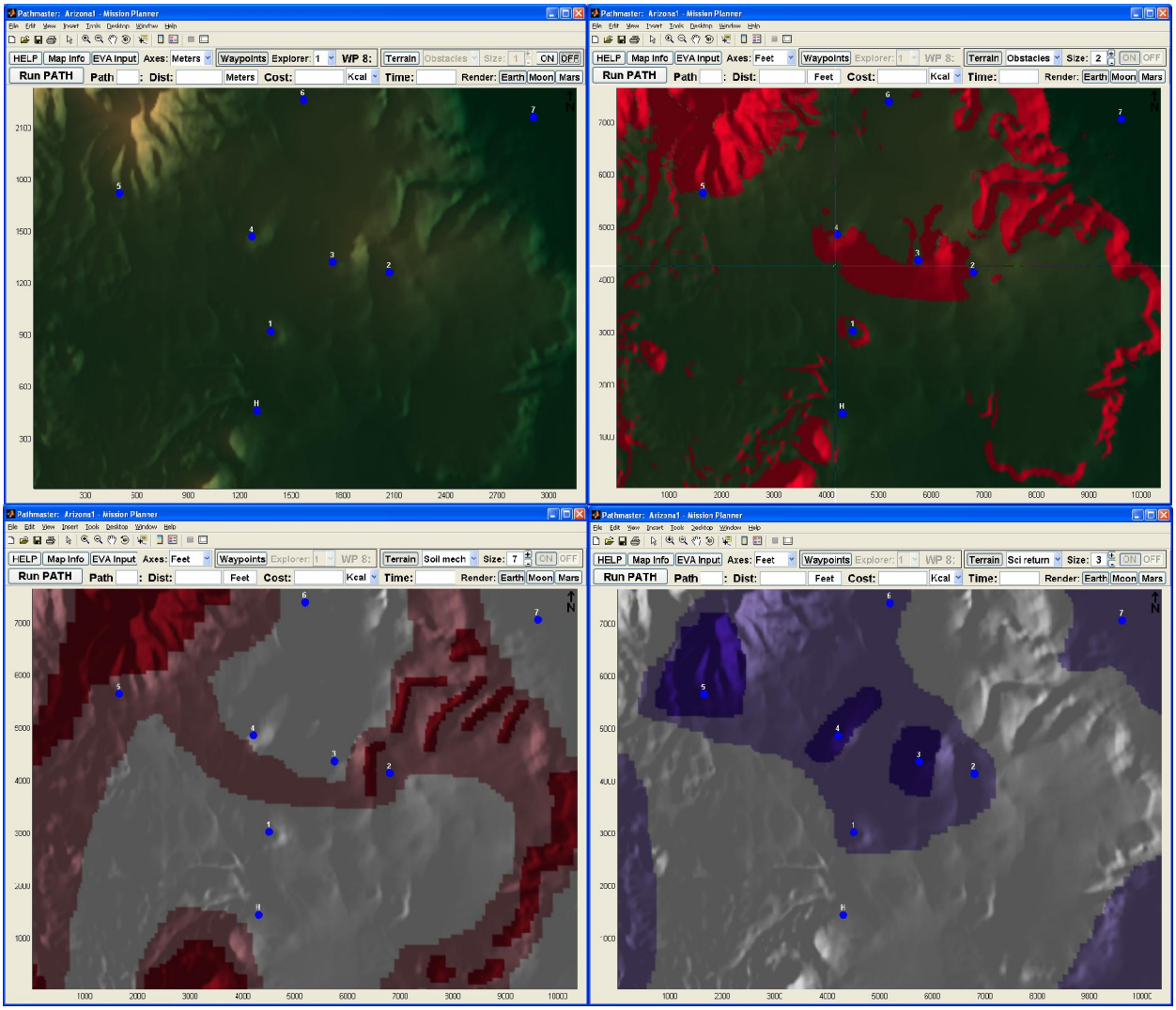


Figure 4.8 Terrain data layer displays: elevations (top left), obstacles (top right), soil mechanics (bottom left), scientific return (bottom right)

4.3.4.3 SUN ILLUMINATION

When rendering the surface, Pathmaster creates a light source that mimics the sun in providing illumination conditions and giving contrast to terrain features. The position of this light is determined by the time at which a mission is run. The current algorithm is relatively simple, and places the lighting directly to the east at 6:00 AM and directly west at 6:00 PM, with varying azimuth and elevation in between. An example of different lighting conditions is shown in Figure 4.9. This limited functionality is only a temporary measure pending the implementation of a complete sun positioning algorithm, the majority of which has been developed.

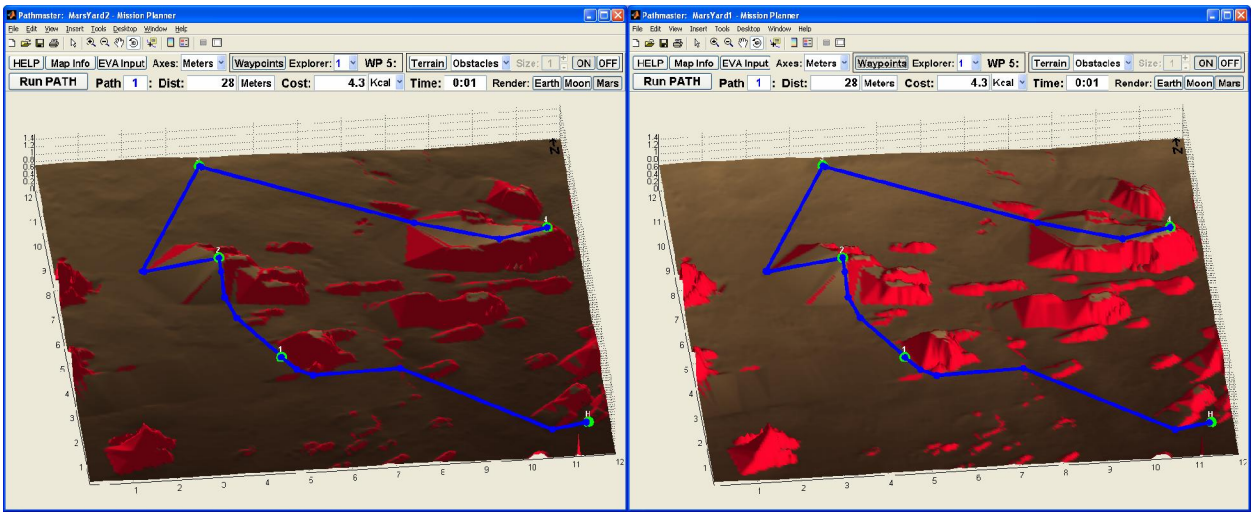


Figure 4.9 Simulated sun illumination at midnight (left) and 9:30 AM (right)

4.3.4.4 DATA DISPLAY

Local terrain data may be displayed by right-clicking anywhere along the surface. The provided data, shown at top in Figure 4.10, includes elevation and slope as well as any soil mechanics, scientific return, or other data that has been specified. If the global positioning feature is active, then the latitude and longitude of the selected location will be given as well. This display feature provides immediate access to quantitative terrain data and enables the precise location of interesting sites along the surface.

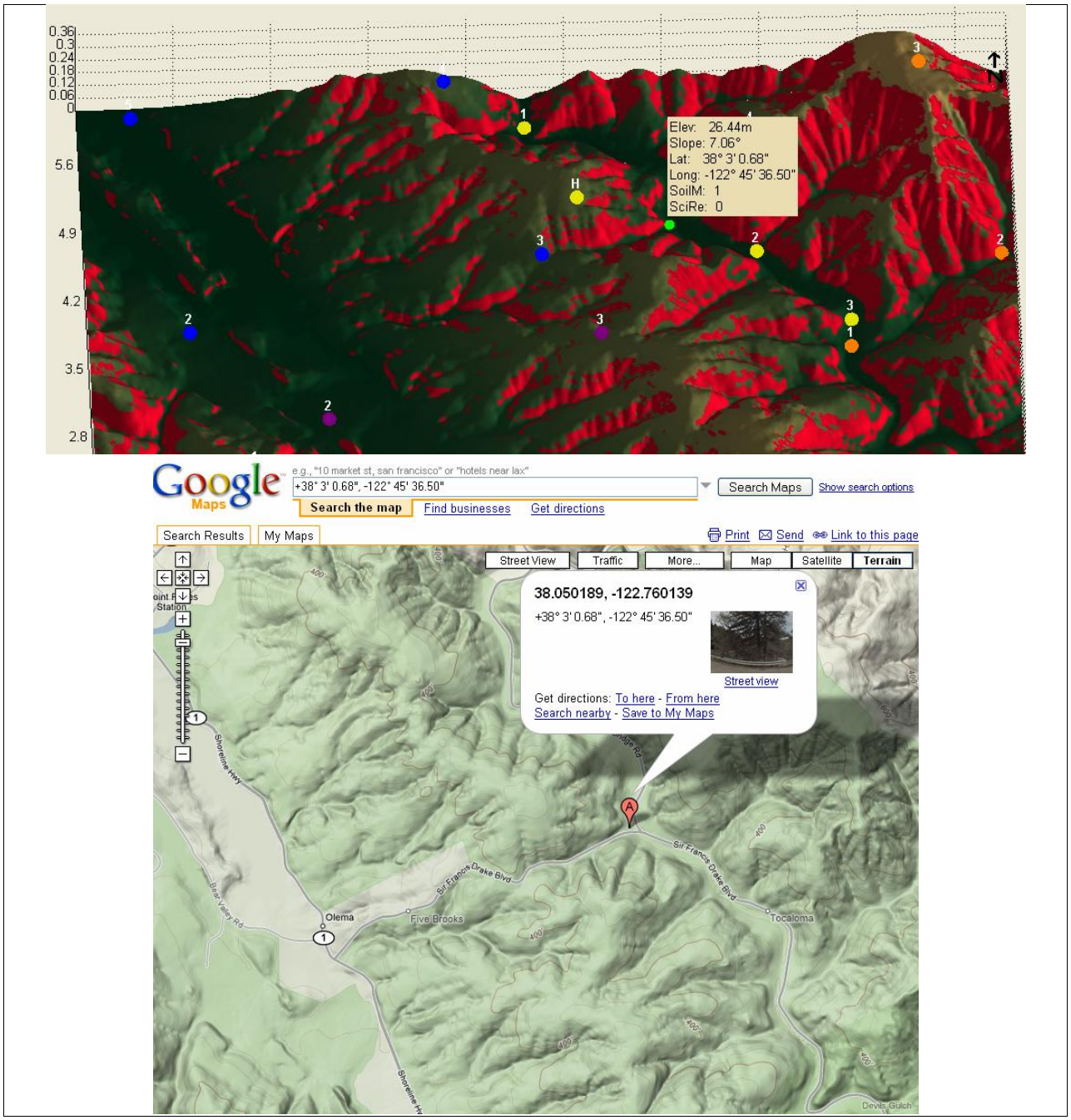


Figure 4.10 Using the terrain data display (top) to locate a feature identified through external mapping software, in this case Google Maps (bottom)

4.3.4.5 EXTERNAL MAPPING SUPPORT

Beyond solely relying on Pathmaster, the global positioning feature coupled with the terrain data display allows users to effectively operate alongside independent mapping systems such as Google Maps or ArcGIS for additional support (Figure 4.10). These external systems can

provide a significantly higher-fidelity characterization of the mission terrain than offered by Pathmaster alone. Such advantages include satellite imagery as well as the ability to handle much more detailed terrain parameter databases. Any features of interest identified in Google Maps or ArcGIS may be located in terms of latitude and longitude coordinates. These positions, confirmed via the terrain data display, can then be precisely recorded in the Pathmaster model.

4.3.5 DEFINING MISSION WAYPOINTS

In Pathmaster, mission objectives are defined as a set of waypoints along the terrain to be visited. Each explorer follows a unique set of corresponding waypoints which are entered separately and color coded. When the "Waypoints" button in the Mission Planner GUI is depressed, waypoint edit mode is active. Individual explorers may then be selected with the "Explorer" drop-down menu (Figure 4.11). Left-clicking on the surface display will add a waypoint at that location for the chosen explorer. In turn, holding Shift while clicking anywhere on the surface deletes the latest explorer waypoint. When determining traverses, Pathmaster currently visits waypoints in the order in which they were entered, regardless of surface position. This ordering is indicated by a small numeral appearing above each waypoint. The first waypoint entered is labeled as "H" for "home", and successive waypoints are numbered beginning with 1. Such a planning scheme relies upon human reasoning to determine the overall order in which to visit waypoints. In general, waypoints with a higher priority should be entered earlier, though relative surface positions must also be considered. Pathmaster does offer support in directly comparing various potential ordering scenarios, as will be discussed later.

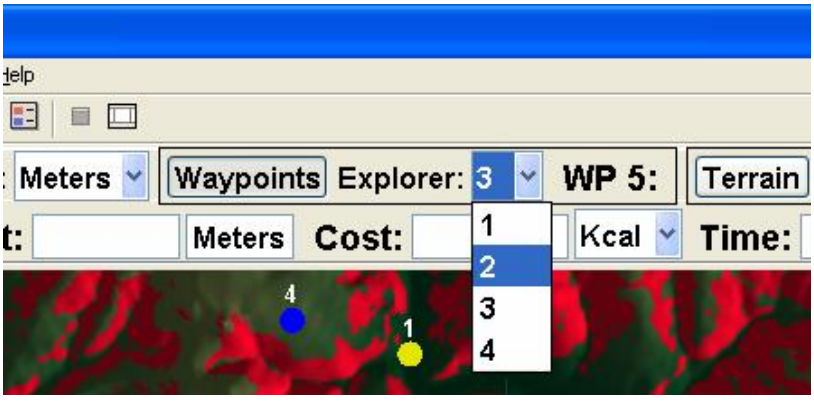


Figure 4.11 Waypoint edit controls

4.3.6 EDITING TERRAIN CHARACTERISTICS

Users can also manually edit the terrain data layers as desired. These parameters including obstacles, soil mechanics, scientific return, and possibly other data may be employed in the determination of traverse routes and costs. Recall that all terrain layers are stored as a matrix. Editing a layer hence involves entering new values into specified indices within these matrices. Terrain edit mode is activated by clicking on the "Terrain" button in the Mission Planner GUI. This will force the terrain layer display to be turned on, and the active layer may be selected with the neighboring drop-down menu (Figure 4.12). In this mode, left-clicking, holding Shift while clicking, and even double-clicking on the surface will perform various edits to all corresponding parameter data values within a distinct rectangle surrounding the point clicked. The relative size of this "edit rectangle" may be adjusted with the "Size" control as shown in Figure 4.12. The size value is altered by clicking on the neighboring increment and decrement buttons, and can range from 0.1 to 10. The actual number displayed corresponds to an approximate percentage of the total map X and Y size that the edit rectangle will encompass. The intuitively functionality of the terrain editor is comparable to simple drawing software such as MS Paint. Refer to Figure 4.8 for visualizations of the terrain data layers which may be edited.

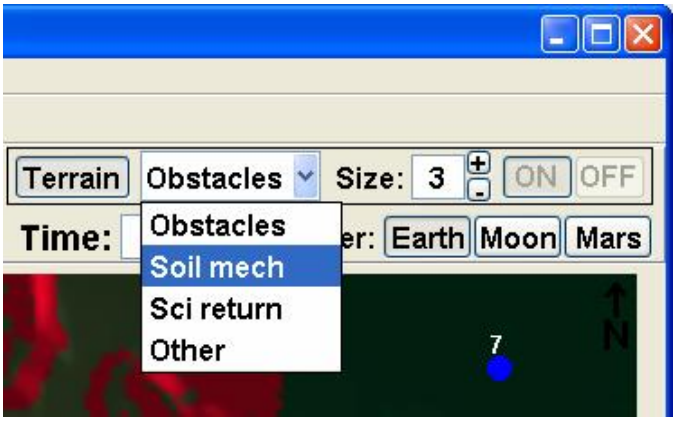


Figure 4.12 Terrain edit controls

4.3.6.1 OBSTACLES

Obstacles are non-traversable areas of the terrain, initially set as all points where the local slope is greater than the chosen maximum traversable slope. These impose constraints on permissible traverse trajectories, as these regions must be avoided. Obstacle data is binary. All points along the terrain representing an obstacle will store an obstacle value of one. Meanwhile, all

traversable points will store an obstacle value of zero. Left-clicking or double-clicking on the surface will add obstacles, or set all obstacle values within the edit rectangle to one. Conversely, holding Shift while clicking will clear obstacles, or set all points within the edit rectangle to zero. Caution is necessary when editing obstacles to ensure that mission waypoints do not become enclosed by them. Such a situation makes reaching the affected waypoint operationally impossible, and Pathmaster will be unable to determine a traverse path for the corresponding explorer and instead will return a warning. When viewing the obstacles layer, all points with an obstacles value of one will appear in bright red, while all traversable areas will be shown with the regular elevation rendering.

4.3.6.2 SOIL MECHANICS

Soil mechanics refer to qualities of the terrain surface in terms of rockiness and rock distribution, firmness, strength, stability, and homogeneity, each of which can impact the explorer stability, traction, and slippage. Collectively, these parameters characterize the relative ease of traversability of a terrain from which associated explorer traverse velocities and power requirements may be predicted. In this manner, the numerous local soil mechanics properties may be represented as a whole in terms of a single index denoting the overall effect these conditions have on a traversing explorer. This index value may then be interpreted within explorer cost functions to precisely represent any physical cost effects.

Pathmaster stores soil mechanics data as a matrix of such index values. Each data point may take a value of zero, one, or two. By default, all points on the terrain have a soil mechanics value of zero. When this layer is active, left-clicking on the surface will set all points within the edit rectangle to a value of one. Double-clicking will further set all applicable points to a value of two. Holding shift while clicking will reset the chosen points back to zero. When viewing the soil mechanics layer, all points with a value of zero will be grayed. Points with a value of one will clearly appear as semi-transparent maroon, and points with a value of two will be visible in sharp, dark maroon.

The soil mechanics index data used in Pathmaster is completely arbitrary on its own. It is up to the explorer cost functions to give meaning to these index values. For instance, a value of zero

could correlate to nominally easy terrain, a value of one represent moderately challenging terrain, and a value of two denote particularly difficult terrain requiring substantial additional time and energy to traverse. As another example, a value of zero could represent clear, hard terrain, a value of one indicate extensive scattered rocks, and a value of two indicate sand. Once such a scheme is defined, the explorer cost functions will explicitly determine the effect that these summarized characteristics have on traverses. While such modeling may seem rather crude, it enables planners to very quickly represent terrain conditions within a reasonable approximation.

4.3.6.3 SCIENTIFIC RETURN

Terrain scientific return refers to the relative interest of a region in terms of apparent potential scientific gain. Features such as craters or rilles as well as distinct characteristics including chemical composition or radioactivity can make certain areas of the terrain far more interesting than others. Exploration through these locations hence is preferential to travelling over more mundane territories. The overall desirability of traversing over a particular terrain region for scientific gain may be represented in terms of a single comparative index. In this manner, the relative levels of interest or priority of distinct terrain areas may be quickly established.

Pathmaster stores terrain scientific return data as a matrix of such index values in an identical manner as soil mechanics data. Each data point may take a value of zero, one, or two. By default, all points on the terrain have a scientific return value of zero. When this layer is active, left-clicking on sets all applicable values to one, double-clicking sets them to two, and holding Shift while clicking resets them to zero. In the display, all points with a value of zero will be grayed, points with a value of one will appear as semi-transparent purple, and points with a value of two will appear in deep purple.

The scientific return index data alone is also completely arbitrary, and these values must be interpreted by the explorer cost functions in order to establish meaning and effect within the mission. Again, while this may be a considerably limited modeling of complex and subjective information, it enables planners to very quickly identify and prioritize interesting areas within the terrain.

4.3.6.4 OPTIONAL ADDITIONAL PARAMETERS

In addition to the data layers presented, Pathmaster offers another layer which is not pre-defined. Denoted as "Other", this layer accommodates the incorporation of an additional parameter significant to the mission. It is again stored as a matrix of index values. Functionality is identical to soil mechanics and scientific return, with stored values of zero, one, or two appearing as gray, semi-transparent blue, and dark blue, respectively. These values may be incorporated into the explorer cost functions when establishing traverses. As an example of using this additional layer, assume that on a particular EVA a robot is constrained to remain within a certain distance of the traverse plan for a team of astronauts. This could be represented by highlighting all points within the given range of the astronaut path with a distinct "other" value. It would then be up to the robot cost functions to recognize this parameter and apply the stated constraint to all robot traverse paths (which in this case could be done in the same manner as applying obstacles).

4.3.7 ESTABLISHING OPTIMIZED TRAVERSE PATHS

Once all mission inputs have been entered, users may click on the "Run PATH" button in the Mission Planner GUI to generate explorer traverse paths. Here, Pathmaster goes to work determining specific costs for crossing the surface and establishing routes to destination sites. In all scenarios, Pathmaster assumes a single fixed mission objective of visiting every defined waypoint in order. The only internally applied operational constraints on traversals are the terrain obstacles. Assuming all waypoints are accessible (i.e. none are enclosed by obstacles, in which case a warning would be returned), traverse paths are calculated for each explorer until all waypoints have been visited, at which point the mission plan ends.

Within Pathmaster, therefore, explorer activity planning is limited by the exhaustion of objectives. As explained in Chapter 3, the function of EVA optimization in this case is to minimize the exploration cost of the mission. Thus, the goal of path planning is to automatically calculate routes of minimized cost for each explorer.

Pathmaster's general traverse plan functionality is based upon the Planetary Aid for Traversing Humans (PATH) software presented in Márquez, 2007. All explorer cost criteria and related functions derive directly from PATH. Pathmaster employs these costs within a novel

implementation of the A* ("A star") algorithm over the surface domain as demonstrated in Johnson, 2008. This routine establishes the optimized traverse routes and provides the associated explorer costs.

4.3.7.1 THE PATH SOFTWARE AND COST FUNCTIONS

PATH was developed in 2007, under a team led by Dr. Jessica Marquez at MIT. The purpose of this software was to investigate how humans collaborate with automated support, specifically applied to the task of optimal EVA traverse path planning for an astronaut on the moon. In order to compare paths, the PATH team established a set of functions to estimate the distance, required time, and metabolic cost of each traverse. These cost values were calculated on an incremental basis of moving from a single data point on the terrain model to an adjacent point. All explorers in Pathmaster currently assume this same model, intended to characterize a suited astronaut on foot, pending further development of rover and robot specific cost functions.

Distance Cost

The first traverse cost found is distance, which is based upon the physical length between data points on the map. This length is determined by the map resolution and the direction of travel. Lateral motion between data points has a distance of the resolution, while diagonal motion distance is greater by a factor of the square root of two. Overall path distance is minimized by straight-line travel.

Time Cost

The second traverse cost found is the time required to travel from one location to another. This is based upon both the distance, already calculated, and the surface slope. The local slope between points, given in degrees, is determined trigonometrically as the arctangent of elevation differential over distance. This slope value is fed into a model which gives the predicted traverse velocity, as shown in Table 4.2 and Figure 4.13. By this model, maximum explorer velocity is 1.6 meters per second, which occurs on flat terrain, and going downhill is faster than going uphill. Required time is finally calculated as the quotient of distance over velocity.

Table 4.2 Estimated explorer velocities as a function of surface slope, from Márquez, 2007

Slope, α	Velocity (m/s)
$-20° \leq \alpha < -10°$	$0.095 \bullet \alpha + 1.95$
$-10° \leq \alpha < 0°$	$0.06 \bullet \alpha + 1.6$
$0° \leq \alpha < 6°$	$-0.02 \bullet \alpha + 1.6$
$6° \leq \alpha < 15°$	$-0.039 \bullet \alpha + 0.634$
$\alpha < -20°, \alpha > 15°$	0.05

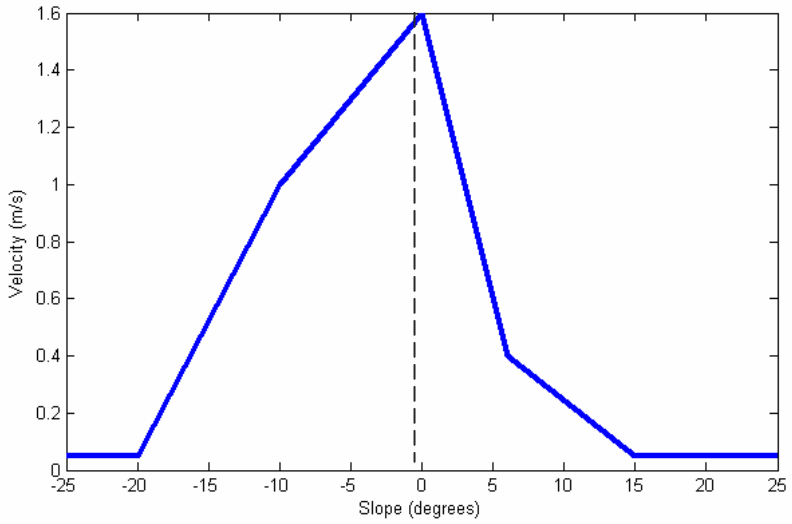


Figure 4.13 Explorer velocity profile as a function of slope (modified from Márquez, 2007)

Metabolic Cost

The final traverse cost found in Pathmaster is the metabolic expenditure, or energy consumed, by a traversing explorer. This calculation builds upon the preceding functions, and is dependent upon traverse time, surface slope, explorer velocity, explorer mass, and gravity. The model employed was developed by Santee et al. (2001), and gives the energy consumption rate of an explorer as they cross the surface. The formulation, shown in Table 4.3 and Figure 4.14, is well suited for approximating extra-terrestrial EVA conditions as it incorporates surface slopes and explorer velocities along with planetary gravity (Márquez, 2007). In this model, energy rates are broken up into the base energy required to move forward along with the additional energy required to move uphill or downhill. High energy rates are incurred for uphill travel, while there are minimal energetic penalties for going downhill. Metabolic cost is ultimately calculated as the product of energy rate and required time.

Table 4.3 Estimated explorer energy consumption rates, from Santee et al., 2001

Energy rate (J/s) = W_{level} + W_{slope}	
$W_{level} = [3.28 \cdot m + 71.1] \cdot [0.661 \cdot v \cdot \cos(\alpha) + 0.115]$	
Slope, α	**W_{slope} (J/s)**
$\alpha = 0°$	0
$\alpha > 0°$	$3.5 \cdot m \cdot g \cdot v \cdot \sin(\alpha)$
$\alpha < 0°$	$2.4 \cdot m \cdot g \cdot v \cdot \sin(\alpha) \cdot 0.3^{|\alpha|/7.65}$

Where m is explorer mass, g is gravity, and v is explorer velocity

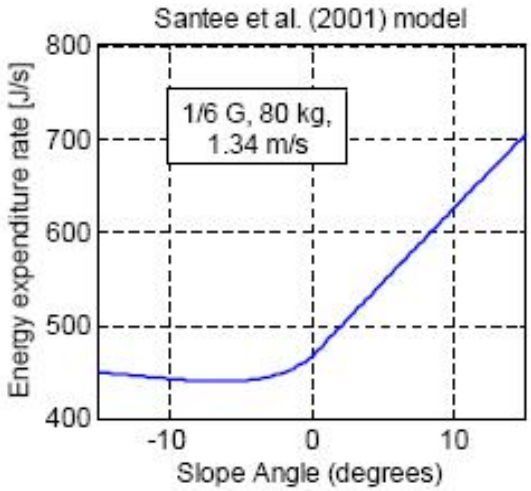


Figure 4.14 Explorer energy consumption rates, shown for lunar gravity (Márquez, 2007)

4.3.7.2 MATLAB IMPLEMENTATION OF A*

With the cost profile fully determined for any potential travel between adjacent points on the terrain map, a graph search algorithm may be utilized to identify the desired route of minimal cumulative cost from a starting point to a goal point. To begin, each data point on the terrain map is represented as a node with edges connecting to all neighboring nodes both laterally and diagonally. The cost of crossing each edge is then assigned with a specific desired quantity from the cost profile. Presently, Pathmaster operates upon the metabolic cost of traverses and hence searches for paths of minimized explorer energy expenditure.

Pathmaster's traverse optimization routine implements the A* algorithm, first described by Hart et. al (1968), chosen for its fast computation speed and completeness without sacrificing accuracy. Beginning at a waypoint, the direct cost for traversing to a neighboring node is coupled with a heuristic estimation of the cost to travel from that node all the way to a goal point, i.e. the next waypoint. The heuristic assumes the best-case scenario of straight line travel over flat terrain, and hence is admissible since it will never overestimate actual traverse costs (Johnson, 2008). The algorithm tests all possible neighboring nodes and proceeds to the one with the lowest collective direct and heuristic cost. The process then repeats itself from that node, incorporating the cumulative direct cost to travel from the starting point to all new neighboring nodes along with heuristic estimations to the goal, meanwhile still considering any previously searched nodes.

With every iteration, the algorithm proceeds to the successive "best" node with the lowest running cost, keeping track of which nodes were visited along the way there. In this way, the routine is known as a "best-first" search. Most importantly, this functionality ensures that the first time a new node is visited implicitly comes via the best possible route to that node. In other words, when the algorithm proceeds to a new node, the series of nodes from the start leading to that point represents an optimized route. If there were a better (least costly) way of getting there, it would have already been established earlier due to best-first searching. Hence, as soon as the algorithm first arrives at a goal waypoint, the optimal route to that waypoint has been established. The resulting path is represented as the series of connecting nodes from start to finish.

Another important feature of Pathmaster's optimization routine is the incorporation of bi-directional searching. Instead of solely examining nodes branching out from the starting point, simultaneous searches are performed from both the current starting and destination waypoints. Once the search paths first meet, the optimal route is established again according to the best-first principle. Using this method, significantly fewer nodes must generally be searched to arrive at the optimized path. Hence, computation time is reduced to further facilitate real-time planning.

When three or more waypoints are defined, Pathmaster further invokes a "smart searching" algorithm that augments the benefit of bi-directional searching and reduces redundancy. Along a path, the goal waypoint for one traverse segment becomes the starting waypoint in the next leg. Pathmaster recognizes this, maintaining all search data stemming from the goal of a search and automatically applying that work toward searching for the next waypoint. Figure 4.15 illustrates this functionality. On the left, a third waypoint is already within the nodes visited from the goal of a previous search, waypoint two. Hence, with no additional work, the optimized path from waypoint 2 to 3 has already been established. In the more common case where a third waypoint is outside of the visited region, Pathmaster builds upon the nodes already scanned and only a limited set of additional nodes, represented in darker grey, must be searched in order to generate the path. This process iterates for all successive waypoints.

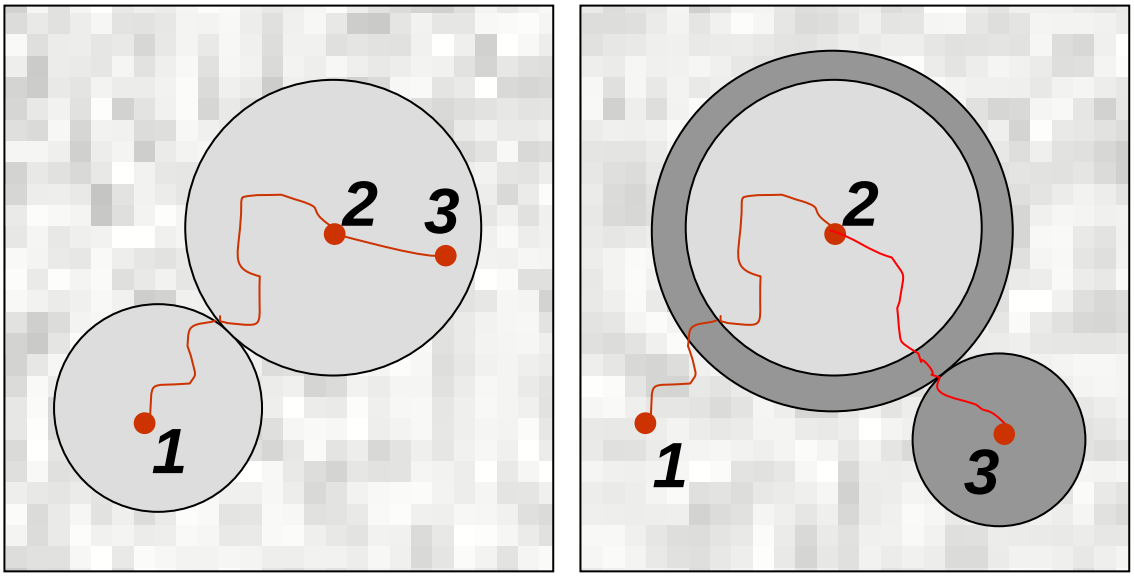


Figure 4.15 Visualization of "smart searching" for a third waypoint (modified from Johnson, 2008)

The entire path search routine must operate within our problem domain to respect any defined terrain obstacles. Fortunately, A* is easily adaptable to incorporate non-traversable areas simply by setting the cost of crossing edges connecting to such nodes infinitely high. Better yet, Pathmaster outright ignores these edges and operates as if they don't exist. Hence, obstacles effectively alter the makeup of the map representation by removing all corresponding graph edges. This assures that the search algorithm finds the optimal route incorporating and necessarily navigating around all obstacles.

Once an optimized path has been established, Pathmaster finishes by smoothing the set of route nodes into best-fit representative line segments via the midpoint line algorithm (Bresenham, 1965). Each path is hence ultimately stored as a set of line segment endpoint coordinates. The cumulative distance, time, and metabolic costs from the start to each segment point are calculated and recorded as well.

4.3.8 TRAVERSE PATH AND COST DISPLAY

As soon as optimized traverse paths are created, the smoothed line segments are clearly overlaid on the terrain display in representative explorer colors, with waypoints highlighted in green (Figure 4.16). This display represents the completed mission plan. The estimated total costs of a selected traverse are displayed in the appropriate fields in the Mission Planner GUI menu. Users may choose explorer paths either with the explorer drop-down menu or by right-clicking directly on the desired route. Distance costs are shown in the currently selected axes scale units. Metabolic cost is displayed in the center, and associated units of kilocalories, BTU, or kilojoules may be selected with the neighboring drop-down menu. Lastly, estimated time is displayed in hour and minute format.

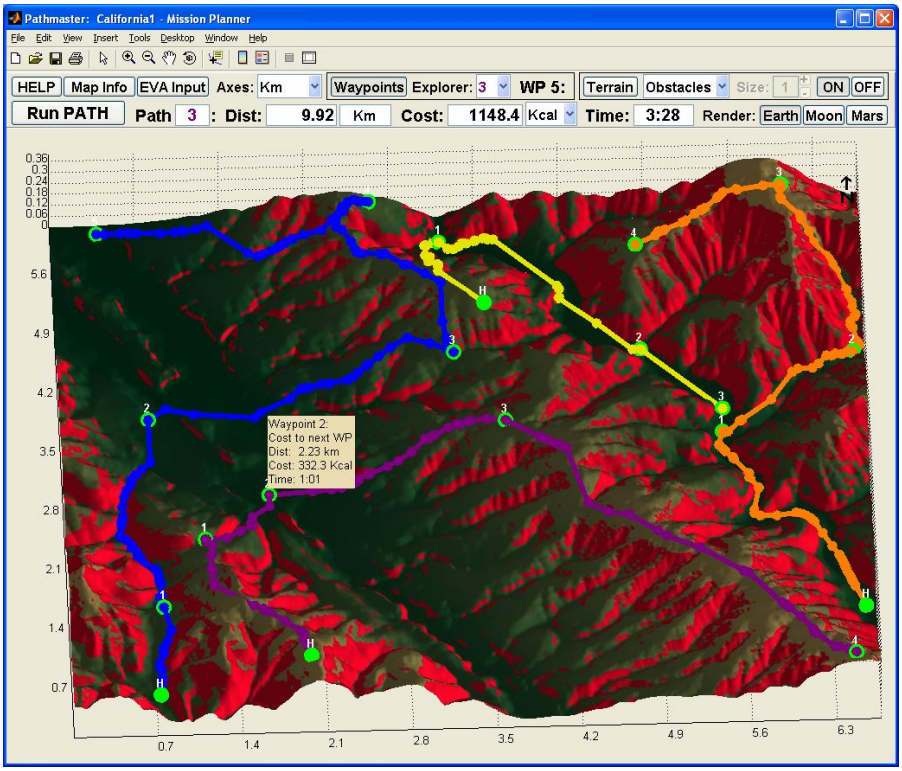


Figure 4.16 Traverse path and cost display

In addition to the total traverse costs, users may display detailed information for each leg between waypoints. Right-clicking on a path will open a data display at the nearest waypoint, as shown in Figure 4.16. Successive right-clicking on the path or display allows the user to cycle through various information of interest to planners, including the cost from start, cost from the previous waypoint, cost to the next waypoint, cost to end, and the local terrain information as given with the terrain data display.

4.3.9 SIMULTANEOUS MISSION SCENARIOS

Multiple instances of Pathmaster may be run simultaneously on a single machine. With Pathmaster already open, users can simply re-enter the "pathmaster" command in the MATLAB main prompt and a new instance will run completely independently of the mission already open. This enables great flexibility for users to quickly evaluate various potential mission scenarios with distinct situations as well as operate multiple explorers with unique parameters. Such functionality facilitates manual optimization of overall mission strategies beyond simply optimizing activity within a single scenario.

4.3.9.1 SIDE-BY-SIDE COMPARISON

By loading differing mission situations into separate instances of Pathmaster, various mission plans may be placed literally side-by-side on a controller's computer screen. This empowers a direct comparison of all mission routes and costs such that a most desirable option may be determined. For example, Figure 4.17 depicts the side-by-side evaluation of two strategies for an EVA involving two astronauts. On the left, explorers travel together for the duration of the sortie. To the right, the astronauts split objectives and proceed alone. As seen in the menu displays, the predicted EVA costs for Explorer 1 decrease by less than 15% with the divide and conquer approach compared to staying together. This somewhat modest cost savings may not offset the likely increased risk of sending astronauts out alone. In this case, planners could soundly decide upon keeping the astronauts together based upon this strategic comparison in Pathmaster.

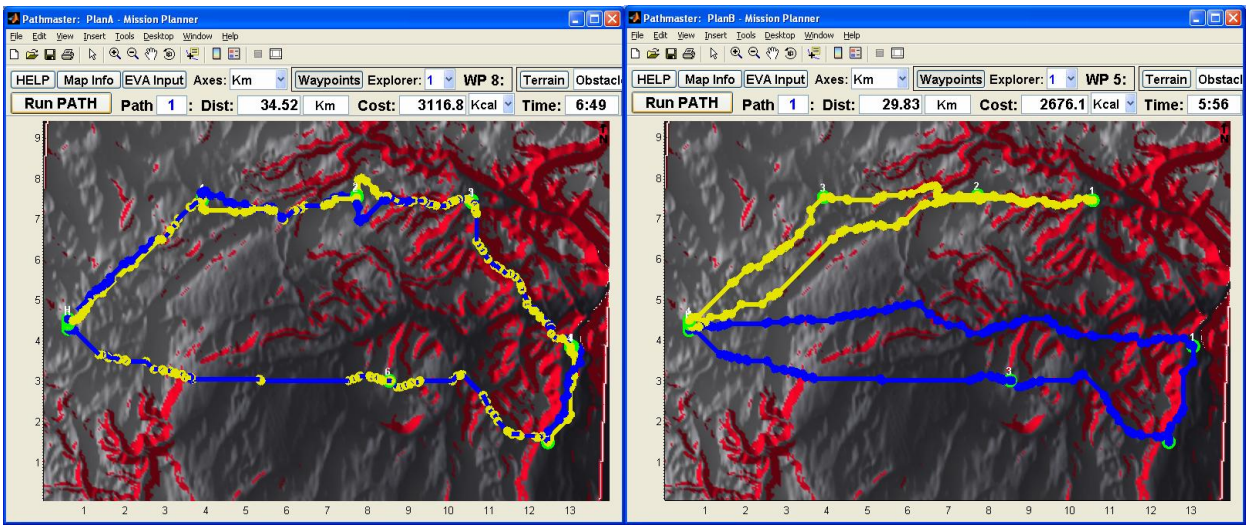


Figure 4.17 Side-by-side comparison of two EVA strategies
At left, explorers travel together; at right, explorers divide and conquer

4.3.9.2 EXPLORERS WITH DISTINCT PARAMETERS

Within a single instance of Pathmaster, all explorers share many common environmental factors including gravity, sun lighting, obstacles, soil mechanics, scientific return, etc. If a certain explorer faces different parameters on a mission than other members, distinct modeling may be accomplished through a separate instance of Pathmaster. The most common example would be explorers with differing obstacles, such as astronauts and small robots. In this case, astronaut obstacles and traverse plans can be developed in a separate window from the differing robot obstacles and subsequent plans. This general strategy may be applied to all other environment parameters as well. As another important example, consider explorers operating at different times, hence with differing lighting conditions. Here, all corresponding data could simply be entered in separate EVA Input menus to model distinct points in time during the mission. Taking advantage of such diverse modeling capabilities, Pathmaster is able to handle complex missions with large teams of differing explorers.

4.4 VIRTUAL REALITY SIMULATION

Aside from internal mission plan development and display, Pathmaster also provides output text files to the "Render" directory that can feed an external virtual reality simulation. The Astronaut Rover Mission Simulator (ARMS), written in C++ and under development by Uday Kumar at

Arizona State University, cooperates with Pathmaster to provide a fully interactive 3D environment for mission simulation (Figure 4.18). ARMS incorporates a mobile astronaut and MER-style robot on a scaled virtual rendering of the physical terrain, offering a realistic surface level experience. The explorers function independently, and may be controlled remotely within the environment. Waypoint and traverse path coordinates can be loaded from Pathmaster and clearly displayed within ARMS in real-time. This system enables teams to run an entire virtual simulation of a mission ahead of time, which facilitates preliminary evaluation of activity scheduling and strategies, practicing of missions, and even a general familiarity with terrain features and objectives before ever stepping foot on the surface.

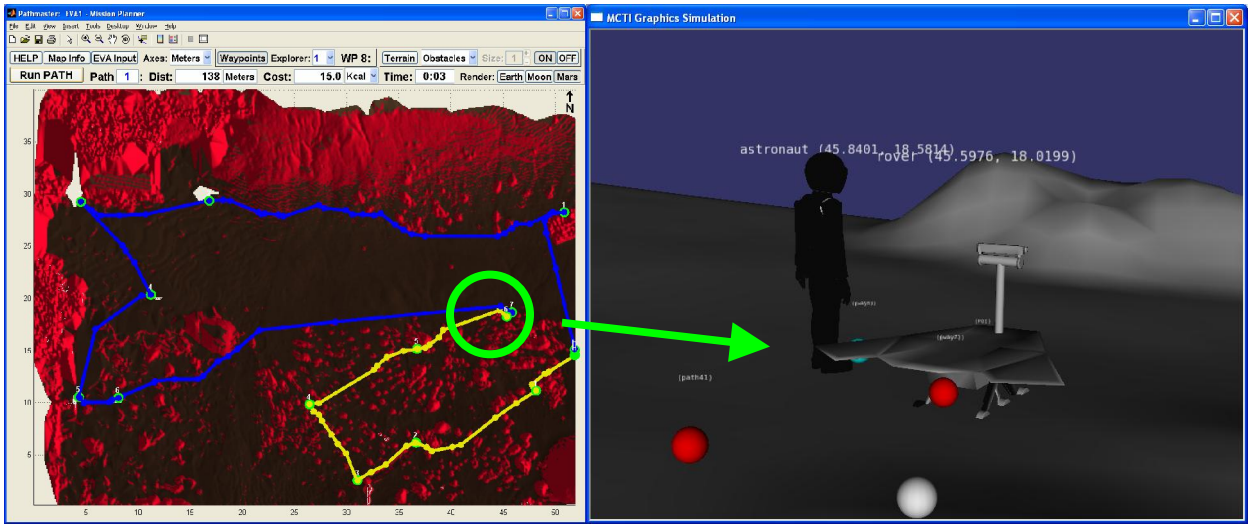


Figure 4.18 Running a Pathmaster developed mission (left) in the Astronaut Rover Mission Simulator (right)

4.5 REAL-TIME MISSION SUPPORT

As operations shift to real-time during a mission, the function of a support system becomes the familiar cyclic pattern of sending out mission information, assisting explorers in following the plan, responding to changing situations, and updating the plan when necessary. For Pathmaster, this translates to passing mission plans to all parties, assisting in explorer navigation, and enabling near real-time mission re-planning. The program currently offers varying degrees of capability in each of these areas.

4.5.1 RELAYING MISSION INFORMATION

Pathmaster offers three general methods for relaying mission information to other systems, locations, and explorers. First, comprehensive output files are automatically written to feed any cooperating applications on the same machine, such as an external rendering system like ARMS. Second, to relay information to other locations capable of running Pathmaster, a MATLAB data file containing all mission information is always written to the directory containing the Pathmaster m-file. This data file may be directly transmitted to all desired sites (up to now, these files have been routinely e-mailed). By choosing the transmitted file when opening Pathmaster and selecting the options to load all existing data (waypoints, obstacles, etc.), every controller will share the same information. Clicking on "Run PATH" in the Mission Planner GUI further generates the mutually identical mission plan. Lastly, to relay information to field explorers or any other site with limited computing capabilities, an image (i.e. screenshot) of the terrain model detailing waypoints and traverse paths may be sent. This serves as an overview of the mission plan as well as map which explorers may follow.

4.5.2 EXPLORER NAVIGATION

The mission plan image available from Pathmaster, while useful as a summary or reference, alone is insufficient to accurately guide explorers over unfamiliar terrain to destination sites as has been explained. Instead, an interactive display capturing explorer position and motion in relation to a planned traverse is necessary. Although Pathmaster currently has no such capabilities, certain compatible systems may be employed to achieve this desired navigation support.

4.5.2.1 GPS LINK VIA ARCGIS

Shortly before development on Pathmaster began, a related mission planner system was completed by Lindqvist (2008). This system, also based upon PATH, operates within ArcGIS. The interface enables familiar, though limited, functionality including terrain map display, calculation of obstacles based upon surface slopes, point-and-click waypoint addition for a single explorer, and finally calculation of optimized traverse routes via a direct call to the PATH Java software (Figure 4.19). Pathmaster was designed to be compatible with the ArcGIS mission planner. Specifically, all output map text files, originally intended to be loaded in Java when

calling PATH, may also be directly loaded into Pathmaster. In fact, this is the general method by which new terrain maps have been created for Pathmaster.

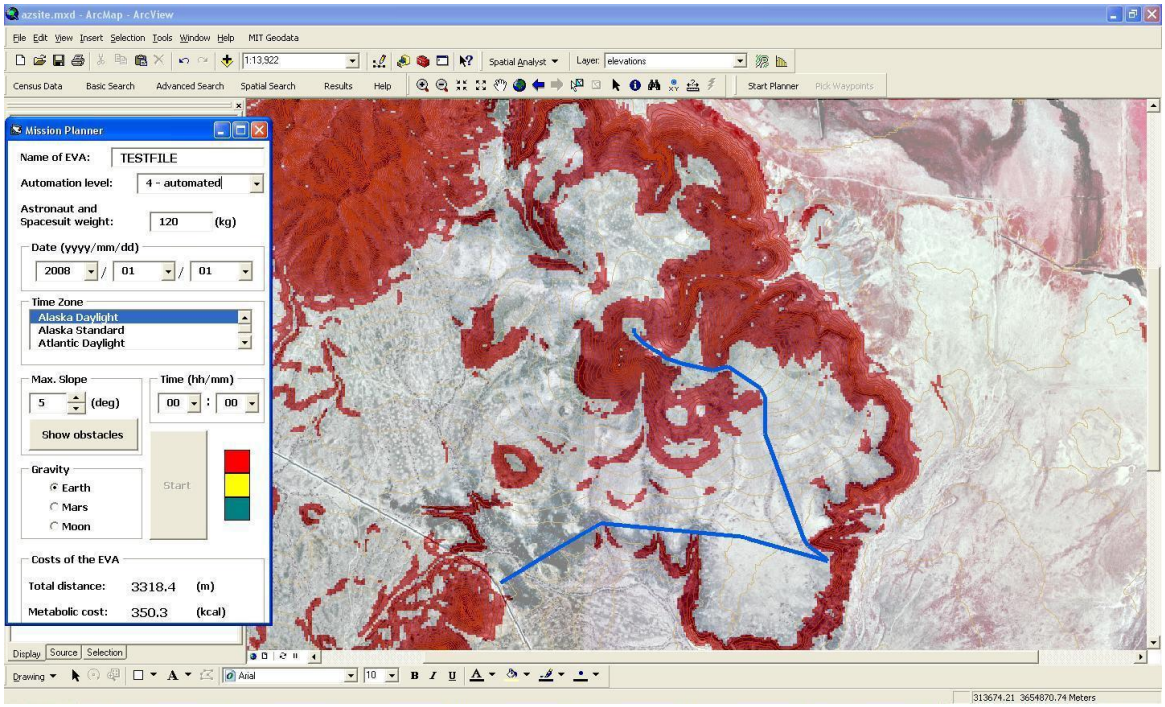


Figure 4.19 Mission planner configuration in ArcGIS (Lindqvist, 2008)

By loading the corresponding terrain into ArcGIS, mission waypoints may be manually entered to match a plan developed in Pathmaster. Clicking "Start" in the ArcGIS mission planner GUI will run PATH and display the corresponding optimized route on the map.

From here, the advantage of using ArcGIS is that instead of a simple screenshot, the terrain view with overlaid traverse path may be exported as a spatially referenced image by creating a "world file". This image and accompanying world file can be loaded directly into a field computer with a GPS receiver, where the terrain display orientation will be automatically recognized in terms of corresponding global position. In this manner, the GPS receiver may display current explorer position along the loaded image. As established in Chapter 3, by ensuring that the displayed position at all times coincides with the drawn traverse route while crossing the surface, an explorer physically follows the planned optimal path (see Figure 3.6). Such a GPS link not only provides extremely intuitive real-time navigation support, but is also very practical since field explorers need only to upload an updated image and world file to follow a new mission plan.

4.5.2.2 VIRTUAL REALITY DISPLAY

Although still under development, ARMS or a similar system can potentially provide greatly enhanced navigation support over the aerial view display offered by typical GPS receivers. Equipped as a heads-up display with motion capture, such a system would offer explorers a real-time 3D view of the virtual terrain with clearly highlighted traverse paths and waypoints to seamlessly guide them as they cross the physical terrain (see Figure 3.8).

4.5.3 MISSION RE-PLANNING

When unexpected EVA situations arise to warrant an operational response, mission information must be updated accordingly. Pathmaster is designed to be well suited for this task, enabling streamlined modification of mission models and creation of new plans. The general process by which a mission is modified is the same as planning the original mission. Once edits have been made, simply clicking the "Run PATH" button again will generate a new optimized mission plan incorporating the latest information. This updated plan can then be distributed through the channels identified earlier.

4.5.3.1 UPDATING MODELS AND CONTINGENCIES

As feedback from the surface team arrives, any necessary updates to the mission models may be made in the Mission Planner GUI via the waypoint and terrain edit controls presented earlier. Editing an explorer's waypoints will clear an existing traverse path, if any, while editing terrain parameters will clear all traverse paths. Additional EVA parameters may be modified by clicking on the "Map Info" or "EVA Input" buttons in the menu at the top (Figure 4.20). This re-opens the respective menus, and the desired data fields may be freely altered. Any changes will be automatically incorporated when the Mission Planner GUI reopens.

Figure 4.20 Map Info and EVA Input buttons to re-open the respective menus

Terrain edit mode allows controllers to modify the terrain models directly in real-time with new information, while waypoint edit mode allows controllers to freely clear waypoints and establish new sites in seconds. Taking advantage of these capabilities, any contingency situations and associated responses may be quickly modeled within Pathmaster. In particular, by modifying waypoints controllers can immediately instruct explorers to move to urgent sites such as shelter or the location of an ailing team member. For making high-level operational decisions in response to contingencies, features such as the side-by-side scenario comparison may be used to quickly evaluate potential courses of action.

4.5.3.2 RETURN HOME PATHS

Pathmaster offers a specialized built-in contingency response: the "return home" path feature. At any point along a traverse, an explorer may be directed to immediately return to the starting base, or home. Perhaps the most recognizable example where this feature is well suited would be a walk-back situation. Return home paths are found by holding Shift while clicking along a traverse path. The location clicked on is assumed as the point at which the explorer begins the return, and an optimized route directly back to the starting point is automatically found. These special traverses appear as dotted paths along the terrain (Figure 4.21). Associated costs will be displayed in the menu at the top, and these paths may be selected and evaluated by right-clicking as with any other path.

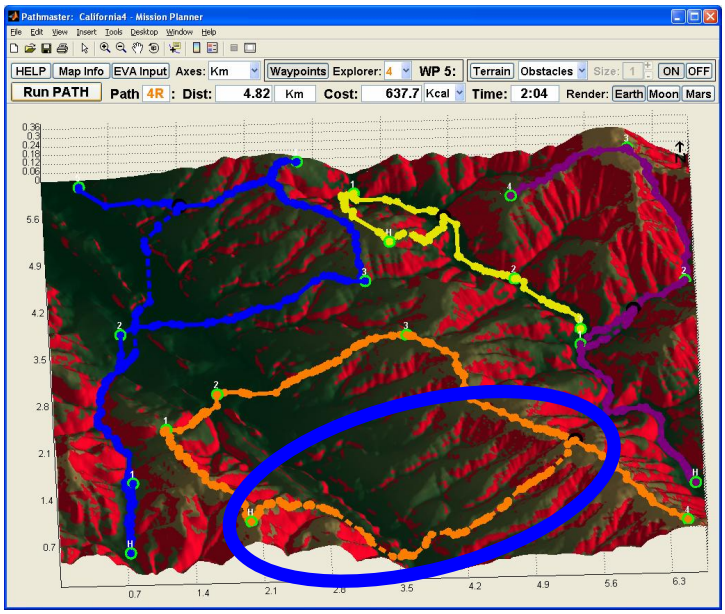


Figure 4.21 Return home paths, shown as dotted routes

4.6 ADDITIONAL FEATURES

There are two final features of Pathmaster worth noting.

4.6.1 'LITE' OPTION

Pathmaster may be called directly from the command line with the 'lite' option, entered as:

```
>> pathmaster('lite')
```

This invokes a simpler surface rendering, as shown in Figure 4.22. Use of this option speeds plotting time and prevents problems on some machines. It is well suited for cases with limited computing power. Aside from the terrain appearance, all mission planning functionality is fully maintained. If a machine encounters problems with Pathmaster terrain renderings, use of the 'lite' option is recommended.

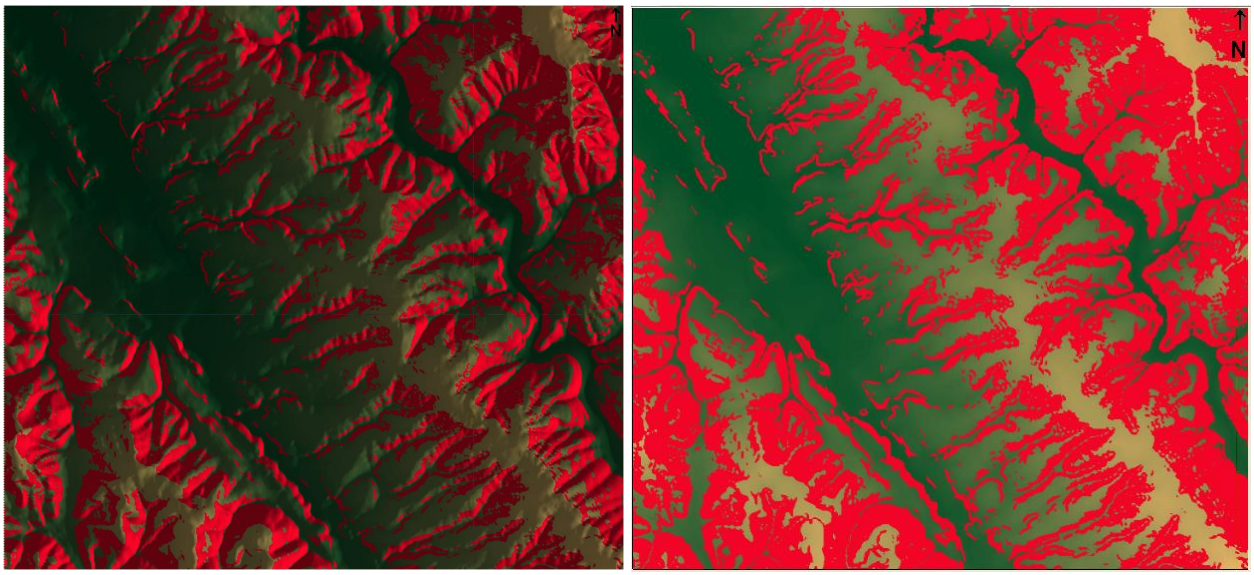


Figure 4.22 Normal surface rendering (left) and 'lite' rendering (right)

4.6.2 RELOADING MISSION INFORMATION

Though briefly mentioned before, one last feature is worth highlighting. Each time a mission is run in Pathmaster, a MATLAB data file sharing the given name entered in the EVA Input menu is written to the directory containing the Pathmaster m-file. This file holds all stored mission information, which includes the elevation map, terrain parameter data, and any waypoints. In order to re-load such a previous mission or terrain, simply select this corresponding file when

opening Pathmaster. The options to use the data within this file will appear as check-boxes near the top of the EVA Input menu (to load waypoints, all desired explorers must first be added within this menu). This functionality greatly facilitates creating multiple related mission scenarios since a common base situation can be mutually loaded. The user should take note though to rename each successive scenario so that previous files will not be overwritten.

5 FIELD TESTING

5.1 TRAVERSE PLANNING AND GPS-LINKED NAVIGATION

The first test of mission support features in a real traverse situation was performed on the MIT campus during December of 2007. The purpose of this experiment was to test the efficacy of using a support system for optimal traverse planning and subsequent real-time explorer navigation using GPS.

5.1.1 SETUP

This experiment employed the mission planner system developed in ArcGIS by Lindqvist (2008), presented in the previous chapter. To begin, a map of the MIT campus was loaded into ArcGIS and the view zoomed to cover the general area including the main entrance at 77 Massachusetts Avenue, Kresge Auditorium, and the corner of Massachusetts Avenue and Memorial Drive (Figure 5.1). This was chosen as the terrain to be explored.

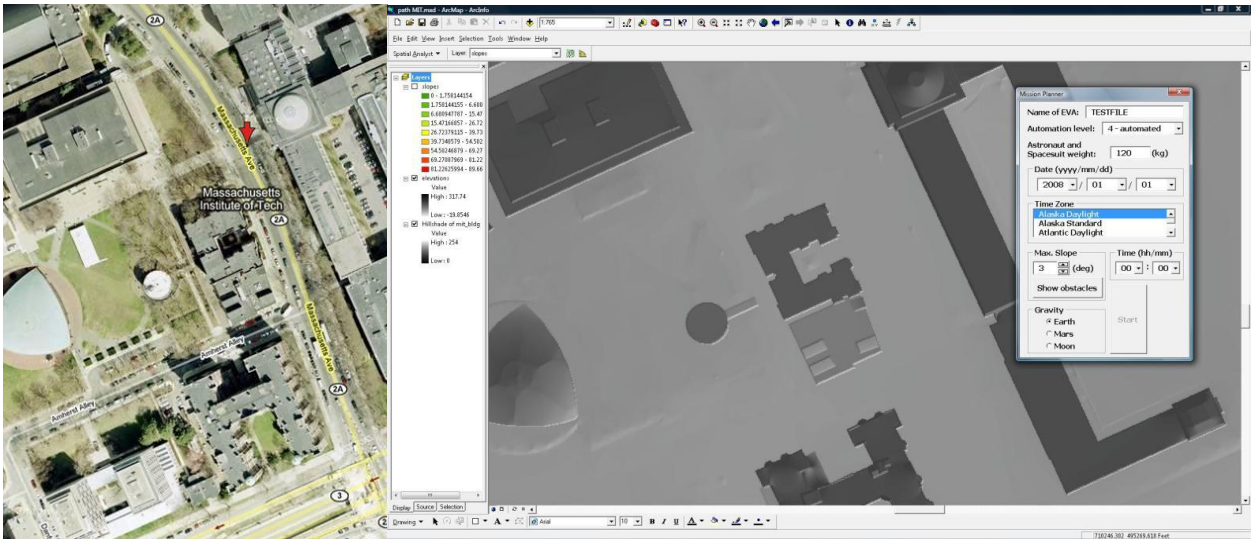


Figure 5.1 Aerial photograph of the MIT campus (left, courtesy Google Maps), and the corresponding terrain model loaded in ArcGIS (right).

Next, terrain obstacles were set for all areas with a surface slope greater than three degrees. Though this is a very low limit for the maximum traversable slope, this value was chosen so that

a significant amount of obstacles would be presented. The selected terrain is rather level and normally would be quite mundane for navigation. For testing purposes, the low slope threshold was thought appropriate to present some challenge to the system. The resulting terrain obstacles are depicted in Figure 5.2.

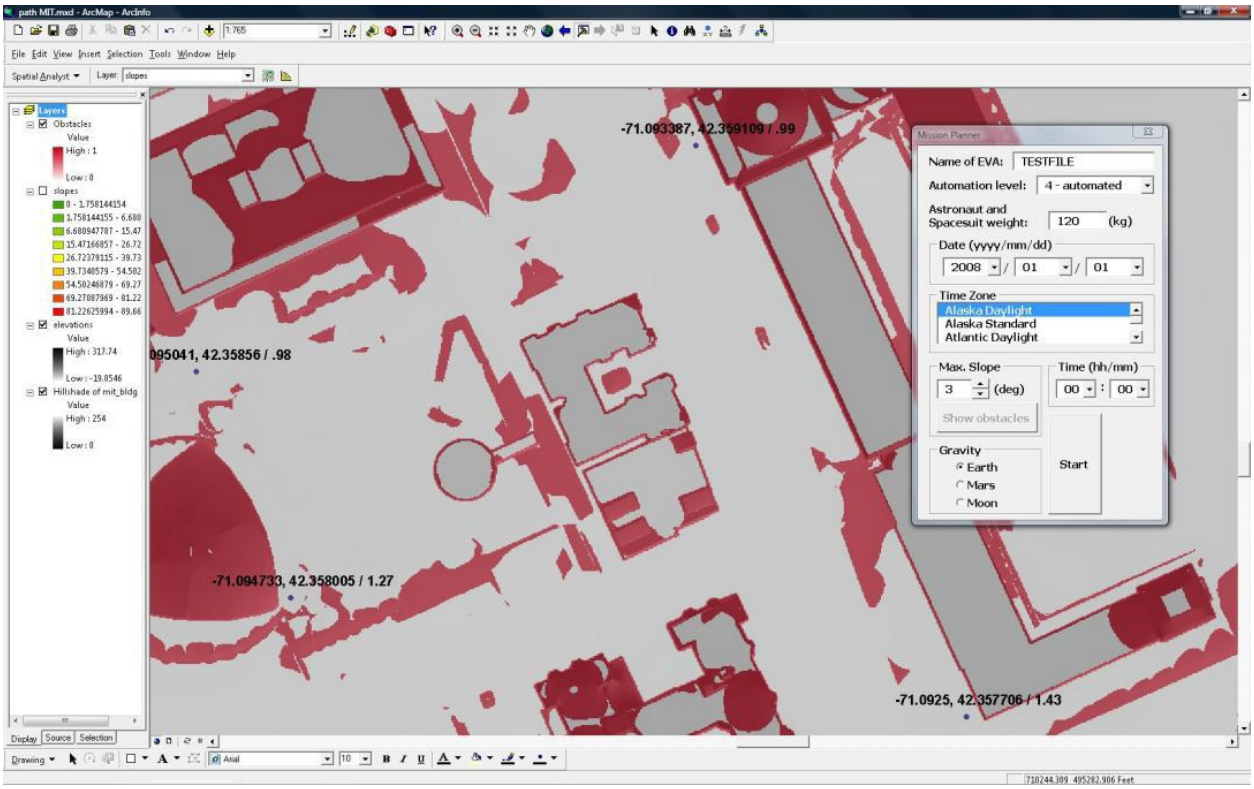


Figure 5.2 Terrain obstacles, shown in red, and mission waypoints

Four mission waypoints were then established, also shown in Figure 5.2. The first waypoint, or starting point, was set at the main MIT entrance. The second and third were set just north and southeast of the Kresge auditorium, respectively. The final waypoint was set at the northeast corner of Massachusetts Avenue and Memorial Drive. The Planetary Aid for Traversing Humans (PATH) software, presented in Chapter 4, was then called to find the optimal routes between these waypoints. The resulting planned mission path was plotted along the terrain, as shown in Figure 5.3.

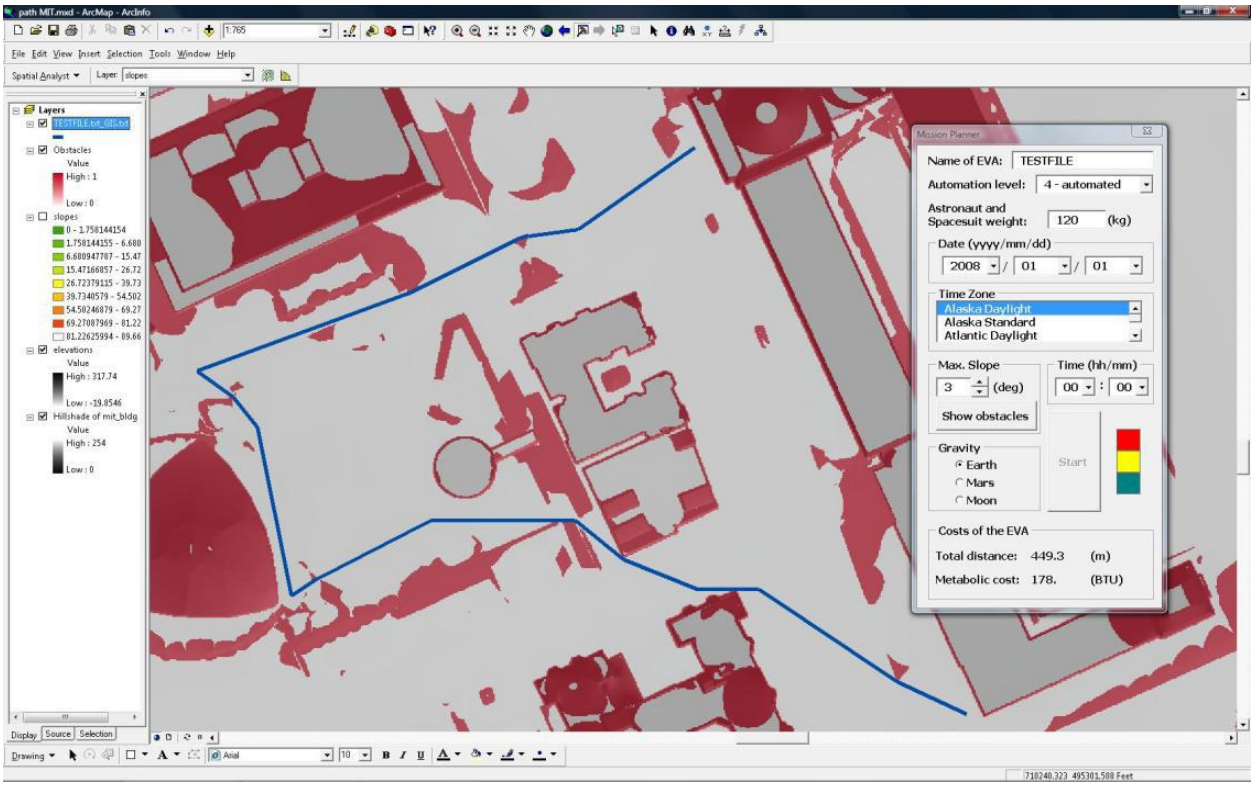


Figure 5.3 Planned traverse route for the field test

Finally, the terrain image with the traverse path, which represented the mission plan, was exported to a handheld computer with a GPS receiver via the procedure described in the previous chapter. The field unit used was the Trimble Juno ST, shown in Figure 5.4

Figure 5.4 The Trimble GPS receiver used in the field (courtesy Lindqvist, 2008)

5.1.2 OPERATION

A team of four "astronauts", including the author and Lindqvist, embarked on foot from the starting point outside the main MIT entrance and attempted to follow the planned route as closely as possible. Real-time guidance was provided by the Trimble, which animated the current crew position along the mission plan image. The team traversed to each waypoint location, and the mission came to an end upon arriving at the final planned road intersection. Figure 5.5 summarizes the execution of the mission.

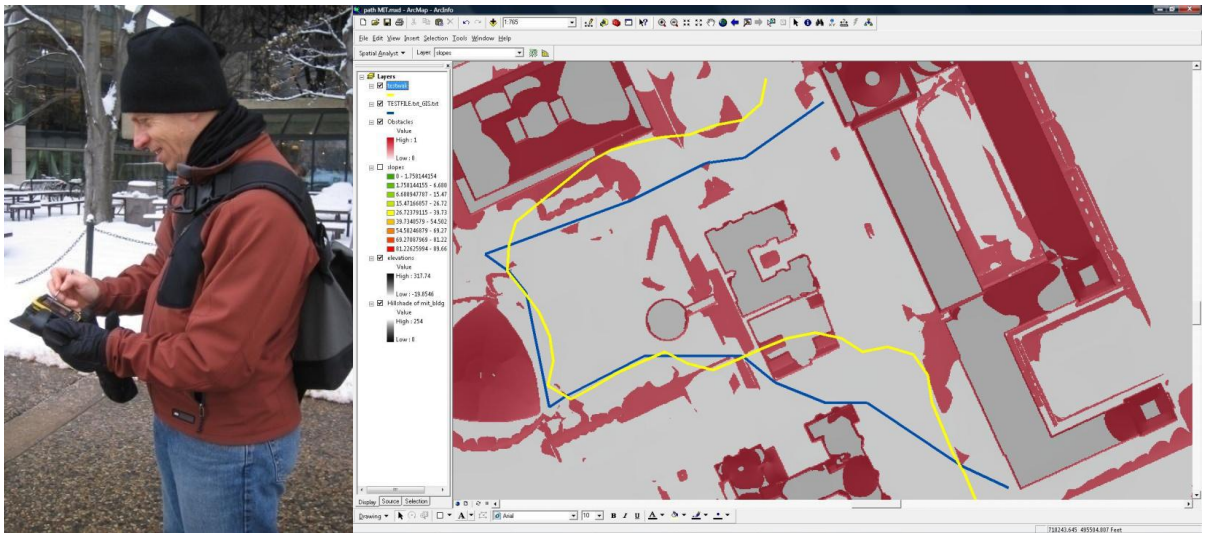


Figure 5.5 Mission plan execution. At left, a crewmember operates the Trimble unit for guidance. The planned (blue) and executed (yellow) routes are shown to the right.

5.1.3 CONCLUSIONS

The tested system was successful in planning a traverse and providing the corresponding information to the surface team. All software and hardware components worked as intended. During the real-time mission execution, however, several shortcomings became apparent.

The most recognizable problem is clearly seen in Figure 5.5, where the executed route apparently crosses several obstacles and even passes through a building. This was clearly not the physical route taken. Instead, these discrepancies are due to complications with the GPS receiver resulting in offsets or jumps in the read positional values. This problem, most likely caused by campus buildings interfering with and reflecting the satellite signal, was an unavoidable product of the chosen terrain. Along with people, trees, cars, traffic lights, etc., the surroundings for this field

test included many hindrances affecting the executed path that would not be present during a remote sortie, especially on another planet. For the path segments where these difficulties were less an issue, such as rounding Kresge and proceeding east, the explorers were able to follow the path relatively well. The lesson here is that the location of future tests should be selected with more thought so as to resemble conditions on the moon or Mars.

With that said, unexpected hindrances cannot be ruled out when providing real-time support. Lindqvist's account (2008) of this field test reveals another shortfall: "As there was a lot of snow on the ground and some fences that the original map did not include, the route could not have been followed precisely…" Due to unexpected obstacles not represented in the mission plan model, the team was forced to respond and re-plan in order to continue the mission. The system offered no direct support here, and the explorers were left to cope on their own. While only a minor issue here given the familiarity and small scale of the terrain, such unforeseen situations could pose a significant problem in a more hostile environment. As stated before, developing an optimal plan is irrelevant if the field explorers are unable to follow it. This simple test uncovers the great potential utility in being able to quickly update mission models and develop new optimal plans accordingly on more complex missions.

5.2 FUNDAMENTALS OF ENGINEERING EXPLORATION LAB

The next test of mission support features involved a complete geological style EVA simulation at MIT during February of 2008. The experiment was performed as the first laboratory exercise for the Fundamentals of Engineering course, a freshmen level introductory subject. It was carried out entirely by students under the supervision of the author. Aside from the educational objectives of the experiment, the purpose of this simulation was to test the feasibility of employing a support system in a mission control setting to aid in strategic EVA operation and decision making. In particular, the real-time performance of mission planning, surface team audio and video feedback, and explorer energetic monitoring systems was examined.

5.2.1 SETUP

A detailed explanation of the simulated EVA procedures and instructions is given in Appendix D. In general, the class was broken up into two teams: the surface team and mission control. The

surface team consisted of a group of “astronauts” and three distinct “rovers” (generally referred to as robots elsewhere in this work) with differing capabilities. In turn, mission control consisted of a director, communicator, positioning officer, medical officer, rover technician, and geologist. The terrain to be explored was chosen as Killian Court, while mission control was situated in a remote conference room.

The surface team was sent out into Killian Court. Astronauts, dressed in mock-up spacesuits, were restricted to remain together at all times. They were provided an audio link to the communicator at mission control via a walkie-talkie. Rovers, on the other hand, were allowed to travel alone. Due to logistics, they were remotely controlled by humans in the field; however, these controllers did not directly perform any mission activities. The rovers, shown in Figure 5.6, were equipped with wireless cameras that provided video feeds to the rover technician at mission control, and the controllers could receive verbal commands from the communicator via walkie-talkie.

Figure 5.6 The various surface team rovers

The Killian terrain was segmented into three distinct zones, each containing two pre-defined sites of interest (Figure 5.7). At each site were various “samples” (blocks, balls, etc.) which could be collected, but only by astronauts who were each permitted to carry one sample at a time. The geologist at mission control was provided information to determine which samples were deemed “interesting”. Not all sites necessarily contained interesting samples. Explorers began at the base location. The mission objectives, listed in order of priority, were:

1) Safely return all astronauts and rovers to base
2) Collect a sample of interest from as many zones as possible
3) Collect as many samples of interest as possible

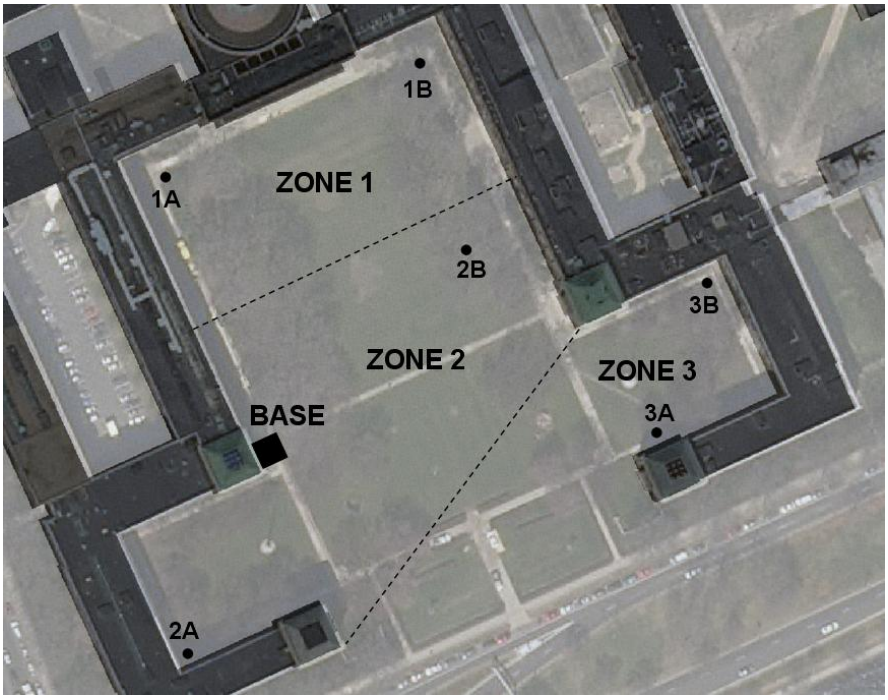


Figure 5.7 Aerial map of the Killian terrain denoting zones and sites of interest

Constraints on the surface team activities were simulated as limited oxygen supplies for astronauts and limited battery power for rovers. These levels were monitored at mission control by the medical officer and rover technician through interfaces designed in LabVIEW, as shown in Figure 5.8 and detailed in Appendix E. The control team members were responsible for keeping track of the real-time explorer activity and selecting the corresponding option within each interface. Simple models then determined the respective oxygen and electrical consumption rates based upon the activity. These systems essentially mimicked such explorer signals as may be routinely gathered through wearable sensors or onboard diagnostics.

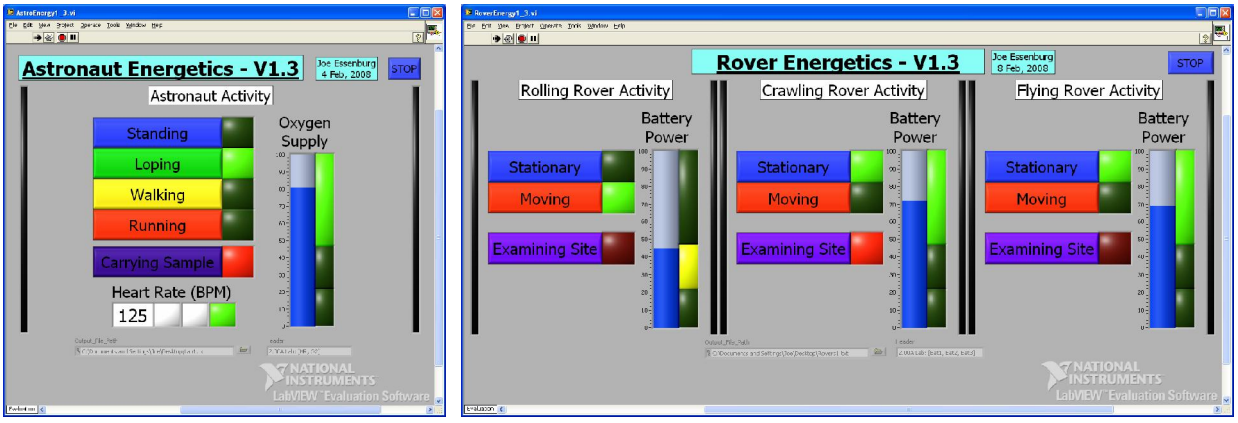


Figure 5.8 Astronaut and rover energetics interfaces

To test the capability of transmitting information to an alternate mission control site, the astronaut and rover energetics data produced by these interfaces was streamed in real-time to an observer at Arizona State University (ASU). A video feed from a camera observing mission control was transmitted as well.

An additional constraint was placed on the astronauts by limiting the total permitted traverse distance to 1,000 meters. This was monitored by the positioning officer, who was equipped with the ArcGIS mission planner system also used in the previous field test (Figure 5.9). The positioning officer was responsible for keeping track of all sites visited by the astronauts and finding the approximate total distance travelled. More importantly, they were to immediately evaluate any proposed astronaut travel to make sure that the astronauts would not violate the distance constraint. This could be accomplished by entering each site as a waypoint and computing the traverse path. The ArcGIS interface then provided the estimated distance for each complete route. Due to the flat and simple nature of the Killian terrain, the optimally straight paths computed by the mission planner were very good approximations of the intuitively direct paths taken by the astronauts, hence the predicted distance values were valid.

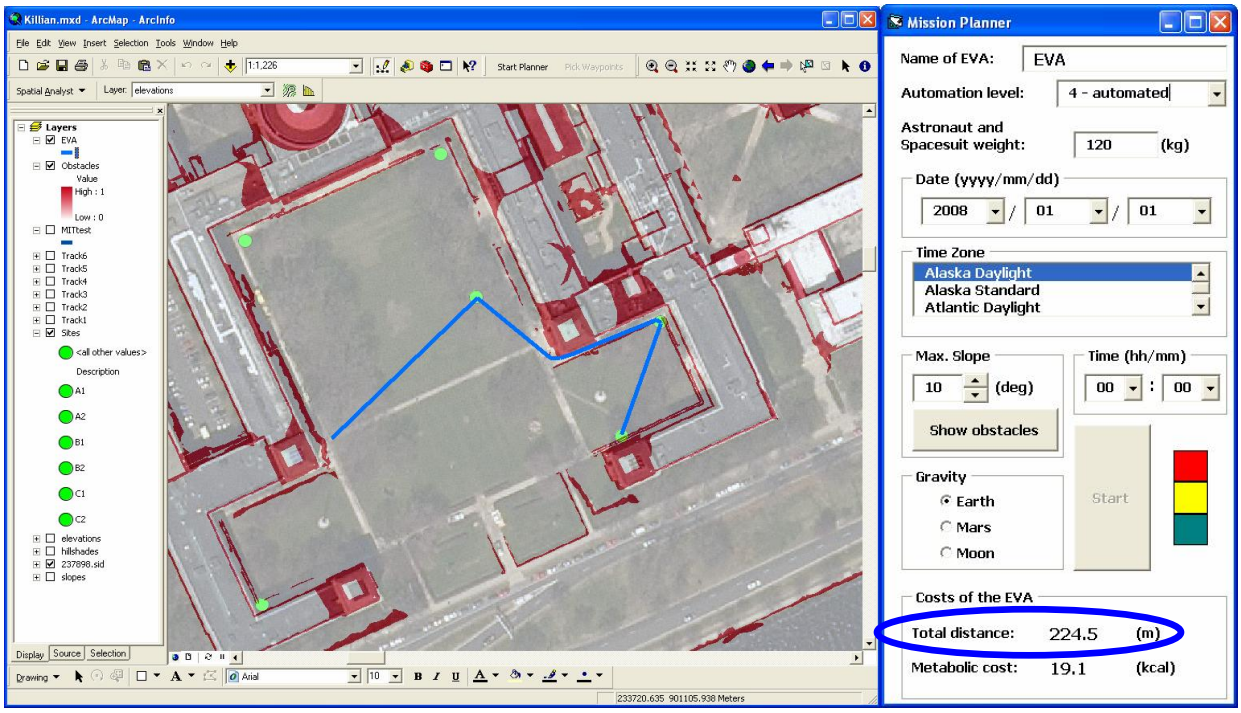


Figure 5.9 Using the mission planner system to monitor traverse distances

Overall, activity constraints were set such that no single explorer could feasibly visit or return samples from all sites of interest. Hence, operational strategies had to be developed in order to best satisfy the mission objectives.

5.2.2 OPERATION

The class was given forty minutes to establish an overall mission operation plan, after which the first EVA simulation began. The rovers, with cameras and no distance constraint, were initially employed as scouts and sent individually to the farthest waypoints while the astronauts were immediately sent to nearer waypoints. The general strategy was to use the video feed from the rovers to identify if sites contained samples of interest. If so, astronauts would be sent there to collect them. If not, then the astronauts could save a trip. Mission control also determined the best order for astronauts to visit interesting sites to minimize distance and oxygen consumption, with all final decisions made by the director.

Teams soon discovered that rover mobility was significantly slower than predicted. This forced mission control to decide whether to have the astronauts wait for the rovers to arrive at the respective objective sites, during which time oxygen is still consumed though at a low level, or to have the astronauts proceed without the desired scout information. Astronauts waited briefly on two separate occasions, but in a third case were instructed to proceed to an uninvestigated site as the rover, running low on battery, turned back toward base.

At the end of the first run, a contingency occurred. Running low on oxygen, the astronauts were making their final return to base with samples when it became apparent that a rover would be unable to make it back under the remaining battery power. Since the rover had a higher priority, the astronauts were instructed to abandon the samples and immediately divert to the rover to carry it back (a permitted astronaut ability). This took the astronauts significantly off their planned course. In the end, although all explorers did make it back, the astronauts were left with less than one percent of the oxygen supply oxygen remaining.

The first run accomplished all EVA objectives, collecting a total of 9 samples with at least one from each zone. The astronauts travelled a total of 803 meters. At this point the surface team and

mission control members swapped roles. The class was given an additional five minutes to re-plan a new strategy before a second EVA simulation began. This was run with identical map and objectives, though the samples at all sites were shuffled.

An interesting astronaut-robot cooperation scheme was devised for the beginning of the second run. Recognizing the limitations of the rovers and the much greater mobility of the astronauts, the astronauts were immediately sent to the farthest waypoints while carrying two rovers most of the way. The third rover that had nearly been stranded in the first run was left unused at base. The rovers were released at spots nearest to two other waypoints along the way. While the astronauts explored the two farthest waypoints, the rovers easily made it to their respective waypoints in plenty of time to scout them. The astronauts returned samples from the farthest waypoints, one of which had nothing of interest, and then proceeded to the scouted waypoints while the rovers made their way back to base via the remaining nearest waypoints. In this manner, by the time the astronauts returned with their second set of samples, all waypoints had been scouted. Furthermore, the rovers were headed back to base with plenty of energy remaining. The astronauts were finally sent to an interesting waypoint in the last remaining zone to satisfy the second objective.

With the luxury of extra time, the team realized that once a sample had been collected from each zone, the third objective of collecting as many samples as possible did not stipulate that they had to be from different zones. Hence, the astronauts completed two final round trips collecting samples at the nearest waypoints with remaining interesting samples. Despite having moderate oxygen remaining that perhaps may have allowed another trip, the director decided to end the mission and avoid putting the astronauts at risk like in the preceding simulation.

The second run also accomplished all objectives, but this time with more than 20% oxygen or battery power left for all explorers. In addition, the team gathered 14 samples and hence was significantly more productive than in the first run. The astronauts travelled a total of 899 meters. This greater distance was covered at a lower oxygen cost since the astronauts were never made to wait.

5.2.3 CONCLUSIONS

Although not an initially stated objective of this experiment, the importance of evaluating mission strategies and surface team interactions became abundantly clear. While both scenarios incorporated identical terrain, explorers, objectives, and constraints, the second simulation produced a significantly greater return with less cost and much more safely due to a superior team operation strategy. These high-level decisions came about through experience and human reasoning, with no aid arising from the mission support system.

The ArcGIS mission planner system used by the positioning officer theoretically could have been employed to compare potential scenarios stemming from different strategies; however this would have been impractical. In fact, the system struggled just to keep up in tracking the total explorer distance travelled. In both runs, the controller fell slightly behind while trying to model the continuous explorer activity. This eventually led the director to somewhat ignore the positioning officer and give astronaut commands before the associated traverses could be verified to not violate the distance constraint. Instead, distances were calculated after the fact. Fortunately the constraint never came into play in these simulations; however, this manner of operation is generally unacceptable in high-risk situations.

The delay in traverse modeling came as a result of both the limited planning capabilities in the ArcGIS interface and the necessary calculation time for generating traverse routes. In this system, the user had to essentially start from scratch in modeling each successive traverse. Entered waypoints could not be edited, and instead an entirely new traverse needed to be established each time. Once a new set of waypoints was entered, it took nearly a full minute for PATH to output the traverse and associated distance. By this point, the astronauts were generally ready to move on if they hadn't already, leaving not enough time for the controller to keep up and certainly none to additionally evaluate potential successive activities.

Aside from this, all systems successfully operated as intended. The energetics models, which ended up limiting explorer activity in each case, functioned well in providing the real-time information necessary for controllers to determine when an explorer must return to base. Furthermore, the corresponding data and video link was received in real-time at ASU. This was a

first step in demonstrating both the feasibility of operating mission control from separate locations as well as the capability of transmitting actual explorer physiological signals or diagnostics to mission control.

The audio and video links to the surface team operated well, however it was found that video feeds were far more informative and alleviated confusion in mission control. When a rover arrived at a site, the controllers knew the precise rover position and the exact character of samples at the site simply by glancing at the display. On the other hand, when the astronauts with only an audio link arrived at a site, there was occasional misunderstanding over exactly which site they were at as well as repeated confusion over the description of samples at a site. Hence, video feedback from the surface team is highly preferred.

Overall, the laboratory activity was a great success. Students showed a high level of interest and enthusiasm, and useful results in regard to the mission support systems were obtained. The test further provided experience to aid in setting up more elaborate and realistic EVA simulations, highlighting specific aspects which were most useful as well as those in need of improvement.

5.3 JOINT EVA SIMULATIONS AND THE MOTIVATION FOR PATHMASTER

This project was completed as part of a collaboration between MIT, Arizona State University (ASU), and the Jet Propulsion Laboratory (JPL). Beginning shortly after the Fundamentals of Engineering EVA simulation presented in the previous experiment, these institutions cooperated in a series of preliminary tests and significant system development over the course of Spring and Summer 2008. The purpose of this work was to create a comprehensive and versatile system by which high fidelity simulated EVA scenarios could be performed. The general desired framework involved sending a physical team of astronauts and robots to venture out on a remote terrain while monitored and commanded by a mission control team operating jointly over three separate locations at MIT, ASU, and JPL. The ultimate goal was to emulate a real lunar or Martian EVA as closely as possible in order to eventually investigate optimal team (astronaut, robot, and mission control) interactions and handling of mission contingencies.

The major criteria of this system were to foster accurate modeling of EVA situations, enable strategic planning and real-time re-planning of EVA activities, and to provide an interactive environment that could be used by both explorers and mission control to evaluate all mission information and accurately carry out a plan. The author at MIT took the lead in creating an interface for real-time situation modeling and mission planning, while a team at ASU undertook the development of a mission information environment and display.

As a beginning point, the ArcGIS mission planner system was considered for support of these simulations. A main shortfall in the first field test using this system was an inability to update mission models and re-plan accordingly. In the second field test, this system was incorporated in a mission control setting to test the feasibility of real-time re-planning. However, even with a dedicated operator tracking only a single traverse, the system was unable to keep up with the physical explorers and much less capable of evaluating potential activities ahead of time. Furthermore, the architecture required users to be familiar with using ArcGIS. Lastly ArcGIS was not readily available at all desired mission control locations.

Faced with these challenges and the desire to create an intuitive, versatile, fast, and easily transferrable mission planning system that would perform well under the demands of real-time EVA simulations, the idea for Pathmaster was born. It was chosen to be developed in MATLAB due to the mutual familiarity and access at all institutions as well as the ease of coding and implementation.

The very first concepts of the MATLAB mission planning system as well as the ASU simulation environment are shown in Figure 5.10. These depict a region of the Mars Yard at JPL, which will be described in more detail in the following section, along with planned traverse routes. In February 2008, the first test of these systems was conducted by placing a physical “astronaut” and robot on the Mars Yard. A joint control team at MIT and ASU, linked via videoconferencing software, instructed the explorers to follow the planned routes as pictured at right in Figure 5.10. The explorers followed a nominal path until a contingency was assumed. At this point, the explorers diverted and followed other existing routes back to base.

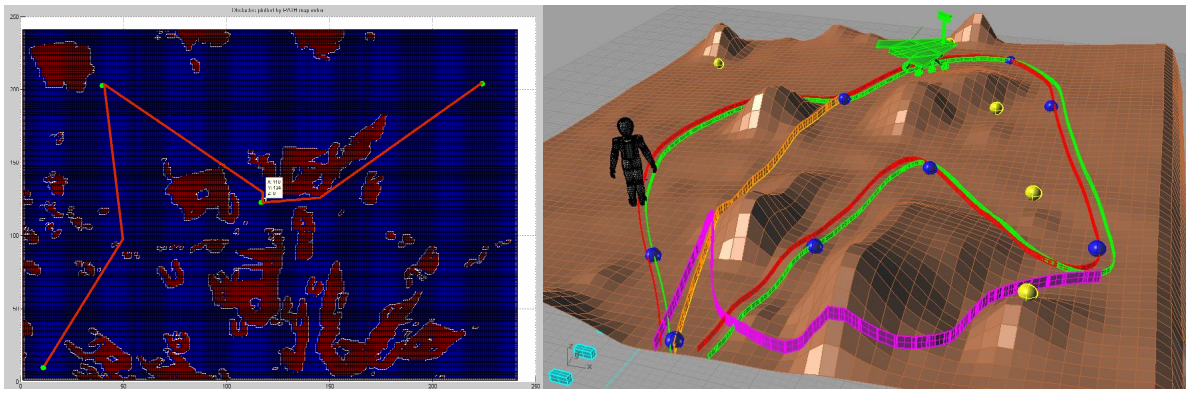


Figure 5.10 Initial mission support system concepts. At left, path planning in MATLAB on a terrain with obstacles, shown in red. At right, OpenSceneGraph rendering of explorers on the same terrain with pictured nominal and contingency routes.

Though this test was a moderate success, with teams at MIT, ASU, and JPL collaborating together to complete the mission with a modeled contingency, the support systems were essentially non-functional. Instead, the astronaut and robot controller simply followed verbal commands given by mission control describing the paths to be followed. In response, over the next few months the MATLAB system known now as Pathmaster was refined and eventually expanded to include a broader set of additional EVA factors from those presented in Chapter 3. Meanwhile, the ASU system evolved into the Astronaut Rover Mission Simulator introduced in the previous chapter.

5.4 JOINTLY CONTROLLED EVA ON A REMOTE TERRAIN

In July of 2008, a complete test of several newly developed mission support systems, including Pathmaster as presented in Chapter 4, was conducted. The experiment was performed as a collaborative EVA simulation involving teams from MIT, ASU, and JPL. The purpose of this test was to evaluate the cooperation and capabilities of the collective support system in a realistic, time-pressured mission scenario. In particular, the ability to adjust mission models, re-plan, and execute commands in real-time while continuously tracking the explorers was examined.

5.4.1 SETUP

The general architecture for this test was presented in the previous section. The simulation included a mission control team and a field team. Mission control operated jointly from each of the three institutions, while the field team consisted of a physical "astronaut" on foot and a four wheeled robot. The remote terrain selected for this test was the Mars Yard at JPL, shown in Figure 5.11. This approximately half-acre region is specifically designed to present an approximation of extra-terrestrial terrain, including the soil type and scattered boulders. It was easily accessible to the JPL team and enabled an internet connection, which was heavily utilized.

Figure 5.11 The Mars Yard at JPL, looking south

A digital mapping of the Mars Yard was made using the Reigl LIDAR scanner. Scans were made from each corner of the yard, and were then "stitched" together to form a continuous surface. In MATLAB, this surface was interpolated to a regular grid using a Delaunay triangulation. This grid, stored as a matrix, was directly loaded into Pathmaster and obstacles were defined for all areas with a surface slope greater than ten degrees (Figure 5.12).

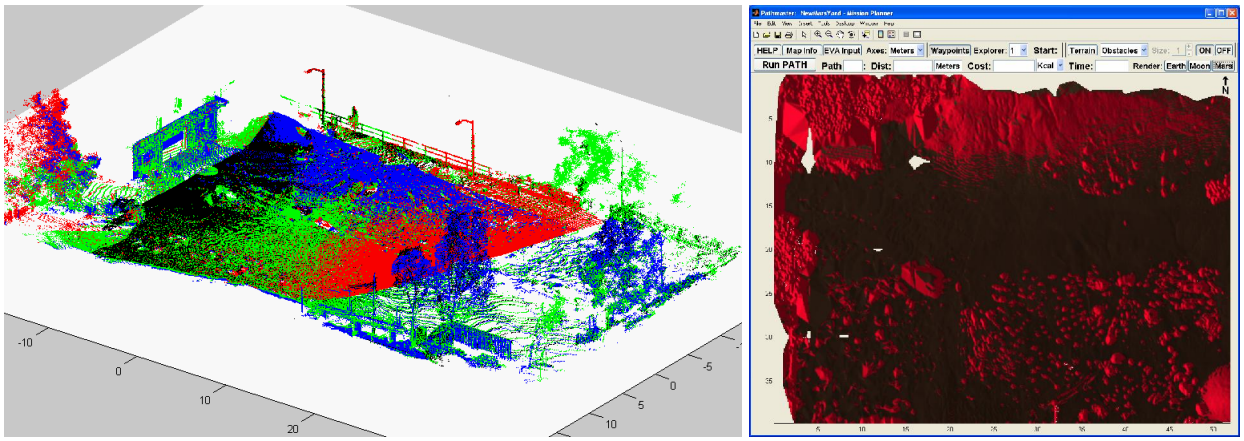


Figure 5.12 Point cloud of Mars Yard Reigl mapping looking southeast (right) and associated Pathmaster mapping viewed aerially (left)

The astronaut was equipped with a laptop for viewing mission information, while the robot was controlled remotely by the mission control team at JPL. The robot was equipped with a camera to provide video feed to mission control, and JPL also provided additional camera views surveying the Mars Yard as a whole. A communicator at JPL was given the task of relaying commands to the field team and passing explorer feedback to mission control.

Mission control at MIT was given the primary task of mission planning and re-planning, which was performed in Pathmaster. The MIT controller also assumed the responsibility of making final decisions and announcing mission commands. In addition, astronaut physiological signals were approximated via a LabVIEW model run at MIT. The controller entered the appropriate astronaut activity in the interface, shown in Figure 5.13 and detailed in Appendix E, and estimated values were given for heart rate, breathing rate, oxygen consumption, and carbon dioxide production. This model, similar to those employed in the previous test, was employed to mimic the actual signals that could be retrieved through wearable sensors and in turn demonstrate the capability of managing these signals.

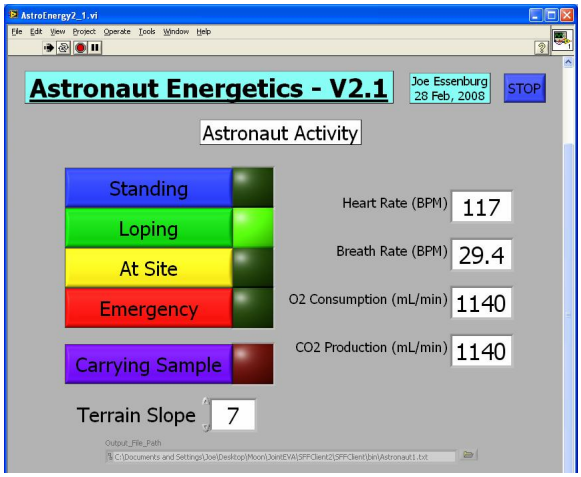


Figure 5.13 LabVIEW interface for approximating astronaut physiological signals

Lastly, the team at ASU was put in charge of tracking the explorer positions in real-time, which was accomplished within ARMS. This team was also responsible for setting up and maintaining the network used for mutual information sharing between mission control sites, described next. The allocation of mission control tasks is summarized in Table 5.1.

Table 5.1 Summary of joint mission control task allocation

MIT	ASU	JPL
• Mission planning & re-planning • Announcing commands & decisions • Monitoring astronaut energetics	• Tracking explorer position • Establishing mutual data network	• Video surveillance of explorers • Relaying commands to the field team • Conveying explorer feedback

Through a variety of software, each mission control site mutually shared all information. A virtual private network was set up through a freeware application called Hamachi. Data streams including the astronaut physiological signals as well as positioning updates from ASU flowed directly over this network. Audio communication between sites was accomplished via a Voice-over-Internet protocol program called Ventrilo, while all video signals were fed to the videoconferencing website MeBeam (http://mebeam.com). By dividing tasks between several systems, issues with limited bandwidth and lagging signals were mitigated.

5.4.2 OPERATION

After a first day of running a preliminary EVA example to ensure all systems were functional and fix any bugs, the mission plan shown in Figure 5.14 was presented for day two. Explorers began at the eastern edge of the map in a shed representing a lunar base. The astronaut mission, shown to the north in blue, was to explore all gaps in the terrain dataset. These areas of no data appear as white holes in the map. The explorer was to evaluate why each gap may have occurred, which could provide useful feedback for improving future mappings. After finishing at waypoint F, the astronaut was to await further commands. The rover mission, shown in yellow to the southeast, was to proceed through the rocky area near the base and examine six potential sites of interest, especially noting if any sites should also be visited by the astronaut. Upon finishing at waypoint 6, the rover was to await further instructions.

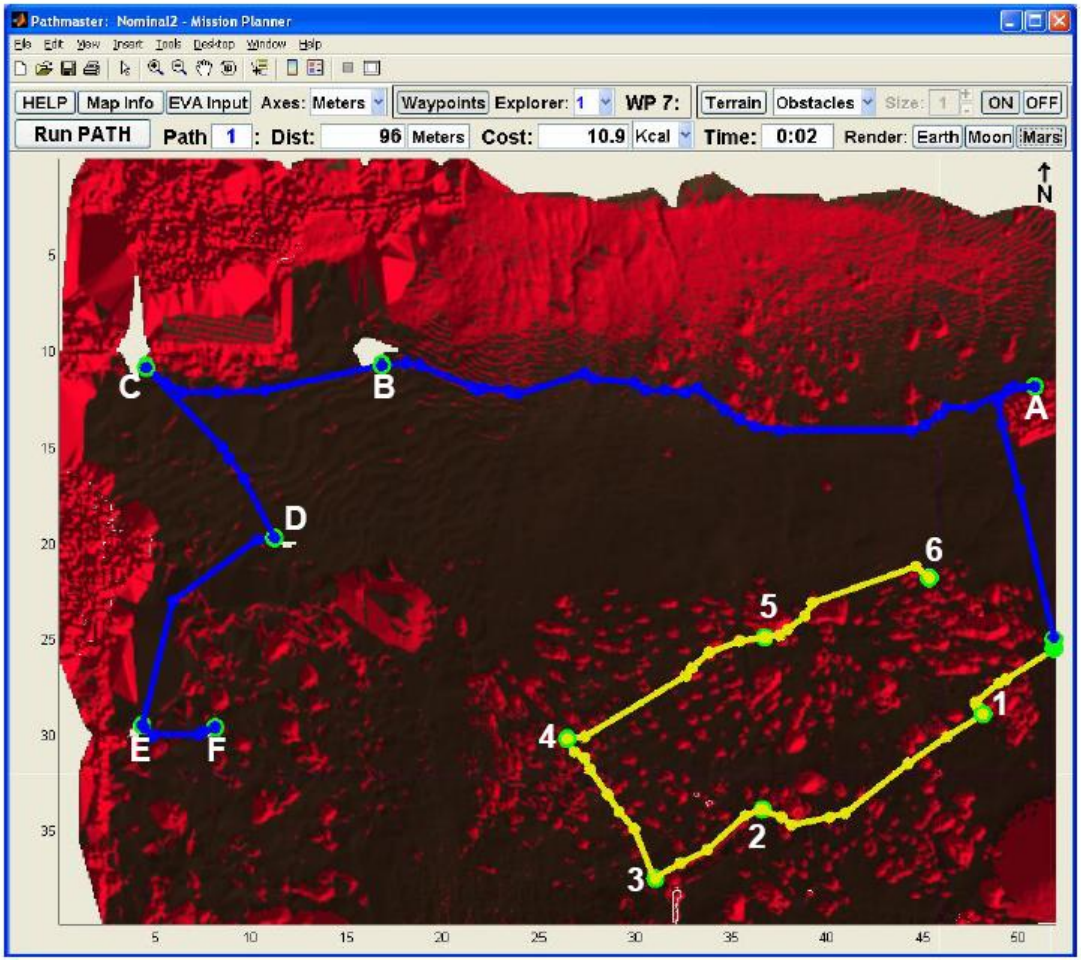


Figure 5.14 Initial mission plan with sites labeled

The astronaut laptop was loaded with an image of this mission plan, and the same image was provided to the robot controller. This served as a map to guide the explorers, which was sufficient in this case given the small scale and familiarity of the terrain. The waypoints and routes were loaded into ARMS as well. The mission began with the command for the astronaut to proceed to waypoint A and the robot to waypoint 1. Upon embarking, the LabVIEW energetics model was engaged and the astronaut activity monitored. Meanwhile, astronaut and rover positions were manually updated in ARMS as the explorers physically moved.

The astronaut completed the entire initially planned traverse without issue. The features at each site were verbally described and are recorded in Table 5.2. An example astronaut view from the ARMS display tracking the astronaut is shown in Figure 5.15.

Table 5.2 Astronaut feedback from planned waypoints

Waypoint	Feature feedback
A	Non-interesting pile of rocks
B	"Crater" apparently caused by water erosion
C	Behind a storage shed
D	Crevice between rocks
E	Divots in the ground
F	Divots in the ground

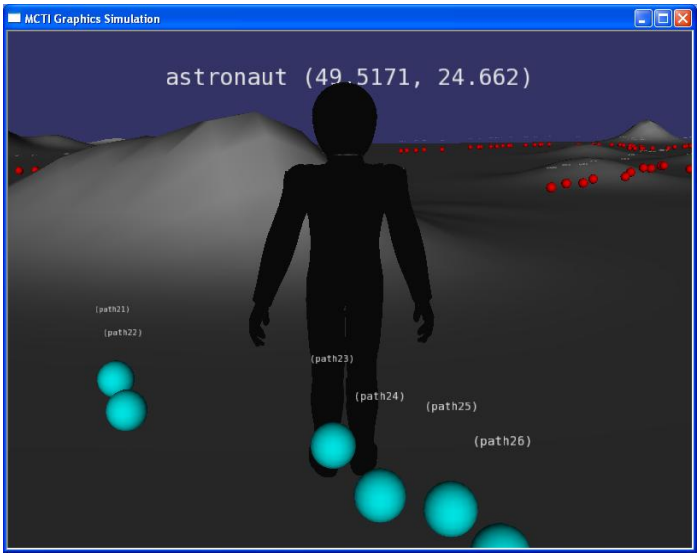


Figure 5.15 Astronaut view of the rocks at waypoint A as seen in ARMS

5.4.2.1 COMMUNICATION FAILURE

Shortly after the astronaut was commanded to proceed to waypoint B, the first mission contingency occurred. The Ethernet connection to the main mission control computer at MIT inexplicably went down, disconnecting all incoming and outgoing communications. The ASU team soon realized the situation, and in response assumed the responsibility of announcing mission commands. In turn, the MIT team coped by establishing limited communication through alternate means. The astronaut energetics model was being run on a separate laptop, and its wireless internet access was unaffected. Since data was streaming live, this system was already connected to the mission control network through Hamachi. Taking advantage of this, the MIT team was able to textually chat with the teams at ASU and JPL (a feature of Hamachi). Hence, despite no longer being in effective control of the mission, MIT was able to remain updated and record explorer data through a backup communication channel.

As a precaution, Pathmaster was quickly loaded onto the laptop in case re-planning became necessary, though it never came to this. The communication failure lasted for just over ten minutes, after which full internet access was restored and MIT resumed all typical responsibilities. During the outage the mission proceeded without delay, and the astronaut visited sites B through E. The field team was likely never aware of any problem.

5.4.2.2 ROBOT FAILURE

Soon after mission operations returned to normal, the first re-planning became necessary. The robot performance was significantly slower than expected. Without yet discovering anything interesting, the robot was only at waypoint 3 by the time the astronaut was finished with the entire initial traverse (the robot visited waypoint 2 during the communication outage). In response, waypoints 4 and 5 were assigned to the astronaut while the robot was instructed to proceed directly to waypoint 6 near the base. However, robot mobility became severely limited after leaving waypoint 3, and the battery soon died. The astronaut, already having noted an interesting rock formation and en route to waypoint 5, was instructed to rendezvous with the downed robot upon leaving the waypoint. The corresponding mission plan, shown in Figure 5.16, was developed in Pathmaster at MIT. The mission data file along with a screenshot were sent to all sites via e-mail.

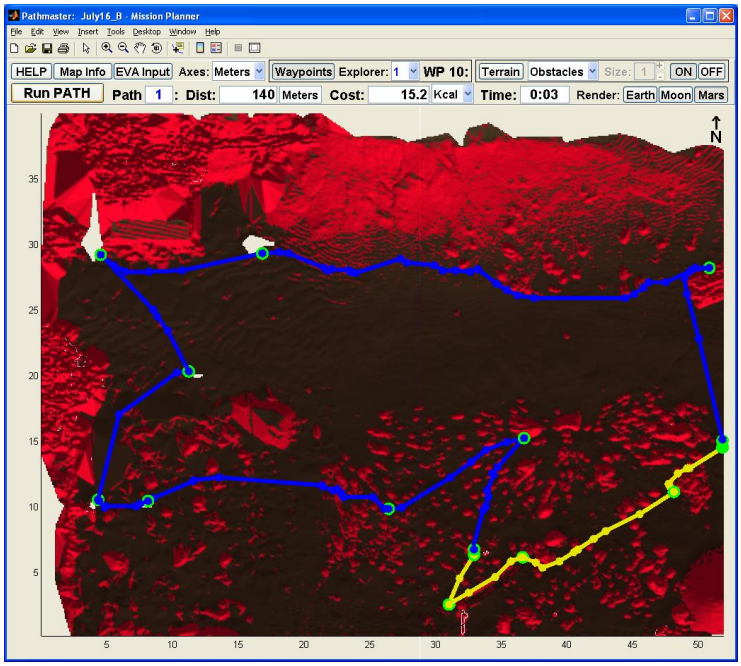


Figure 5.16 Astronaut assumes two robot waypoints, then rendezvous with the robot

Upon meeting with the astronaut and receiving a new battery, the robot was deemed fit for travel. Instructions were given to proceed directly north to get clear of the difficult rocky terrain then head directly back to base. The astronaut was instructed to finish up at waypoint 6. The new mission plan was quickly developed and transmitted (Figure 5.17).

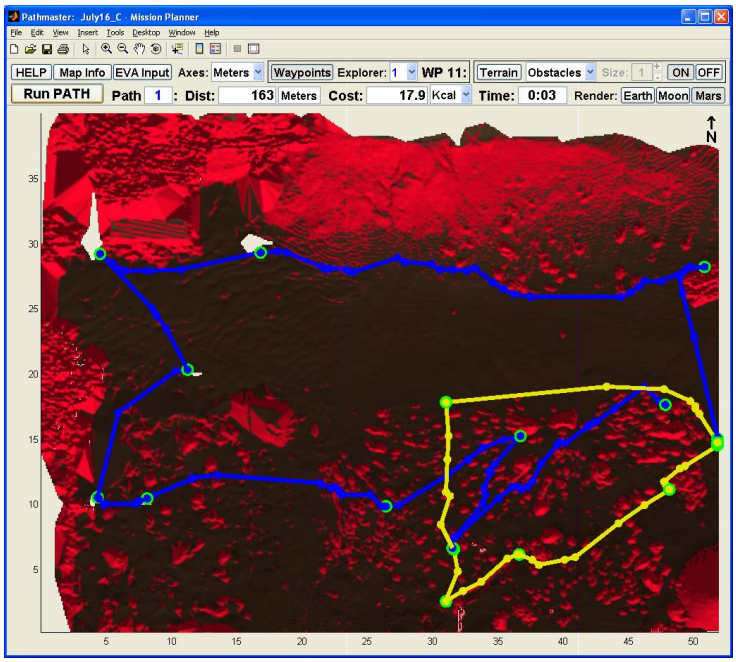


Figure 5.17 Robot proceeds to base, astronaut to waypoint 6

However, after traversing only a short distance north the robot shut down again, this time due to overheating under the California summer sun. The astronaut was instructed to immediately leave waypoint 6 and carry the rover back to base. This final plan update is shown in Figure 5.18. Upon returning to base and with all planned waypoints visited, the mission was ended.

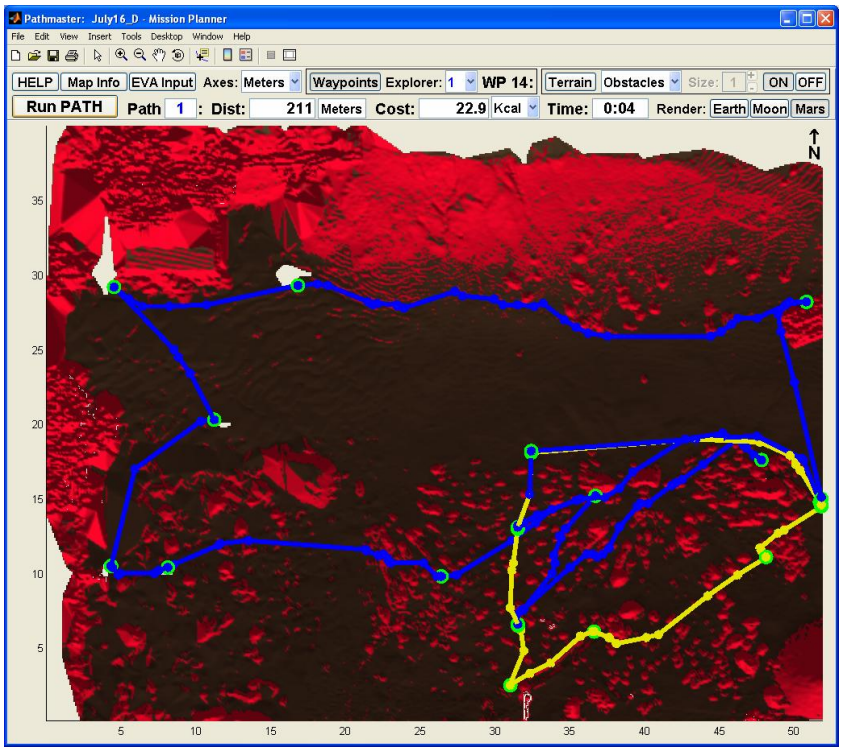


Figure 5.18 Astronaut leaves waypoint 6 to meet robot and carry it back to base

5.4.3 CONCLUSIONS

Overall, this test was considered a resounding success. Although unexpected system failures occurred, mission control was capable of coping in real-time above and beyond the initial scope of the experiment, still salvaging all simulated EVA objectives. The initial thought before beginning was to artificially impose contingencies as desired. However, this was clearly not necessary as unforeseen situations emerged without provocation. This enabled an even more realistic test of real-time response capabilities, and all mission support systems performed as well as could be hoped.

The first contingency, where the MIT team nearly went completely offline, illustrates the importance of redundancy in vital systems. Had solely the MIT team been controlling the

mission, going offline would have crippled all operations and stranded the field team. Operating mission control from several locations enabled activity to proceed uninterrupted as the other control teams were able to cope. Moreover, due to a backup communication line between mission control sites, MIT was able to still remain connected and receive mission updates. The incorporation of redundant backup systems is a crucial consideration for future EVAs.

The second contingency demonstrates the same shortcomings of robots as seen in the Fundamentals of Engineering exercise. The robot progress was unable to keep up with the astronaut, and in this case the astronaut expended a large amount of time and effort attending to the robot and making up for its unfulfilled objectives. This is generally unacceptable considering the high cost of astronaut activity. Robot technology and operational strategies must be developed to make these systems a benefit and not a hindrance in missions.

As far as the simulated EVA objectives of examining gaps in the terrain dataset, the information gathered by the astronaut helps explain why the LIDAR scans were unable to gather data: there was no line of sight to these indented or shaded areas. Though not a major issue here, this is a consideration that should be made when conducting future mappings, especially on more difficult terrain. An aerial mapping could alleviate this problem.

The primary systems being tested, Pathmaster and ARMS, performed as intended. The astronaut and robot positions were able to be tracked in real-time without issue, albeit manually. As far as re-planning in Pathmaster, the first attempt took approximately three minutes to make a decision, develop the plan, transmit it, and begin execution. This was too slow, and the astronaut was forced to wait some time for the new mission plan to arrive. Successive re-planning occurred progressively faster, with the final plan developed in under a minute. While this was somewhat acceptable, a more expedited procedure is desirable.

At this point, a wide variety of EVA scenarios may be robustly handled by these systems. However, there is still much room for improvement in developing higher fidelity models, operating more complex and realistic mission scenarios, and providing enhanced support. This is discussed in the following chapter.

6 CONCLUSION AND RECOMMENDATIONS

6.1 CONTRIBUTIONS

This work provided a detailed characterization of the makeup and challenges of planetary surface EVAs. Moreover, a specific framework for maximizing the productivity of these missions was established. Recognizing the need for automated support in achieving such optimal performance, methods by which all pertinent mission factors may be quantitatively modeled were presented and the subsequent architecture of a comprehensive support system employing these factors was developed.

The greatest contribution of this research was the creation of a prototype automated mission support system for optimizing planetary EVA operations. Based upon the developed architecture, this system is effective both for pre-mission planning and strategic scenario comparison as well as for real-time re-planning and explorer navigation assistance. The prototype presents an intuitive interface where controllers may quickly represent various situations and determine a best course of action for immediate execution. Offering enhanced functionality where preceding systems fell short, the program was shown to robustly respond to situational updates and contingencies to maintain optimal performance in time pressured settings.

This system further serves as a tool for future research into optimal mission strategies and team interactions. By collaborating with ASU and JPL, a complete platform for further EVA simulation and testing was established. Beyond research, there is great educational potential for such a system as experienced in the Exploration Lab field test.

6.1.1 CURRENT DEPLOYMENT AT DEVON ISLAND

The prototype system is currently being deployed as part of ongoing EVA research at the Haughton Crater site on Devon Island, Canada. This extremely remote region offers challenging terrain comparable to areas on the moon or Mars. Headed by Marcelo Vazquez of the National Space Biomedical Research Institute, the efficacy of optimal route planning and real-time

navigation support for an astronaut on moderately long traversals is being evaluated (Figure 6.1). In addition, relative measured costs of both suited and unsuited activity are being compared to the predicted values given by the system cost functions.

Figure 6.1 Navigating along an optimal route on a suited traverse at Devon Island

6.2 AN IDEAL MISSION SUPPORT SYSTEM

Beyond the capabilities of Pathmaster, an ideal mission support system would incorporate several additional traits. These represent open areas for future research.

- Actual EVA missions are generally limited by activity constraints as opposed to exhaustion of objectives. An ideal system would handle either case. Hence, all explorer constraints would be explicitly modeled, and in turn all objectives would be clearly prioritized. In this manner, mission optimization could function either by maximizing objective return within the bounds of all operational constraints, or by minimizing costs when given limited objectives.
- The best predictive models come through experience rather than a priori estimates. Applied to activity costs, all energetic signals would be monitored within the system to update the explorer cost models with actual data from previous similar activities.

- An ideal system would be capable of comparatively analyzing surface team dynamics to automatically find a best scenario. In this way, the support system becomes strategic, determining optimal explorer cooperation schemes including ordering and allocation of mission tasks.
- Field explorer support would be provided in the most effective manner, such as an automatically updating heads-up display for astronauts and analogous data stream for robots. This would seamlessly navigate the explorer along the mission plan and enable consistent optimal operation. It would enhance interaction with the terrain by clearly distinguishing features or sites of significance.

6.3 DESIGN RECOMMENDATIONS

The Pathmaster system, while fully operational in its current state, contains numerous aspects open for immediate development. This section outlines recommendations for improving the fidelity and completeness of the system. Items are listed in general priority as determined by the author.

6.3.1 LINKING PATHMASTER WITH GPS

Before Pathmaster may function as a complete support tool in the field, it must become capable of real-time interactive navigation support. This is done most conveniently through a GPS link. Such capability would enable tests involving re-planning in the field, which is highly desirable. There are several apparent strategies by which this may be accomplished.

All traverses involving GPS positioning are currently run through the ArcGIS mission planner system. When a traverse is made in ArcGIS, a “shapefile” is overlaid along the terrain detailing the planned route. A laptop with a GPS receiver can incorporate a position marker directly within this interface for navigation. To use a handheld unit such as the Trimble, the ArcGIS view is exported as an image with an associated “world file” which is subsequently loaded on the mobile device. The Trimble uses ArcPad to load the image and display current position from the GPS receiver.

1) Pathmaster could write appropriate output files, and the ArcGIS system could be modified to read these files directly and produce corresponding shape files representing the paths.

2) Pathmaster perhaps could generate shape files directly for ArcGIS. Recall that maps contain UTM positioning data, and that coordinates for any point may be found in either UTM format or latitude and longitude.

3) Pathmaster perhaps could bypass ArcGIS and create images and world files directly.

4) Perhaps the best option would be to perform the process entirely in MATLAB. A separate simple m-file could load an image of the terrain map with planned paths, acquire GPS data from a receiver, and plot the position accordingly, updating every second or so. This could also be done within Pathmaster, though that may not be the best option when faced with limited computing resources. The MATLAB central file exchange has some example m-files for collecting GPS data.

6.3.2 EXPLORER COST FUNCTIONS

Pathmaster handles several data parameters that are currently unused in finding traverse paths, including explorer type, time of day, soil mechanics, and scientific return. These values should be incorporated into the cost function used when optimizing traverse routes. In the cost function section of the Pathmaster code, the cost to be minimized is stored in the variable *C*, while the heuristic estimates are stored in the variable *H*.

Distinct cost functions for astronauts, rovers, and robots should be developed. The type of each explorer is stored in a cell array called *Data.Explorer*. This value would be used to signal the corresponding function to be employed.

The time of day is used along with explorer global position in determining sun position. The method by which illumination may be incorporated into the astronaut cost function used for optimizing paths has already been presented by Márquez (2007). In particular:

$$\textit{Exploration Cost} = (\textit{Metabolic Cost}) \bullet (1+1/2 \bullet SS) \text{ , where}$$

$$SS = (\cos(2\theta)+2) \bullet (\cos(2\varphi)+2) \qquad (\textit{Sun Score}, \text{ Carr et al., 2003})$$

The metabolic cost is currently already found in Pathmaster. All that remains is to find θ and φ, the respective azimuth and elevation angles of the sun relative the crew member. The methods for this are essentially fully developed in the PATH Java classes. First, the time information must be converted to UTC time if on earth, or Pasadena time for the moon or Mars. Next, the *SunElevation* class shows how to find the sublatitude and sublongitude of the sun by direct calculation for earth or table lookup for the moon or Mars. Lastly, the *Illumination_from_sunpos* class uses these values to find the sun elevation (φ), and azimuth can be found in relation to the direction of each point to point travel. This calculation also requires knowledge of the explorer latitude and longitude. Pathmaster has a routine for finding these on earth. A corresponding algorithm would need to be developed for the moon or Mars. To speed calculation, a single latitude and longitude coordinate can probably be assumed for an entire terrain due to the relatively small planetary scale of our maps.

Soil mechanics and scientific return data are stored as arbitrary index values. See the respective sections under *Editing Terrain Characteristics* in Chapter 4 for a conceptual overview on how these could be employed in cost functions.

A final consideration would be to incorporate any costs related to waypoint site activities in the total mission cost estimates. Perhaps an activity to perform at every waypoint could be included as a selection in the EVA Input menu, enabling different choices for each explorer.

6.3.3 WAYPOINT ORDERING AND PRIORITIZING

On a traverse, Pathmaster currently visits waypoints in the order in which they were entered regardless of orientation. Alternatively, the order in which waypoints are visited could be automatically optimized as well. The general concept by which this would be accomplished is commonly known as the Travelling Salesman Problem (TSP). This involves finding (or for speed, heuristically estimating) the cost from each waypoint to all others. The general TSP would assume that all waypoints have an equal priority. A more elaborate model would include a method of weighting waypoints to represent relative priority. Presumably, waypoints with a higher priority would be visited earlier whenever practical. Once an order is established, the usual traverse optimization routine could be employed.

6.3.4 ACTIVITY CONSTRAINTS

The only traverse constraint currently imposed by Pathmaster is the terrain obstacles. To promote a more realistic simulation, activity constraints for distance, time, energetic expenditure, etc. could be enforced. Ceiling values could appear as choices within the EVA Input menu (or even a new menu). In turn, these limits could be checked when establishing a path. If a constraint is encountered, the traverse would presumably still visit as many waypoints as possible. A more elaborate model could heuristically measure the cost back to the start. If the current cost plus the estimated cost back to base approaches a constraint, the explorer could automatically be sent back and the remaining waypoints abandoned.

6.3.5 VARIABLE SUN POSITIONING

Assuming that a sun position feature as described earlier in the *Explorer Cost Functions* section has been employed, the next step would be to make that position update along a traverse. This could be accomplished by retrieving a running time estimate as a path is being found, and at certain increments (say every half-hour, or alternatively upon arriving at each waypoint) recalculating the sun position. The path optimization would then proceed with the new lighting values until they are updated again.

6.3.6 INTERFACING WITH ARMS

The ARMS system developed by Uday Kumar at ASU provides an interactive, 3D virtual reality EVA simulation environment (see Figure 4.18). Ideally, all Pathmaster functionality would eventually be incorporated directly into ARMS to form a superior support system. Currently, Pathmaster interacts with ARMS via the "Render" directory chosen at the bottom of the EVA Input menu (this must be C:\Content for use with ARMS). Pathmaster writes waypoint, traverse path, cost, and terrain map data to this directory as a series of text files. Presently, ARMS only loads the waypoint and traverse path information. Although perhaps more the responsibility of the ASU team, ARMS should be developed to incorporate the additional data. If a method for capturing GPS data is developed in Matlab, this may possibly be used to update explorer positions in ARMS as well (or better yet, a direct GPS link to ARMS could be established).

6.3.7 EXPLORER HEADS-UP DISPLAY

While handheld computers have worked well for explorers on EVA simulations up to now, a better option would be to develop a heads-up display for viewing mission information in the field. Such a display could be projected within a space suit helmet, as shown in Figure 2.11 and Figure 3.9. Presumably a feed from either Pathmaster or ARMS with real-time position updates could be employed as the visualization.

6.3.8 INTEGRATION WITH THE DECISION THEATER

A final option for enhancing the capabilities of mission control teams would be to incorporate the support system into the Decision Theater at ASU. The Decision Theater is a seven screen rear-projected environment that fits about twenty people in a conference setting (Figure 6.2). It offers great potential for enriching mission interactions, and has already been employed for virtual EVA simulations on the moon as part of the Engineering Systems and Experimental Design course during Fall of 2007.

Figure 6.2 The Decision Theater at Arizona State University

APPENDIX A: CONTENTS OF ENCLOSED DVD-ROM

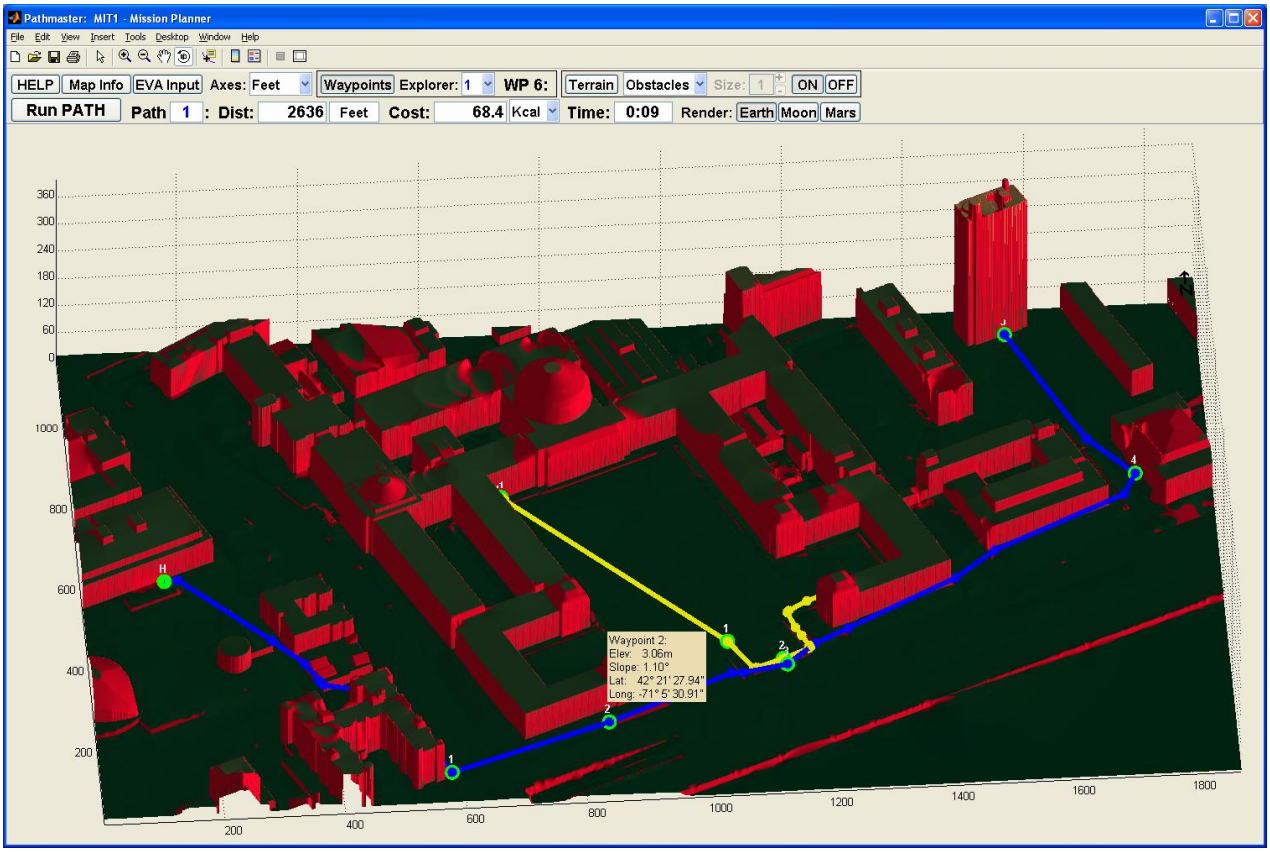
This directory contains electronic copies of all files and software detailed in this thesis. Also included are supplementary files for running joint EVA simulations, the PATH Java software, and additional suggestions for continued work.

FOLDER	SUMMARY
Instructions	EditingPathmaster folder: A copy of the design recommendations presented in Chapter 6 is given along with detailed suggestions on how to modify explorer cost functions and heuristic estimates for the traverse path optimization routine in Pathmaster. Instructions for creating new terrain maps are given as well. Joint EVA folder: A *JointEVA_Procedure* document provides a detailed overview of setting up a computer system as part of mission control for a joint EVA simulation. This is accompanied by instructions for loading and running all necessary software.
Java_Version	The PATH Java software is contained in the PathClasses folder. Also included is an older version of Pathmaster (Version 6.9) which calls PATH directly for determining traverse routes in the same manner as the ArcGIS mission planner system. A *ReadMe* document provides details of this system.
LabVIEW_Models	These are the explorer energetics models used in the Exploration Lab and Joint EVA field tests, as presented in Appendix E.
MissionPlanner	This contains the Pathmaster software. A Terrain_Maps folder includes all developed elevation maps stored as text files, which can be readily loaded in Pathmaster. The Pathmaster m-file itself is given, coded as shown in Appendix C. A PDF User Manual is provided, also shown in Appendix B, along with a PowerPoint presentation outlining the primary features of the software. To load Pathmaster on a new machine, simply transfer over the MissionPlanner directory (this is all that needs to be done, Pathmaster may immediately be run in Matlab on the new machine).
SFFClient	This is the software used to stream explorer energetics data live to the mission control network. Refer to the *JointEVA_Procedure* document in the Instructions folder for details on how to use this system.
WordCopies	Included are Word document files of the Pathmaster user manual and the Exploration Lab instructions, as presented in Appendices B and D, respectively. These are provided to expedite future editing and use of these files.

APPENDIX B: PATHMASTER USER MANUAL

Pathmaster
Mission Planning Interface

User Manual



Joe Essenburg

Man Vehicle Lab, MIT

28 Aug, 2008

Table of Contents

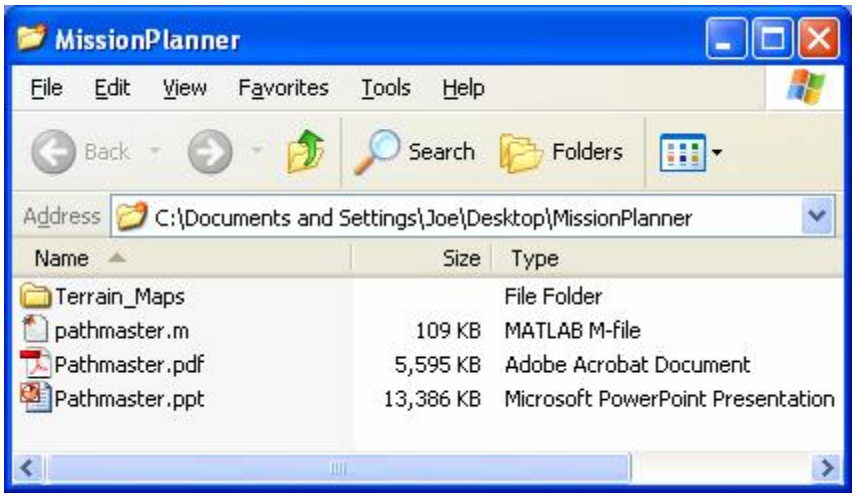
General Information

Pathmaster is a Matlab-based interface for operational support of planetary extra-vehicular activities (EVAs). It is intended to be used both beforehand for mission planning, scenario simulation, and optimization as well as in real-time for explorer navigation and contingency handling. Pathmaster allows users to easily load terrain maps, enter mission data, find optimized traverse routes, record the costs of a traverse, and compare mission scenarios side-by-side. Pathmaster may also be used to feed mission data to an external virtual reality simulation or field display. The optimization employed by Pathmaster is based upon the Planetary Aide for Traversing Humans (PATH) software, developed in Java under Jessica Márquez.

Getting Started

Pathmaster is written for both Windows and Mac OS X. It is intended to be run in Matlab R2007a or later. A minimum monitor resolution setting of 1024 x 768 is recommended.

1) Download and unzip the MissionPlanner directory.

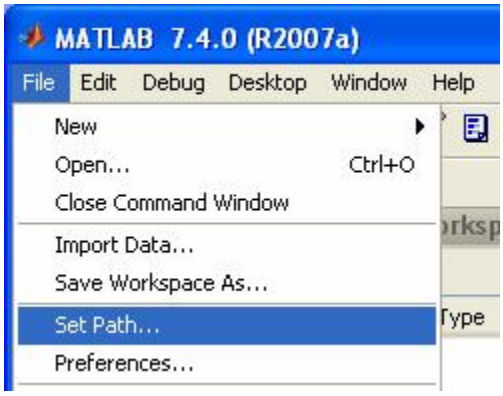


2) Open Matlab. 

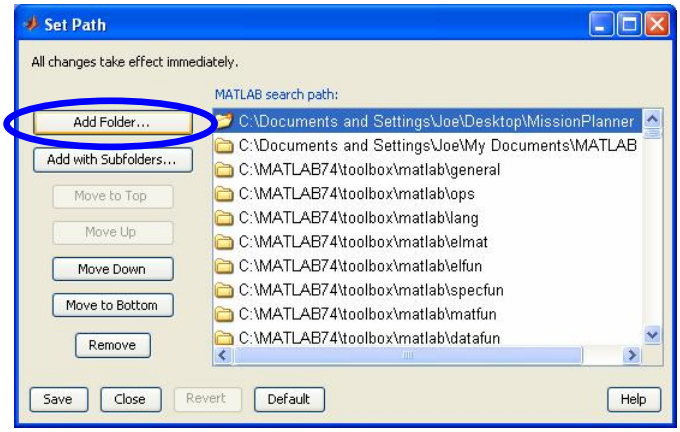
Running Pathmaster

1) Add the MissionPlanner directory to your Matlab search path.

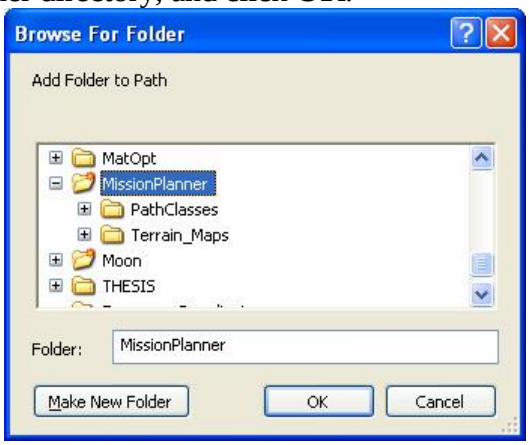
Go to **File\Set Path…**



In the upper left, click **Add Folder…**

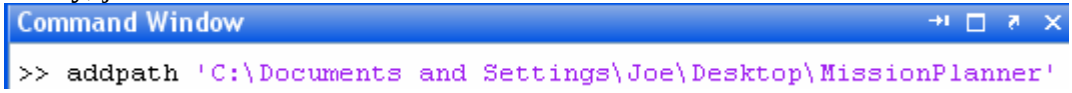


Locate the MissionPlanner directory, and click **OK**.



Click **Save** in the bottom left of the Set Path menu to save changes.

Alternatively, just use the ADDPATH command in Matlab:



Running Pathmaster, continued

2) Enter `pathmaster` at the Matlab command line.

There are four ways to call Pathmaster from the Matlab command line:

>> pathmaster

The command "pathmaster" alone will initialize a prompt to load elevation data from file. This is the normal method of running Pathmaster.

```
>> pathmaster
```

>> pathmaster(Elevmap)

Calling Pathmaster with a matrix argument loads that matrix as the elevation map.

```
>> [a,b] = meshgrid(0:.1:10);
>> elev = sin(a+b);
>> pathmaster(elev)
```

>> pathmaster('lite')

Calling Pathmaster with the 'lite' option employs simpler surface rendering. This speeds plotting time and prevents problems on some machines.

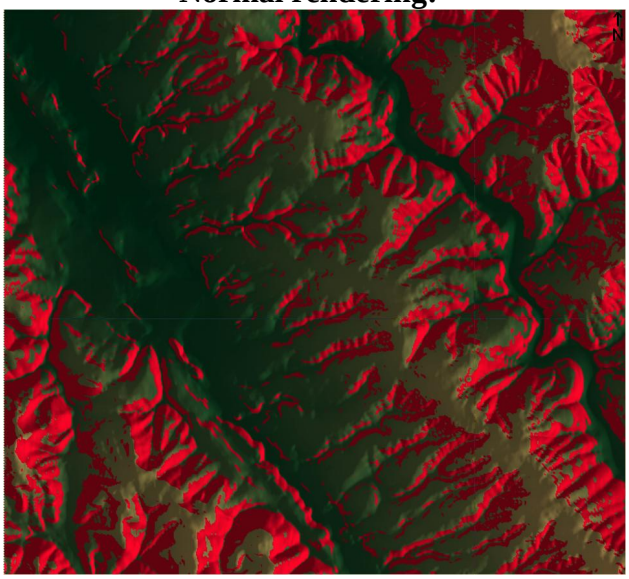
```
>> pathmaster('lite')
```

>> pathmaster(Elevmap,'lite') OR >> pathmaster('lite',Elevmap)

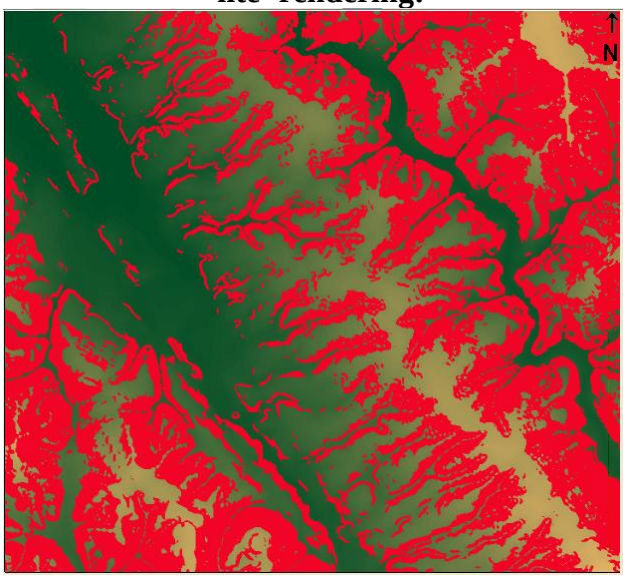
Calling Pathmaster with both a matrix argument and the 'lite' option does both of the above. The arguments may be entered in any order.

```
>> pathmaster(elev,'lite')
>> pathmaster('lite',elev)
```

Normal rendering:



'lite' rendering:

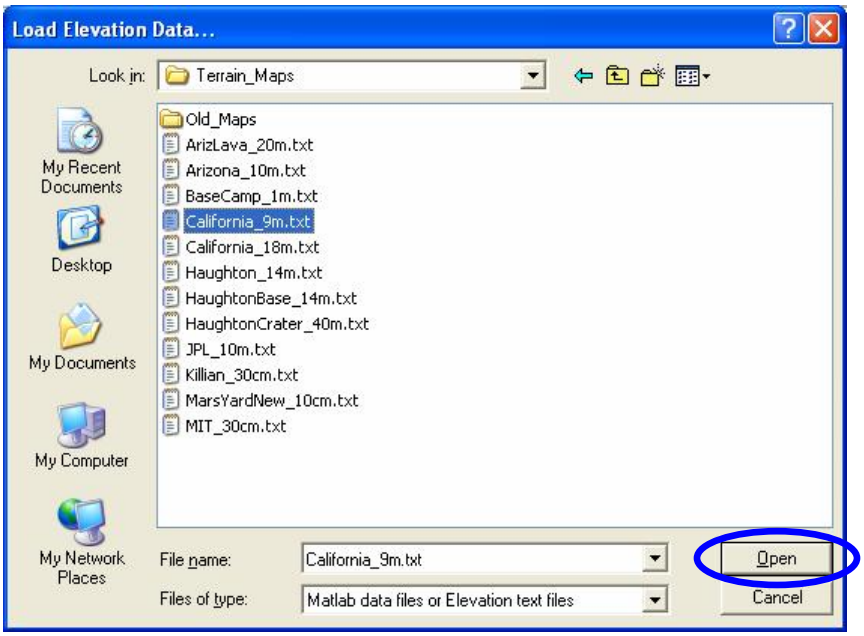


If a machine encounters problems with Pathmaster terrain renderings, use of the 'lite' option is recommended.

To open multiple instances, simply call Pathmaster again from the Matlab command line.

Loading Map Data

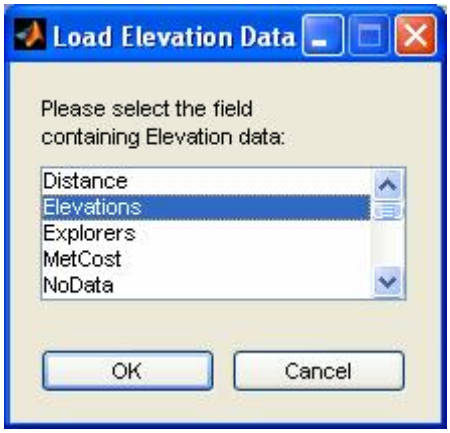
After being called from the Matlab command line, Pathmaster will open the following prompt allowing the user to select the elevation map to be loaded (when no matrix argument is entered):



The elevation map data may be loaded as either a text file (.txt) or a Matlab data file (.mat). All prepared terrain map text files are located in the Terrain_Maps folder. After running a mission, a copy of the mission data will be written to a Matlab data file in the working directory (containing the Pathmaster m-file), which may be used for easy re-loading. For details on these files, see the File I/O section.

Once a file is selected, click **Open**.

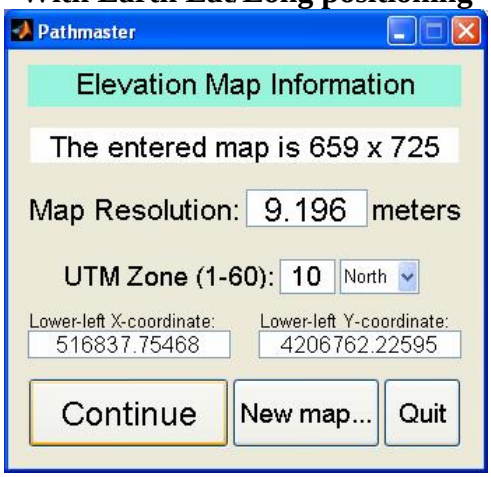
Selecting a Matlab data file will open the prompt below. In this prompt, simply select the field (variable) that contains the desired elevation map matrix and click OK.



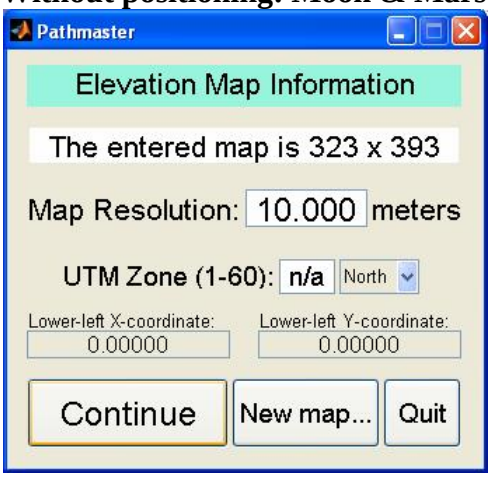
Map Information Menu

Once an elevation map has been loaded, Pathmaster will open the Map Information menu. Here the user may input the map sizing and, if applicable, positioning data. Any data present in the loaded map file is automatically recognized and displayed in the corresponding data fields.

With Earth Lat/Long positioning



Without positioning: Moon & Mars



Map Resolution:
The uniform horizontal spacing between data points in the elevation map matrix, given in meters.

UTM Zone:
Applicable only on Earth, this is the East-West UTM zone where the map terrain is located, numbered 1 through 60. Entering a value here is necessary if the user wishes to use latitude/longitude positioning. When the UTM zone is set, all other positioning data cells become active. The North/South drop-down menu indicates whether the map is located in the northern or southern hemisphere. To deactivate lat/long positioning, simply enter "0" into the UTM Zone cell. A "n/a" will appear and all other positioning data cells will be grayed out.
For more information on the UTM system and coordinates, see:
http://welcome.warnercnr.colostate.edu/class_info/nr502/lg3/datums_coordinates/utm.html

Lower-left X-coordinate:
Easting: The horizontal coordinate of the lower-left (southwest) corner of the map in meters east of the UTM zone origin.

Lower-left Y-coordinate:
Northing: The vertical coordinate of the lower-left (southwest) corner of the map in meters north of the UTM zone origin.

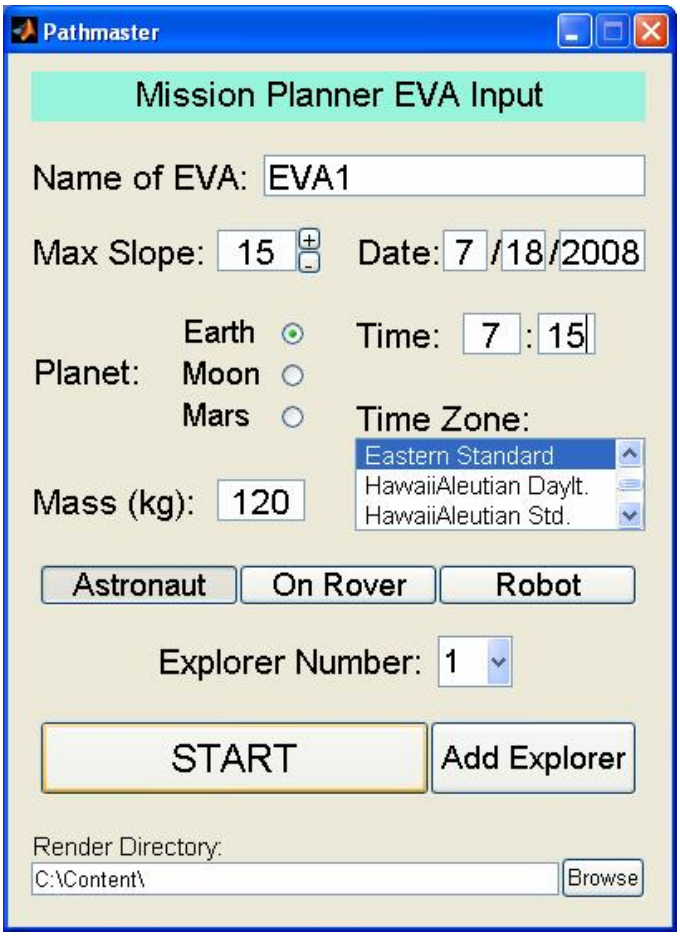
Continue:
Proceed with the current data.

New map:
Clear the current elevation map and open the prompt to load new elevation map data.

Quit or Close:
Exit Pathmaster (nothing has been saved at this point).

EVA Input Menu

After all map information has been entered, Pathmaster opens the EVA Input menu.



Name of EVA:
A descriptive name for the mission scenario to be run. All output files will be stored beginning with this name. Run each mission scenario with a different name to prevent data from being overwritten. Only alphanumeric characters and underscore are allowed, no spaces.

Max Slope:
Maximum traversable terrain slope, in degrees. All terrain with slope greater than this will be presented as an obstacle. The buttons at the side increment and decrement the slope by one.

Mass:
Total explorer mass including gear, in kilograms.

Planet:
Indicates upon which planet the EVA takes place. This sets the gravity assumed when finding traverses as well as the initial rendering mode.

Date, Time, & Time Zone:
Select the date and military time along with the corresponding time zone for which the EVA takes place. This is used in determining the sun illumination angles.

Explorer Type:
Select whether each explorer is an astronaut on foot (Astronaut), riding a rover (On Rover), or whether it is an unmanned robot (Robot).

Multiple Explorers:
Pressing the "Add Explorer" button will add a new explorer to the current mission. Any number of explorers can participate in a mission. Selecting an explorer number from the drop-down menu will make that explorer active, and the explorer type and mass will be shown in the corresponding fields. Each explorer has an independent type and mass; all other fields are constant for all explorers**.** If you wish to use differing terrain parameters or times for a certain explorer, simply open another instance of Pathmaster. Make sure to give the separate instance a different EVA name.

EVA Input Menu, continued

Render Directory:

This is the directory where all external renderer data files are written and read. The user may enter a file path manually or use the Browse button.

Load existing Waypoints, Obstacles map, etc:

If matching waypoints or additional terrain parameters were stored along with the loaded map data, corresponding checkbox options will appear near the top of the EVA Input menu. Selecting the checkboxes causes Pathmaster load the chosen terrain or mission data.

If "Use existing Obstacles map" is selected, Pathmaster will not calculate new obstacles and the Max Slope control will be disabled. This option is useful for loading obstacles that were manually edited in a previous scenario. If other terrain maps exist (Soil Mechanics, Scientific Return, etc.), they may be loaded in the same manner.

If "Load existing Waypoints" is selected, Pathmaster will load waypoints for each corresponding explorer that has been added. Waypoint data, if it exists, will only be loaded for explorers created with the Add Explorer button in this menu. For example, if waypoint data for 4 explorers is stored but only 2 explorers are created in this menu, then only the stored waypoint data for the first 2 explorers will be loaded. To load waypoints from a previous mission, simply select the Matlab data file from that mission when opening Pathmaster and this option will appear. Waypoint loading is only available through selecting a Matlab data file when opening Pathmaster.

Close:

A prompt will appear ensuring that the user wants to close the current mission and exit Pathmaster. No data has yet been saved.

START:

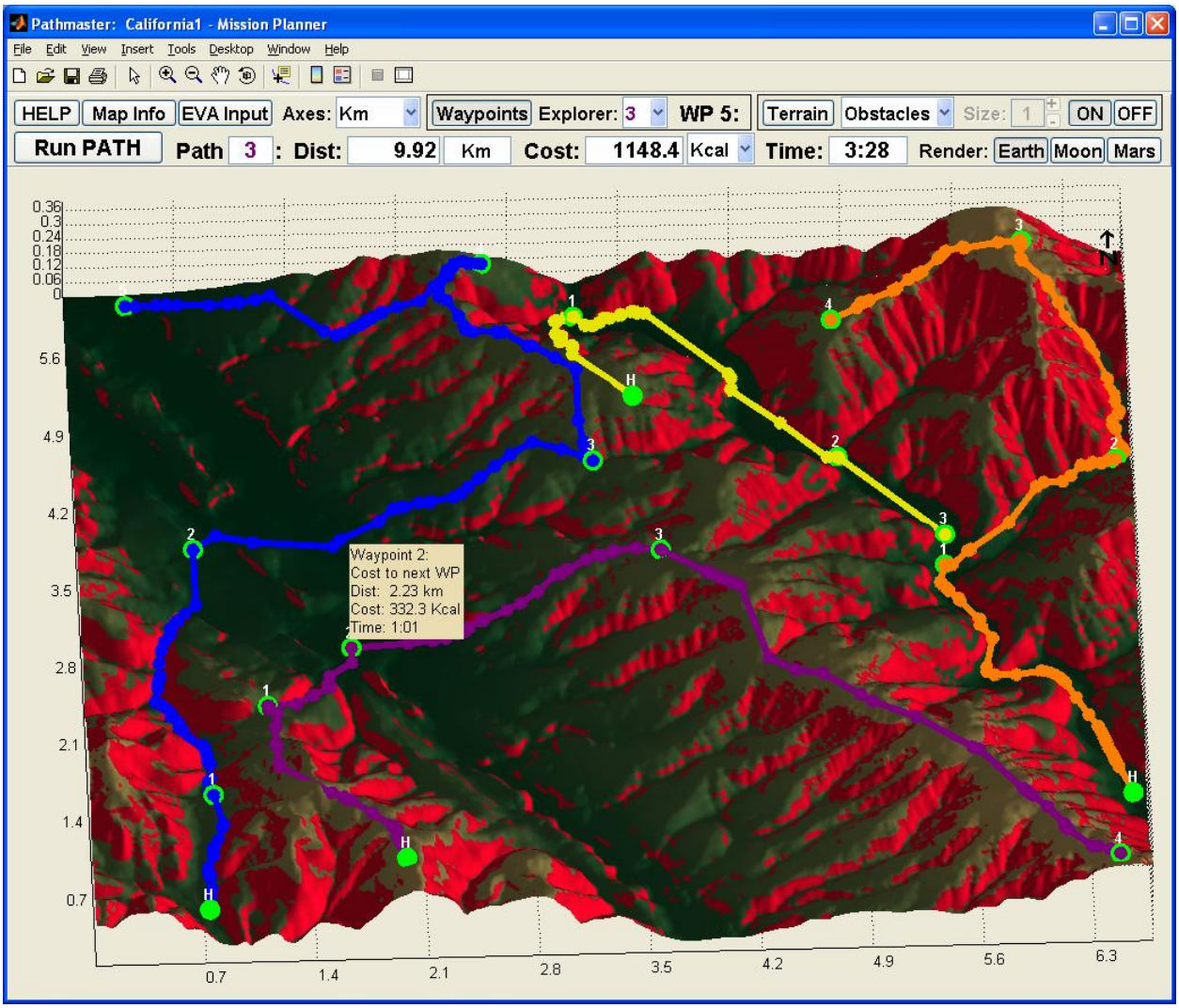
Proceed with the current data. If a mission with the same EVA name exists, a prompt will appear asking if the user would like to overwrite the earlier mission.

Calculating obstacles, writing map files, preparing surface:

After pressing START, Pathmaster goes to work. First, the terrain slopes are calculated via a surface gradient. The obstacles are then identified based upon the value of Max Slope (unless using an existing obstacles map). If the entire map is an obstacle with no traversable terrain, a prompt will appear asking the user to increase the Max Slope. Next, the terrain maps are written to both text and Matlab files. For details on these files, see the File I/O section. These files may be used to conveniently reload the same maps later. Finally, the terrain is rendered in the Mission Planner GUI.

Mission Planner GUI

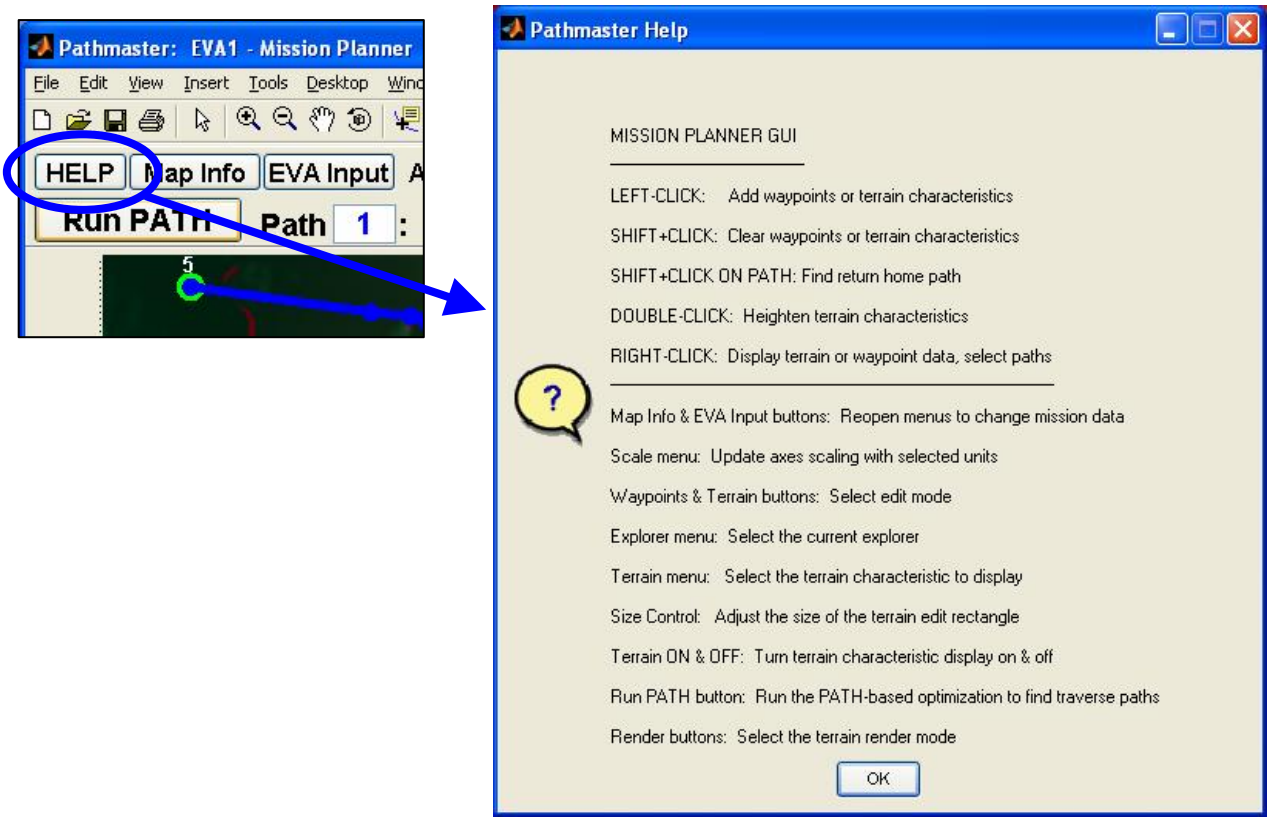
Once the map information and EVA input data have been loaded, the Mission Planner GUI opens. Here, the user may view the terrain, edit mission waypoints, edit terrain characteristics, find traverse paths, and display all mission information in real-time.



Mission Planner GUI, continued

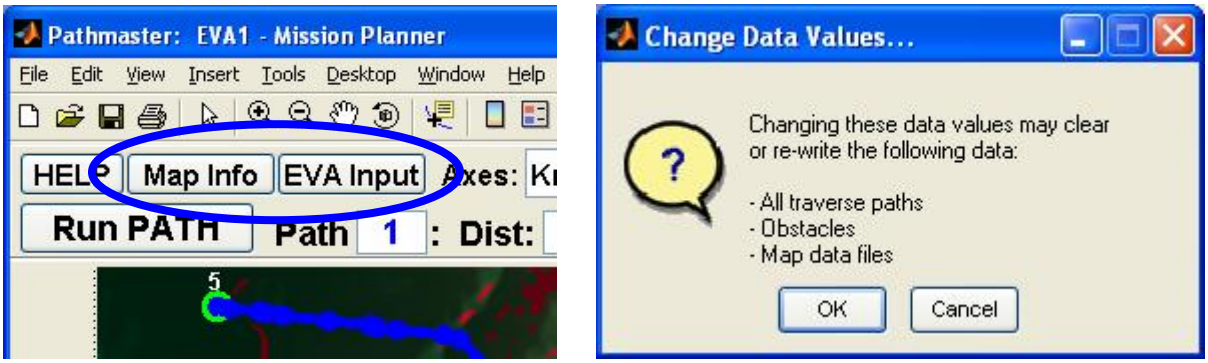
Help Menu:

Pressing the HELP button will open the Help menu, which explains all controls in the Mission Planner GUI.



Menu Buttons:

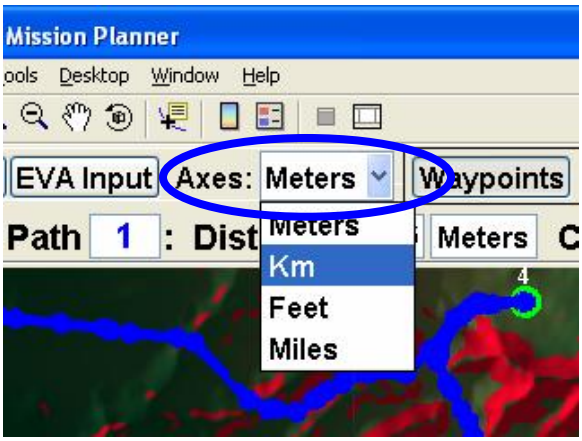
Pressing the Map Info button will reopen the Map Information menu. Likewise, pressing the EVA Input button will reopen the EVA Input menu. Press Continue in either of these menus to return to the Mission Planner GUI. If data values are changed, a prompt will appear warning the user of any data or files which may be cleared or overwritten as a result. Upon pressing OK, the Mission Planner GUI will automatically update to reflect the new data parameters.



Mission Planner GUI, continued

Axes Scaling:

The Axes drop-down menu provides four options for terrain scaling: meters, kilometers, feet, and miles. When a new unit is selected, the surface axes and gridlines will update with new spacing and tick marks to reflect this change. In addition, traverse distances and elevations data will be provided in terms of the new unit.

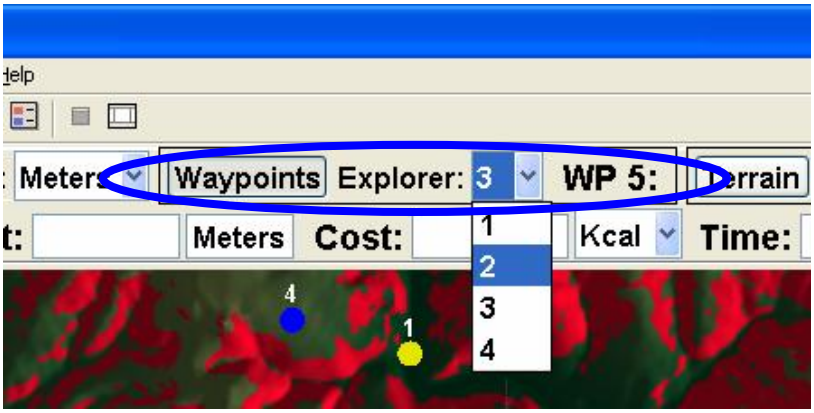


Waypoint Edit Mode:

When the Waypoints button is depressed, waypoint edit mode is active. Select the current explorer with the explorer drop-down menu. Edit the explorer's waypoints by pointing and clicking on the terrain. Waypoints are color-coded for each explorer. The current explorer color will be shown in the Explorer drop-down menu. A small numeral appears above each waypoint indicating the waypoint number, or "H" for starting point or home. On a traverse, waypoints are visited in order beginning with 1.

Left-Click: Left-clicking on the surface will add the next waypoint at that location.

Shift+Click: Holding Shift while clicking anywhere on the terrain will clear the last waypoint for the current explorer.



Mission Planner GUI, continued

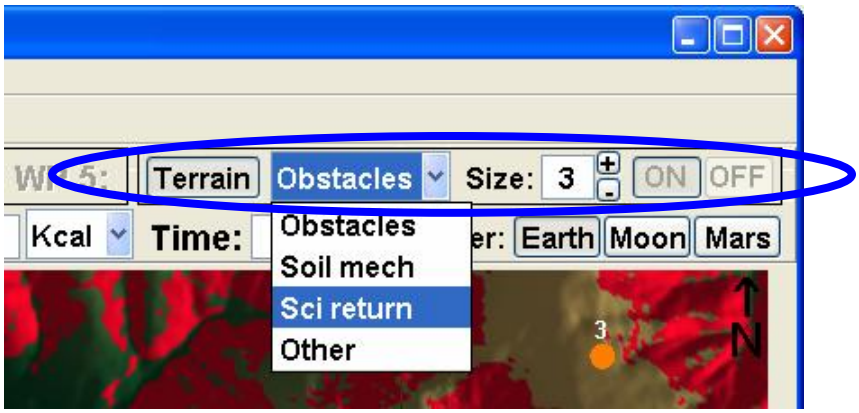
Terrain Edit Mode:

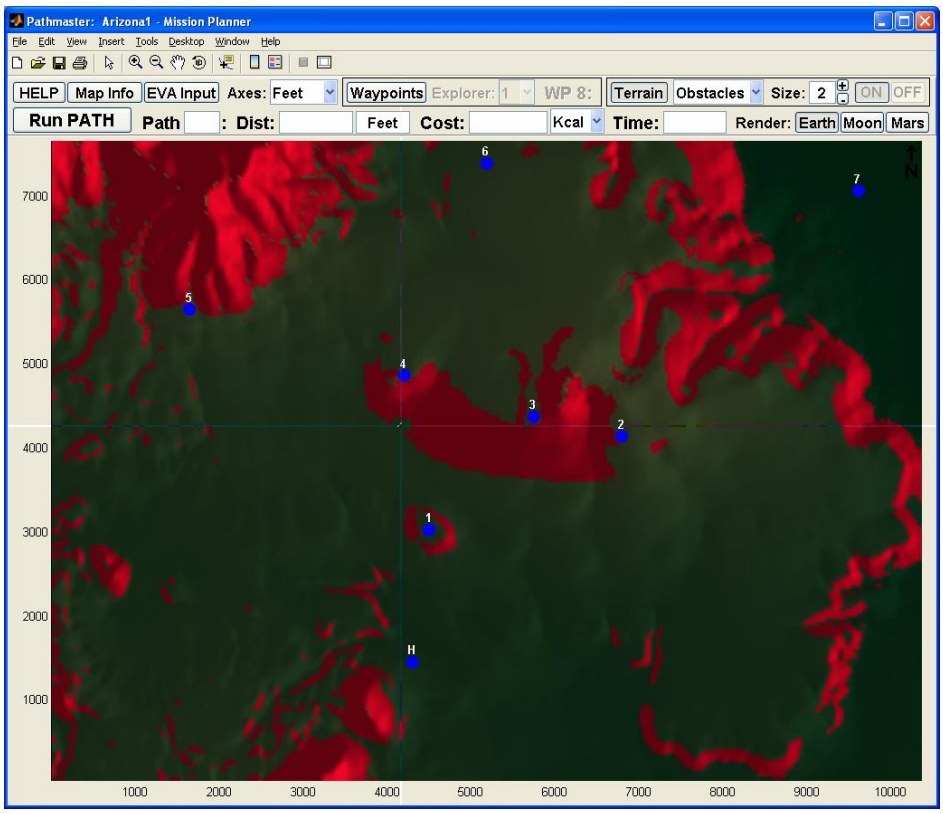
When the Terrain button is depressed, terrain edit mode is active. Select which terrain parameter to edit with the drop-down menu. Edit by pointing and clicking with the commands below. Use the Size control to alter the size of the terrain edit rectangle. The buttons at the side of this field increase and decrease the size, ranging from 0.1 to 10 (the value corresponds to an approximate percentage of the map length). The intuitive functionality is comparable to MS Paint.

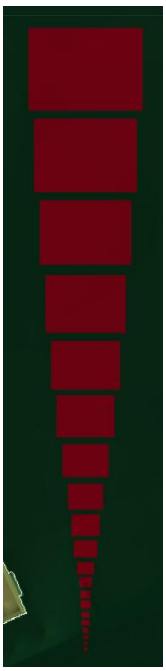
Left-Click: Left-clicking on the surface will add the terrain feature at that location.

Double-Click: Double clicking will heighten the terrain feature (no effect on obstacles).

Shift+Click: Holding Shift while clicking will clear the terrain feature.







Terrain edit sizes, 0.1 - 10

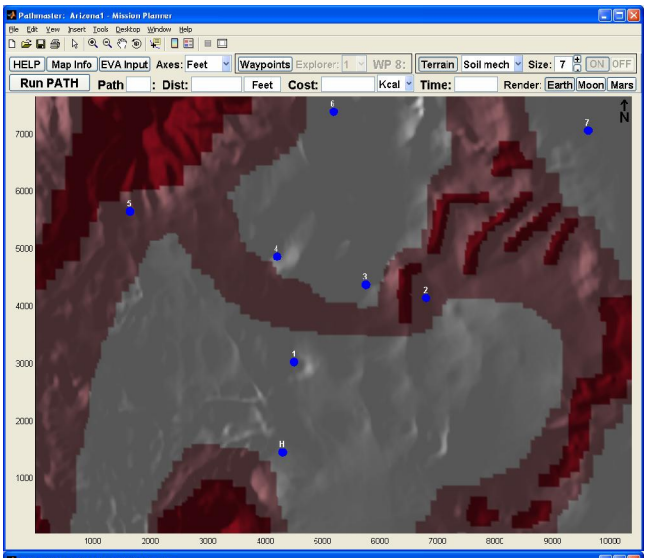
Mission Planner GUI, continued

Terrain Edit Mode, cont'd:

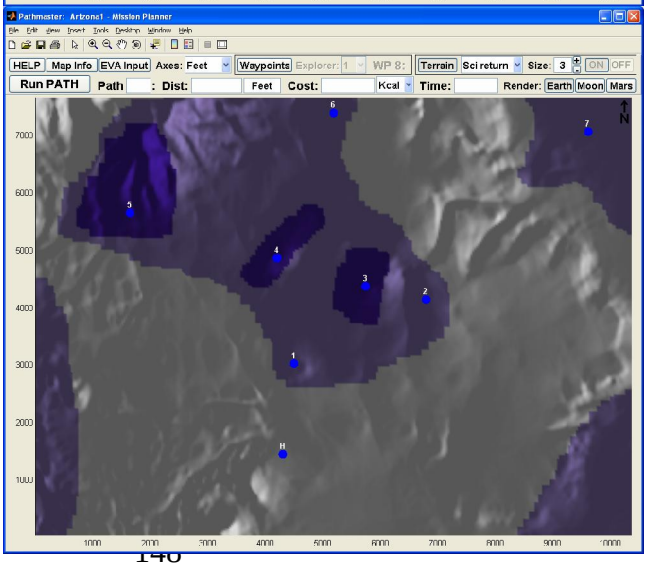
Obstacles: Obstacles are impassable barriers in the terrain, shown in red over the elevation rendering. The initial obstacles are areas where the surface slope is greater than the maximum slope defined in the EVA Input menu (unless an alternate obstacles map was loaded). Obstacle maps have two values: 0 (no obstacle) or 1 (obstacle). An example of editing obstacles is shown on the previous page. Be careful not to cover or enclose a waypoint in obstacles. If this occurs, a warning message will be returned when finding traverse paths.

Soil Mechanics, Scientific Return, Other: Soil mechanics refers to the ease of traversability of a terrain due to the surface characteristics (rocky, sandy, etc). Scientific return refers to the projected scientific gain offered at differing locations. A third map, Other, may be used to characterize an additional terrain feature such as chemical composition, radiosity, or even an additional explorer constraint. Each of these maps accommodate 3 values: 0, 1, or 2. They are set entirely to 0 by default. A Left-Click sets the local terrain to 1, a Double-Click sets it to 2, and holding Shift while clicking resets it back to 0. These arbitrary index values may be fed into the traverse cost functions for determining optimized routes. Examples of editing soil mechanics and scientific return are shown below.

Soil Mechanics:



Scientific Return:



Mission Planner GUI, continued

Traverse Paths:

Pressing the Run PATH button calls the traverse path optimization routine based upon the PATH Java software, and all mission data is saved. A new route will be found for all explorers with a starting point and at least one waypoint that do not already have a path. The traverse paths found are optimized in terms of a cost function. The current cost function minimizes explorer metabolic expenditure along the traverse while avoiding all obstacles. Once finished, color-coded traverse paths are plotted along the terrain. The costs of each traverse are shown in the menu at the top. If UTM positioning is active, Pathmaster also writes text files containing latitude/longitude position coordinates for every point along each traverse to a Traverse_Coordinates folder in the working directory. While finding traverse paths, certain GUI functionality is temporarily disabled.

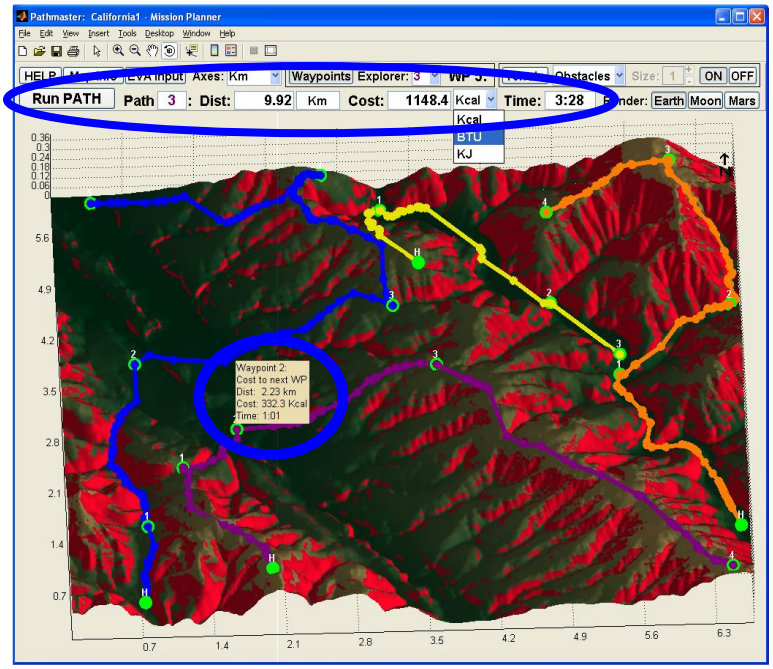
Right-Click: Right-clicking on a traverse path will select that explorer. The corresponding traverse costs will be displayed in the menu at the top, and the nearest waypoint data will be displayed. Continue Right-clicking to cycle through the various waypoint data:
- Cost from start
- Cost from previous waypoint
- Cost to next waypoint
- Cost to end
- Local terrain data

The explorer drop-down menu may always be used to select an explorer as well. Remember, Left-clicking will still edit waypoints or terrain.

Dist: The Dist field displays the total traverse path distance for the selected explorer in the units selected with the Axes drop-down menu.

Cost: The Cost field displays the total explorer metabolic expenditure for the selected explorer along the traverse path. Use the neighboring drop-down menu to select from the following cost display units: kilocalories, BTU, or kilojoules.

Time: The Time field displays the total estimated time to complete a traverse for the selected explorer, in hours and minutes.

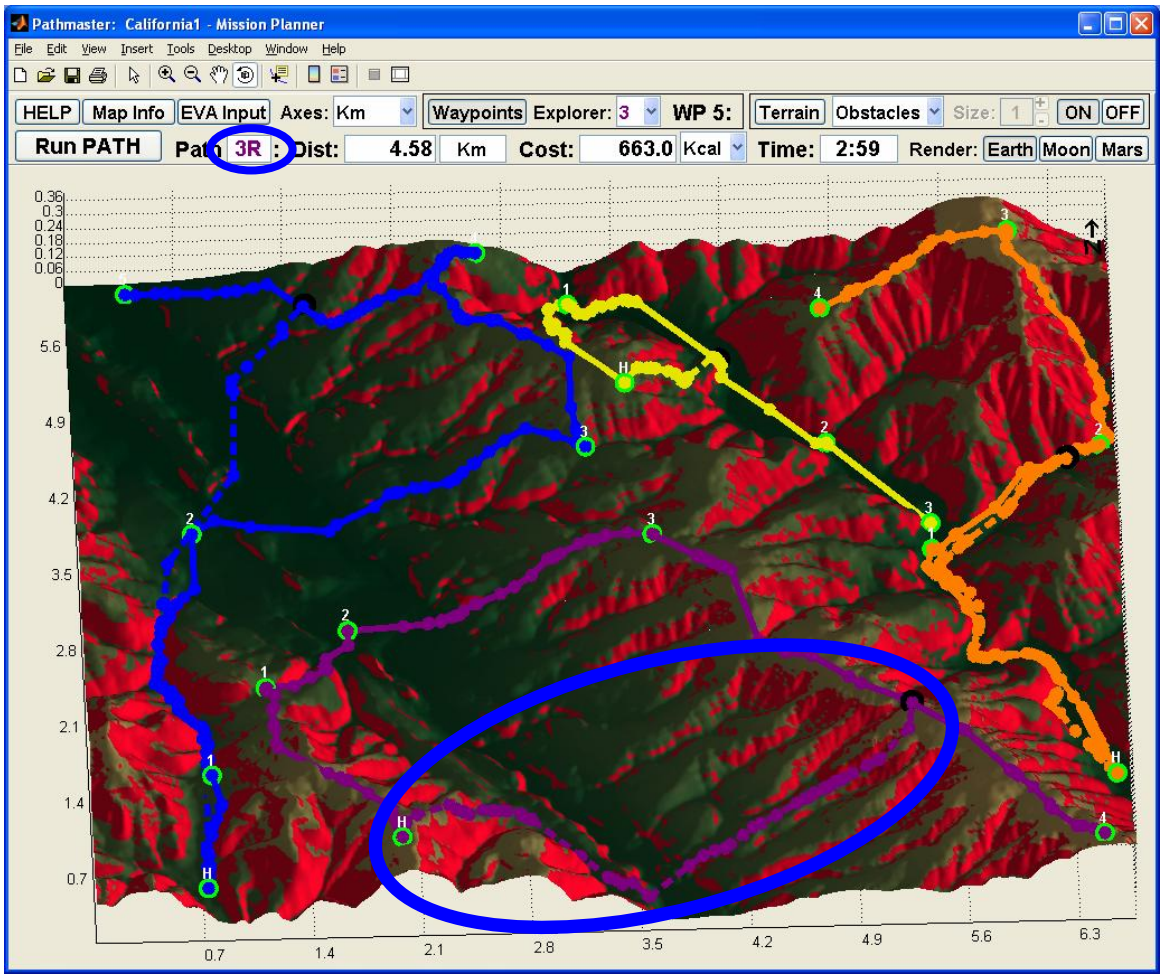


Mission Planner GUI, continued

Return Home Paths:

At any location along a traverse, an explorer may be directed to immediately return to the starting point. Called a "return home path", these special traverses appear as dotted paths. Costs for the return traverse alone are displayed in the menu at the top, with an "R" in the Path field signifying "return home". One return home path may be found per explorer. Creating a new return home path will clear any previous return route for that explorer.

Shift-Click: Holding Shift while clicking on a traverse path will prompt the user whether or not to find a return home path for the corresponding explorer. Pressing Yes will run the traverse optimization routine, and the new route will be calculated and plotted with costs displayed in the menu at the top.

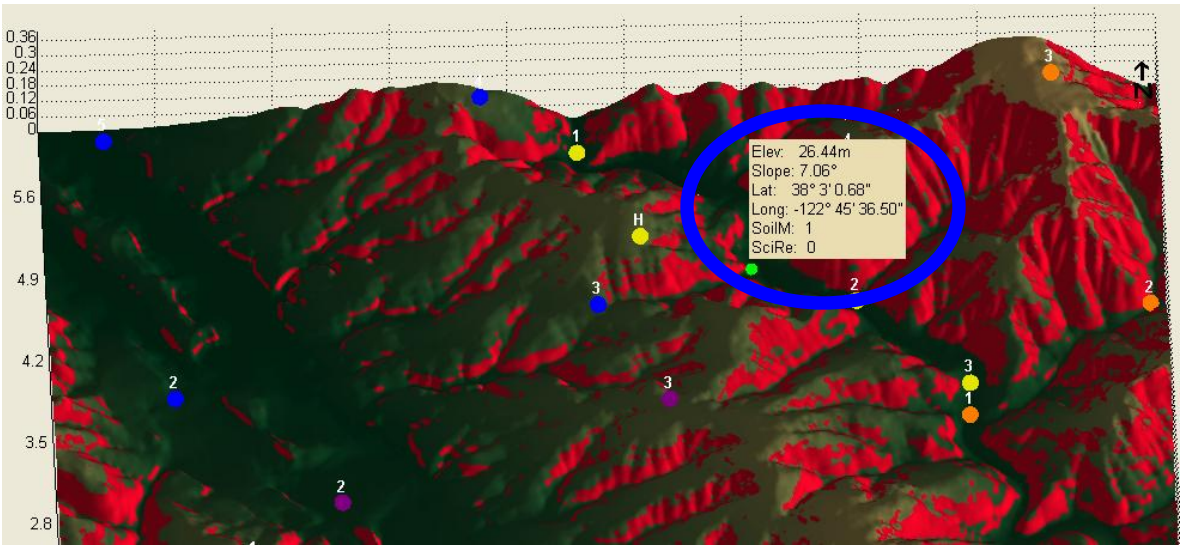


Mission Planner GUI, continued

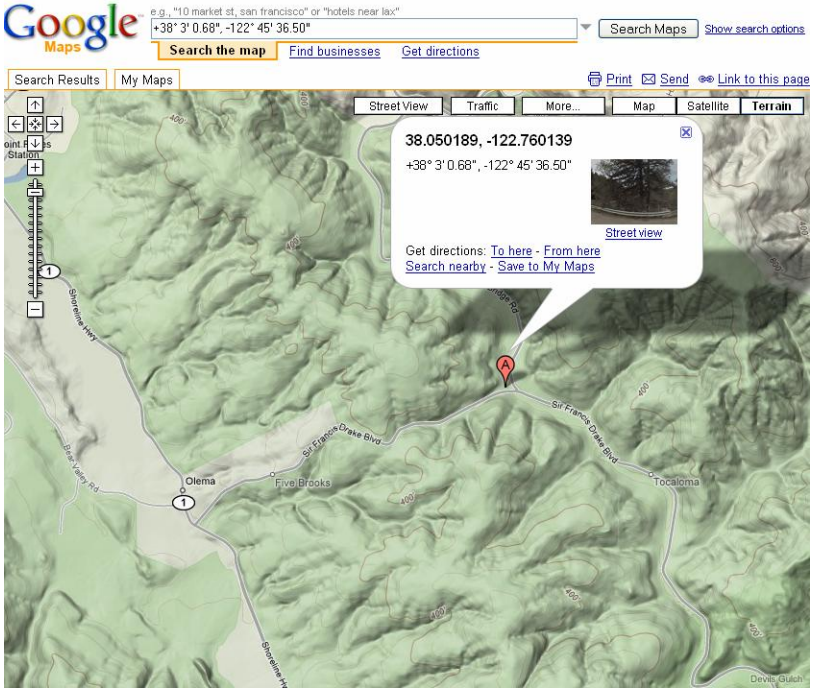
Terrain Data Display:

Local terrain data may be displayed for any point with a click of the mouse. The provided data includes elevation and slope as well as any soil mechanics, scientific return, etc. information if it has been defined. If the data text is not visible, slightly rotate or zoom the terrain to fix this (this is a slightly annoying Matlab bug). Right click on a waypoint to display waypoint data.

Right-Click: Right-clicking on the terrain will display the local terrain data.



If on earth with UTM positioning active, the latitude and longitude of the selected spot will be provided as well. This latitude/longitude positioning feature allows the user to run Pathmaster alongside additional mapping systems such as Google Maps or ArcGIS to supplement terrain knowledge or precisely locate waypoints or terrain features, among other possibilities.

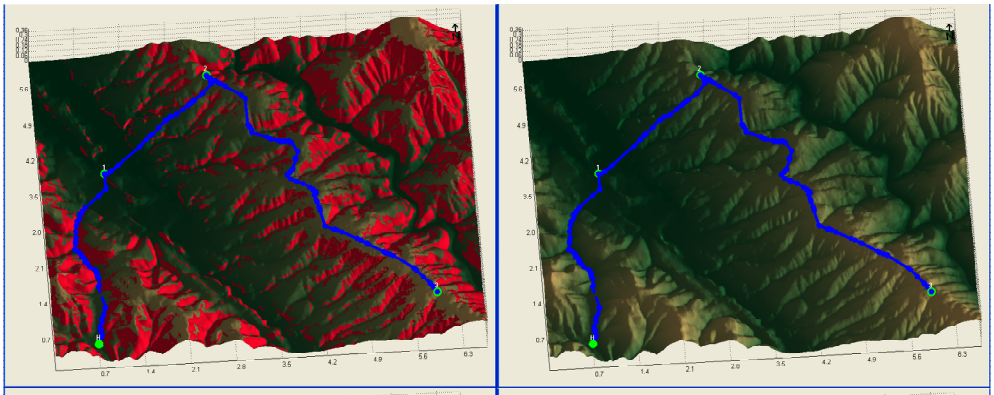


Mission Planner GUI, continued

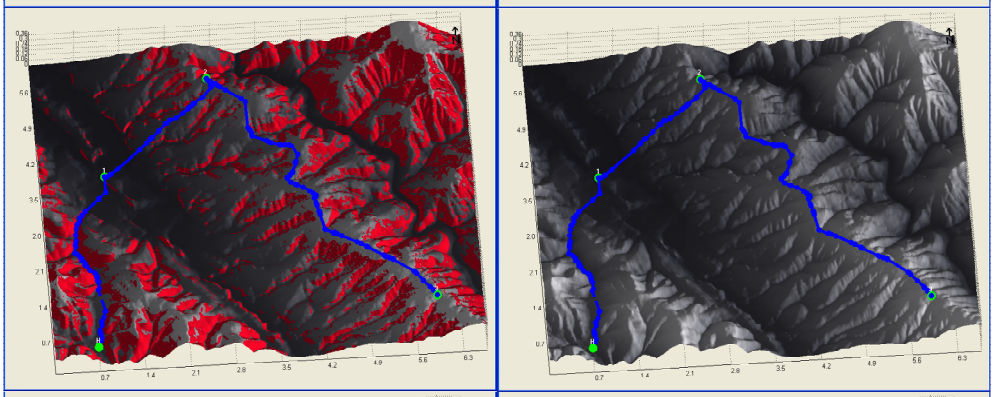
Render Modes:

Pressing the Render buttons will change the terrain surface rendering to mimic the chosen planet: Earth, Moon, or Mars. While the initial render mode is determined by the choice of Planet in the EVA Input menu, changing the render mode affects only the display and does not alter the stored planet or gravity. When in waypoint edit mode, terrain data portrayal may be turned on or off with the terrain toggle buttons just above the Render buttons. Below are the render modes with obstacles on then off.

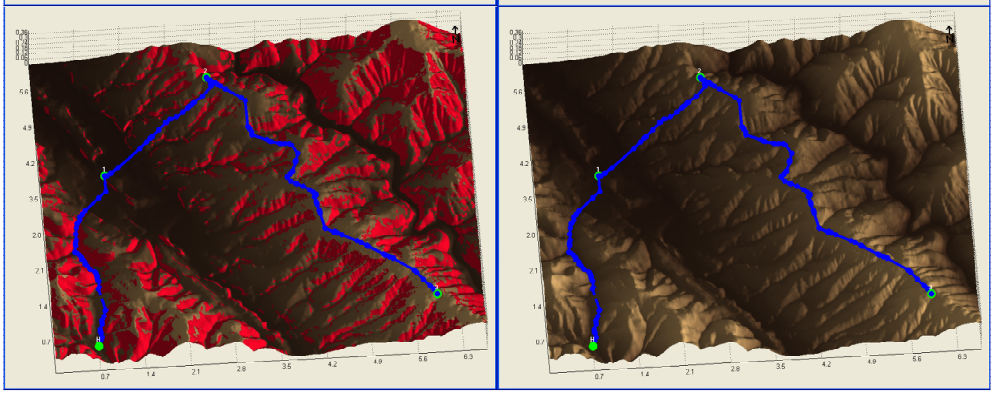
Earth:



Moon:



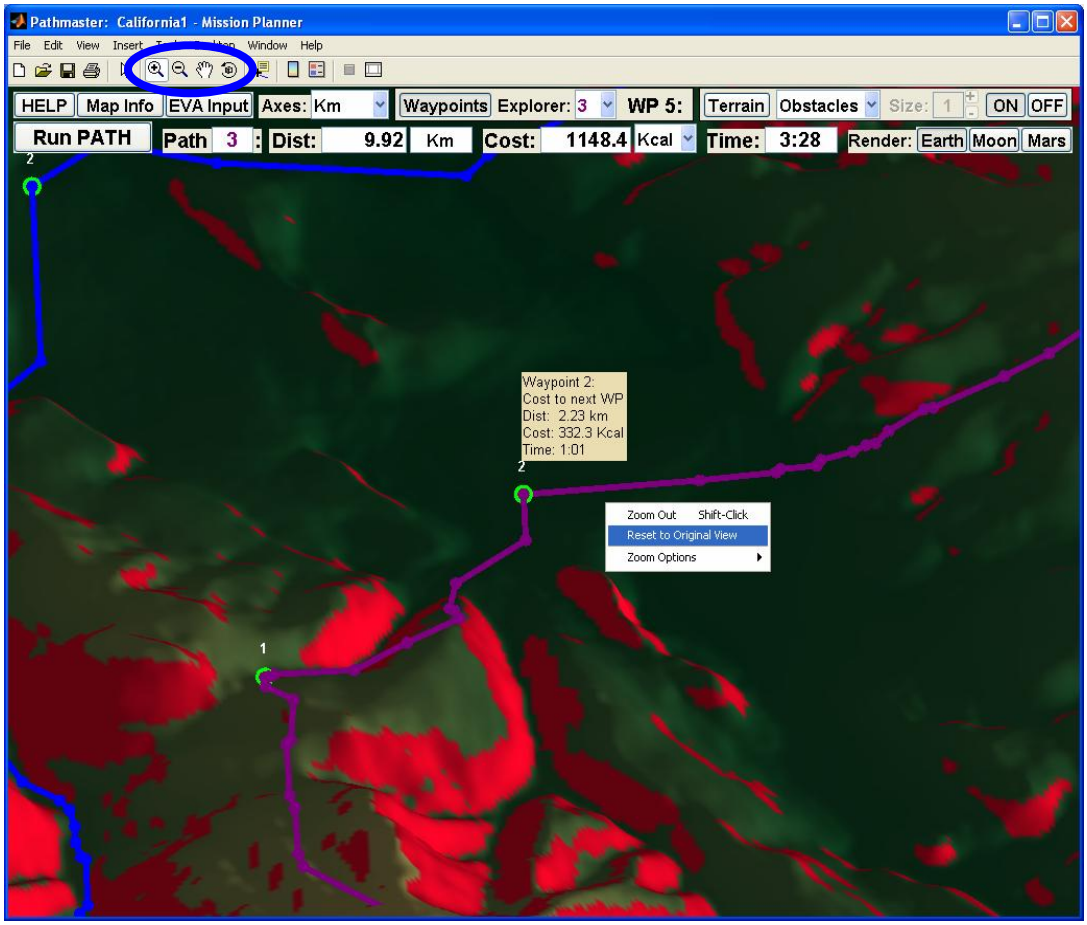
Mars:



Mission Planner GUI, continued

Changing Views:

Use the built-in Matlab menu at the top of the Mission Planner GUI to rotate and zoom the terrain view. To return to the initial aerial view while rotating or zooming, Right-click and select "Reset to Original View". Click on the menu icon again to deactivate rotating or zooming.



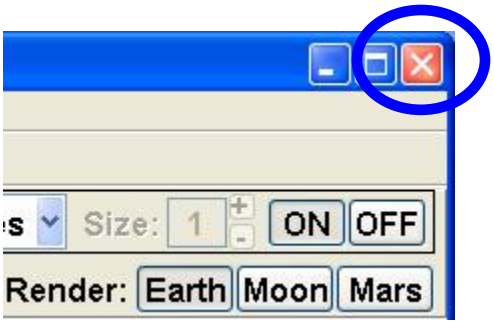
While editing waypoints or terrain, it is best to remain in the initial aerial view. Otherwise, the perceived mouse position may vary due to the projection of a 3D surface on a 2D screen.

It is recommended not to use the other menu options, crossed out above. Editing the plot or changing its format may cause errors in Pathmaster. Mission data is not saved with the Save icon here; it is saved automatically by Pathmaster when the Run PATH button is pressed or before closing the Mission Planner GUI. The built-in data cursor is fully functional, but mostly obsolete in the current release.

Mission Planner GUI, continued

Exiting Pathmaster:

To exit Pathmaster, simply close the Mission Planner GUI.



If any waypoint or terrain edits have been made since last saving (saving occurs when the Run PATH button is pressed), then the prompt below appears. Pressing "Save edits" saves all waypoint and terrain data before exiting Pathmaster. Pressing "Don't save" exits Pathmaster without saving the recent changes. Pressing Cancel returns the user to the Mission Planner GUI.

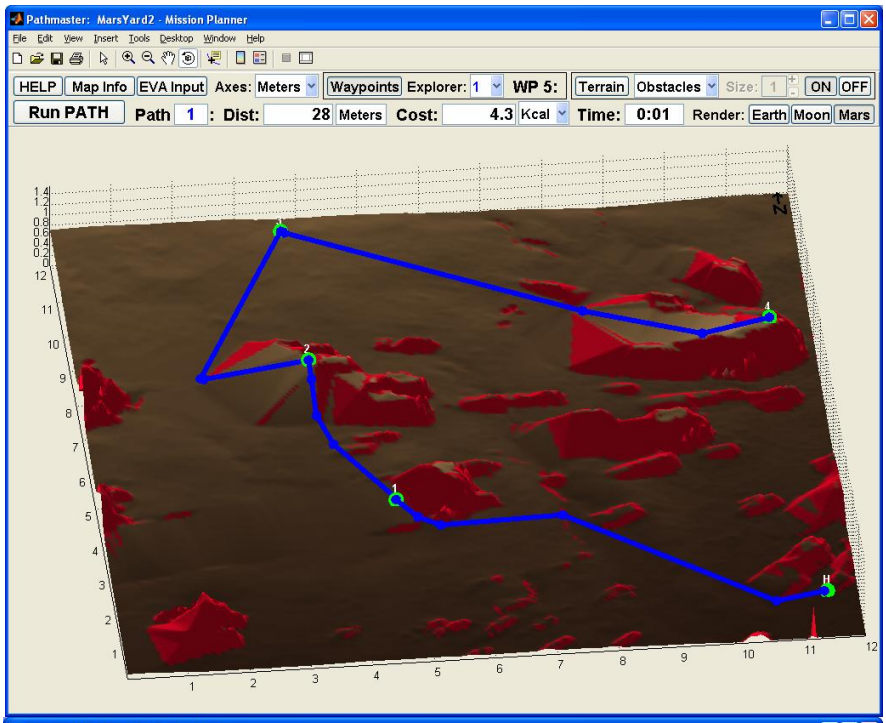
Otherwise if no edits have been made since last saving, then a simple prompt ensuring that the user is finished appears. Pressing Yes closes the Mission Planner GUI and exits Pathmaster.

Sun Illumination

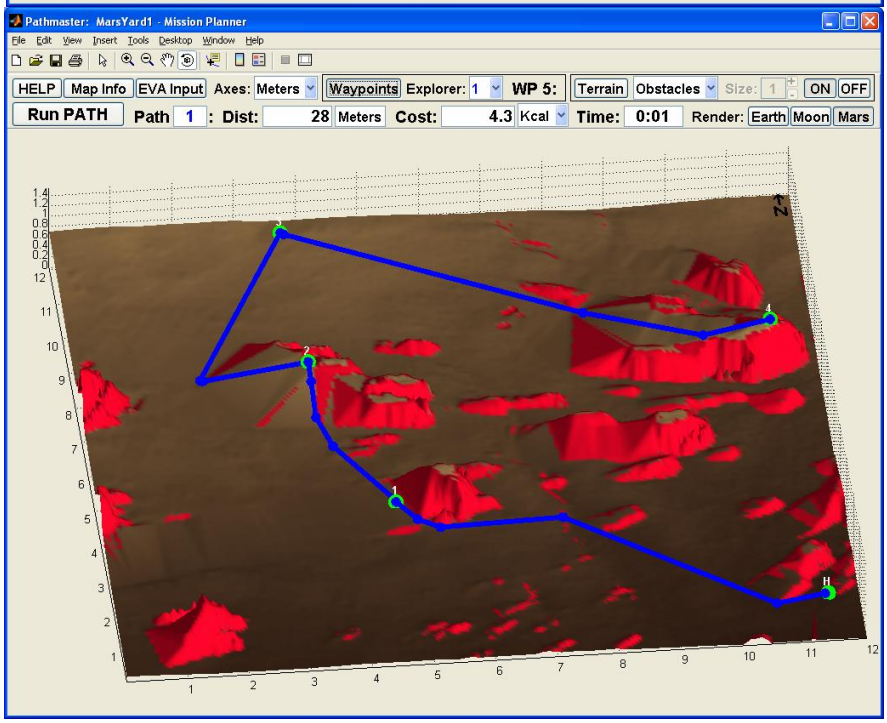
The sun illumination angles are set by the date and time a mission is run, chosen in the EVA Input menu. Pathmaster mimics the current lighting conditions when creating the terrain rendering for the Mission Planner GUI. Lighting display functionality is not applicable when using the 'lite' rendering option. An example of lighting differences is shown below.

Sun illumination of the JPL Mars Yard:

Midnight:



9:30 AM:



Traverse Path Optimization

When the Run PATH button is pressed, Pathmaster determines optimized traverse paths for each explorer that has waypoints defined. This routine employs the A* algorithm with bi-directional searching to individually establish route legs between successive waypoints. Paths are optimized with respect to a cost function. The current cost function used for all explorers derives directly from the PATH software, which minimizes the metabolic expenditure of each explorer along the traverse while avoiding all obstacles. Once a route is established, it is smoothed into distinct line segments via the midpoint line algorithm.

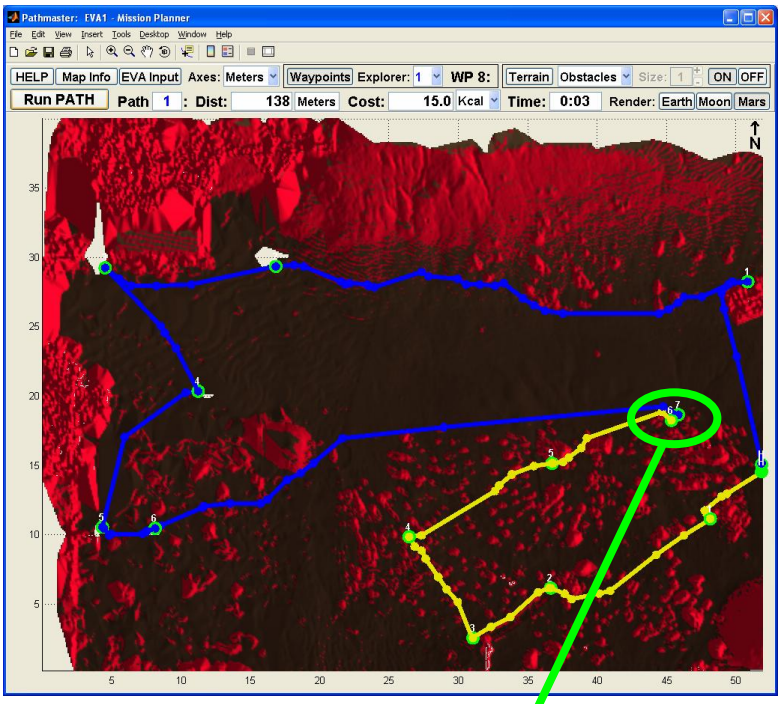
This routine was developed in cooperation with Brandon Johnson. For further details on the optimization and search process, please refer to Joseph Essenburg's thesis (2008), *Mission Planning and Navigation Support for Lunar and Planetary Exploration*, pages 84-90. For a complete description of the PATH software, please refer to Jessica Márquez's thesis (2007), *Human-Automation Collaboration: Decision Support for Lunar and Planetary Exploration*.

External Rendering

Pathmaster writes text files to the Render directory which may be used to feed an external higher fidelity render engine, such as OpenSceneGraph, or even a virtual reality simulation. The Astronaut Rover Mission Simulator (ARMS), developed by Uday Bandaru at Arizona State, provides a virtual mission simulation which is capable of receiving waypoints, traverse paths, and costs from Pathmaster in real-time. This system serves as a prototype heads-up display to aide a traversing astronaut in navigation, site recognition, and handling mission information updating in real time. For details on the external renderer files, see the File I/O section.

Example mission on the JPL Mars Yard and accompanying simulation:

Pathmaster mission plan:



ARMS Simulation:

File I/O

The following provides full details of all data files associated with Pathmaster. They are categorized by their common use. Provided are the characteristic name, directory location, contents, and use for each.

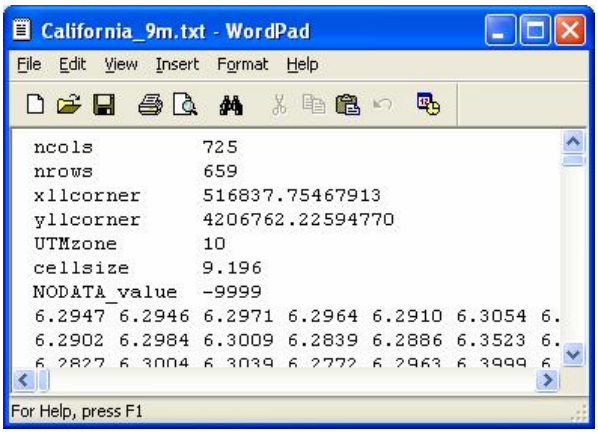
Loading Map Data:

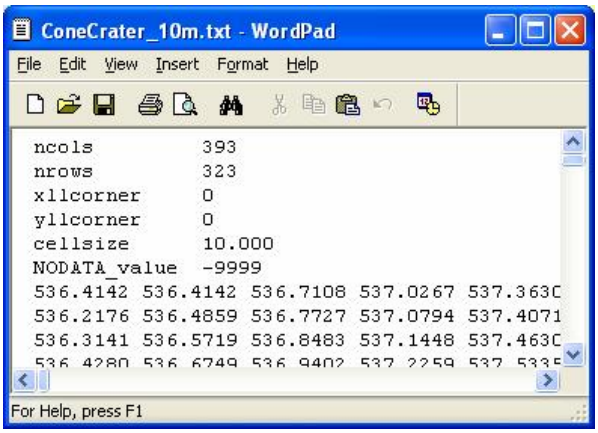
The elevation maps used in Pathmaster are arranged as a rectangular matrix. In terms of (x,y) coordinates, the x-coordinate refers to the column index, and the y-coordinate refers to the row index, starting at (0,0) in the upper-left corner (the y-axis is reverse-ordered). Pathmaster assumes that north is in the upwards direction, and this orientation is necessary for Pathmaster's optional latitude/longitude positioning feature to function properly.

Map text files: **`Terrain_##m.txt`**

Terrain_Maps Directory

Pathmaster compatible map text files consist of 6 or 7 header lines followed by a space-delimited matrix of elevation data points. Each line in the matrix represents a row of data. The header lines provide all map information, and are required besides the optional "`UTMzone`" line. This format is identical to files generated with the ArcGIS PATH interface. Any such file may be loaded when opening Pathmaster. During every run, Pathmaster writes all maps to such text files in the Render directory. Again, these may also be used when opening Pathmaster to re-load mission data.





Line:	Data:
`ncols`	Number of columns in the elevation data matrix
`nrows`	Number of rows in the elevation data matrix
`xllcorner`	UTM "Easting" of the lower left corner of the map
`yllcorner`	UTM "Northing" of the lower left corner of the map
`UTMzone`	(Optional) East-West UTM zone location of the map
`cellsize`	Resolution of the map in meters (horizontal spacing between data points)
`NODATA_value`	Data value entered when no terrain data exists (default is -9999)

The Terrain_Maps directory holds a collection of all existing maps ready for use in Pathmaster. In this directory, the file name describes the general location and resolution of each map.

File I/O, continued

Loading Map Data, cont'd:

Matlab data files: **`EVAname_Data.mat`**

Working Directory (containing pathmaster.m)

Matlab data files are used to store variables from the Matlab workspace. During every run, Pathmaster automatically records all mission data to a Matlab data file in the working directory. Select these files when opening Pathmaster to re-load any mission data. Optional variables in addition to the elevation map provide additional mission parameters. This information will be automatically recognized by Pathmaster so long as the field (variable) name is one of the following and the corresponding value is appropriate:

Field:	Data:
Resolution	Resolution of the map in meters (horizontal spacing between data points)
UTMzone	East-West UTM zone location of the map, numbered 1 to 60
xllcorner	UTM "Easting" of the lower left corner of the map
yllcorner	UTM "Northing" of the lower left corner of the map
NoData	Data value entered in the elevation matrix when no terrain data exists
Obstacles	Obstacle map matrix
SoilMech	Soil mechanics map matrix
SciReturn	Scientific return map matrix
Other	Other map matrix
Waypoints	Explorer waypoint coordinates

Other fields may exist that provide additional mission information (i.e. Pathpoints, MetCost, etc.). These are for archiving and reference, and are not loaded in a new mission.

Traverse Coordinates

Traverse Coordinate files: **`EVAname_Coords#.txt`**

Traverse_Coordinates Folder within the Working Directory

If UTM positioning is active, these files are written to a folder called Traverse_Coordinates within the working directory after finding traverse paths. They provide each explorer's traverse path coordinates in terms of latitude and longitude. The number before the file extension corresponds to the explorer number, and an "R" indicates a return home path.

The first line of these files is: "`Explorer # Lat/Long:`"
After that, each line has the format:
`point# Latitude Longitude`

File I/O, continued

External Rendering

Waypoint files: **`EVAname_Waypoints#.txt`**

Render Directory

These files provide each explorer's waypoint coordinates in terms of (x,y) matrix coordinates, with (0,0) at the lower-left corner (these coordinates are written with the y-axis regularly ordered, opposite the reverse-ordering used internally in Pathmaster). They are written after pressing the Run PATH button in the Mission Planner GUI. The number before the file extension corresponds to the explorer number, and an "R" indicates a return home path.

Each line of these files has the format:

```
way# X Y
```

Traverse files: **`EVAname_Traverse#.txt`**

Render Directory

These files provide each explorer's traverse path coordinates in the same manner as the waypoint files described above. They are written after finding traverse paths.

Each line of these files has the format:

```
path# X Y
```

Cost files: **`EVAname_Costs#.txt`**

Render Directory

These files provide each explorer's traverse costs listed cumulatively for each traverse path point written in the corresponding traverse file described above. The costs are listed as distance in meters, elapsed time in seconds, and metabolic expenditure in BTU. They are written after finding traverse paths. File numbering carries the same format as those above.

Each line of these files has the format:

```
cost# Cum-Distance(m) Cum-Time(sec) Cum-MetCost(BTU)
```

APPENDIX C: MATLAB CODE FOR PATHMASTER

```
%PATHMASTER   Version 7.5
% Joe Essenburg - Last edited August 28, 2008
% Mission planning interface employing a PATH-based optimization
%
% PATHMASTER alone initializes a prompt for the user to load elevation data
%  from file. Elevation text files or Matlab data files may be used.
%
% PATHMASTER(ELEVMAP) loads the matrix ELEVMAP as the elevation data.
%
% PATHMASTER(...,'lite') calls the 'lite' option, which uses simpler
%  surface rendering to speed plotting time and prevent problems on some
%  machines.
%
%  Map information and all other mission data are entered on the following
%  menus. Pathmaster will then open a GUI for point-and-click waypoint
%  editing, terrain editing, traverse path optimization, and displaying
%  those paths along with all cost data.
%  Data files are written for an independent render engine.
%
%  Several mission scenarios may be loaded into multiple instances of
%  PATHMASTER simultaneously.


% Pathmaster is a single function that iteratively calls itself with three
% parameters: PROGRESS, SELECT, and DATA. The parameter PROGRESS determines
% which section of code is to be executed, and the parameter SELECT
% determines which sub-section or option to execute when applicable. This
% is accomplished through SWITCH constructs, with the main progress switch
% beginning on Line 178. DATA is a structure holding all necessary
% application and mission data, which is passed and updated in each
% iterative call to pathmaster.
% Any open pathmaster GUI contains its current DATA structure in the
% 'UserData' property, and all GUIs are shielded from the command line.
% This allows multiple instances of pathmaster to be run simultaneously.

function pathmaster(Progress,Select,Data)
%% ************ CHECK FUNCTION CALL & LOAD ELEVATION MAP ******************
if nargin <= 2  % This section only runs on the initial call to pathmaster
switch nargin
    case 0
        calldata = 'LoadFromFile'; Data.Lite = '';
    case 1
        if ~ischar(Progress)
            calldata = 'Progress'; Data.Lite = '';
        elseif strcmp(Progress,'lite')
            calldata = 'LoadFromFile'; Data.Lite = '''lite''';
        else
            calldata = 'error';
        end
```

```
    case 2
        params = {'Progress','Select','error'}; Data.Lite = '''lite''';
        calldata = params{find(strcmp('lite',{Select,Progress,'lite'}),1)};
end
mfilepath = mfilename('fullpath');
cd(mfilepath(1:end-10))  % Change directory to that containing pathmaster
[Data.Obst,Data.Soil,Data.SciR,Data.Othe,Data.hasWP] = deal(false);%Terrain
cost map
                                                                    %boolean
values
switch calldata
    case 'LoadFromFile'  % Load elevation map from file
        [datafile,filepath] = uigetfile({'*.mat;*.txt',...
                                'Matlab data files or Elevation text
files'},...
                                'Load Elevation Data...');
        if ~datafile, return, end  % Exit pathmaster

        if strcmp(datafile(end-3:end),'.txt') % Load from Elevations text
file
            Mapid = fopen([filepath,datafile],'r');
            if ~strcmp(fscanf(Mapid,'%s',1),'ncols') % Check if proper file
                message = ['Incorrect data or data type:\n\n',...
                           'Text file must be a PATH Elevations
file.\n\n',...
                           'Example:  "EVA name"_Elevations.txt'];
                waitfor(warndlg(sprintf(message),'Load Elevation Data'))
                eval(['pathmaster(',Data.Lite,')'])  % Restart
                return
            end
            frewind(Mapid)
            Scandata1 = textscan(Mapid,'%*[^ ]%n',4);  % Read 4 data values
            hasUTM = strcmp(fscanf(Mapid,'%s',1),'UTMzone');  % UTM check
            fseek(Mapid,-8+hasUTM,'cof');
            Scandata2 = textscan(Mapid,'%*[^ ]%n',2+hasUTM); % Rest of data
            Mapdata = [Scandata1{1};Scandata2{1}];
            xll = Mapdata(3);
            yll = Mapdata(4);
            utm = hasUTM*round(Mapdata(5));  % Nonzero if hasUTM
            res = Mapdata(end-1);
            ndt = Mapdata(end);
            Elevmap = dlmread([filepath,datafile],'',...  % Read elevations
                              [6+hasUTM 0 Mapdata(2)+5+hasUTM Mapdata(1)-1]);
            fclose(Mapid);           % Check for matching terrain cost maps
            for tmap = {'Obstacles' 'SoilMech' 'SciReturn' 'Other'}
                try TCM.(tmap{1}) = dlmread([filepath,datafile(1:max(end-
14,1)),...
                      tmap{1},'.txt'],'',[6+hasUTM 0 Mapdata(2)+5+hasUTM
Mapdata(1)-1]);
                  Data.(tmap{1}(1:4)) =
all(size(TCM.(tmap{1}))==size(Elevmap));
                catch  %#ok<CTCH>
                end
            end
```

```
        elseif strcmp(datafile(end-3:end),'.mat') % Load from Matlab data
file
            Mapdata = load([filepath,datafile]);
            Vars = sort(fieldnames(Mapdata));
            if length(Vars) == 1
                Elevmap = Mapdata.(Vars{1});
            elseif length(Vars) >= 2
                elevi = listdlg('Name','Load Elevation Data',...
                                'ListString',Vars,...
                                'SelectionMode','single',...
                                'ListSize',[182 67],...
                                'PromptString',...
                                {'Please select the field',...
                                 'containing Elevation data:'});
                if isempty(elevi)
                    eval(['pathmaster(',Data.Lite,')'])  % Restart
                    return
                end
                Elevmap = Mapdata.(Vars{elevi});
            end
            Check = whos('Elevmap');
            if isempty(Vars) || ~strcmp(Check.class,'double') ||
min(Check.size)<2
                message = ['Incorrect data or data type:\n\n',...
                           'Input must be an elevation map matrix.'];
                waitfor(warndlg(sprintf(message),'Load Elevation Data'))
                eval(['pathmaster(',Data.Lite,')'])  % Restart
                return
            end
            i=1;  mapinfo = {1,0,0,0,-9999};  % Default map values
            for mv =
{'Resolution','xllcorner','yllcorner','UTMzone','NoData'}
                if any(strcmp(mv{1},Vars))  % Check for existing map values
                    mapval = Mapdata.(mv{1});
                    Check = whos('mapval');
                    if strcmp(Check.class,'double') && all(size(mapval)==1)
                        mapinfo{i} = mapval;
                    end
                end
                i = i+1;
            end
            [res,xll,yll,utm,ndt] = mapinfo{:};  % Set map values
            for tmap = {'Obstacles' 'SoilMech' 'SciReturn' 'Other'}
                if any(strcmp(tmap{1},Vars)) %Check for matching terrain cost
maps
                    [Termap,TCM.(tmap{1})] = deal(Mapdata.(tmap{1}));
                    Check = whos('Termap');
                    Data.(tmap{1}(1:4)) =
any(strcmp(Check.class,{'double','logical'})) &&...
                                           all(size(Termap)==size(Elevmap));
                end
            end
            if any(strcmp('Waypoints',Vars)) % Check for existing waypoints
                [Wpts,Data.LoadWaypoints] = deal(Mapdata.Waypoints);
                Check = whos('Wpts');
                Data.hasWP = strcmp(Check.class,'cell') && Check.size(2)==2;
            end
```

```
        else                                                % Incorrect filetype
            message = ['Incorrect file type:\n\n',...
                       'Elevation data file must be a\n',...
                       'Matlab (.mat) or Text (.txt) file.'];
            waitfor(warndlg(sprintf(message),'Load Elevation Data'))
            eval(['pathmaster(',Data.Lite,')'])  % Restart
            return
        end

    case {'Progress' 'Select'}  % Elevation map entered as an argument
        res = 1;  xll = 0;  yll = 0;  utm = 0;  ndt = -9999;  % Defaults
        Elevmap = eval(calldata);
        Check = whos('Elevmap');
        if ~strcmp(Check.class,'double') || min(Check.size)<2 ||
ndims(Elevmap)~=2
            message = ['Incorrect input argument:\n\n',...
                       'Argument must be an elevation map matrix.'];
            warndlg(sprintf(message),'Open Pathmaster')
            return  % Exit pathmaster
        end

    otherwise  % Unrecognized parameter
        disp('Error: Pathmaster only accepts ''lite'' as a parameter option')
        return  % Exit pathmaster
end
Data.Elev = true;
Progress = 'Initialize';    % Elevation map etc. loaded, go to 'Initialize'
end

switch Progress % This switch determines which code to execute in each call
%% ************ DEFAULT DIRECTORIES & DATA INITIALIZATION *****************
case 'Initialize'
Data.Work_dir = pwd;            % Working directory containing pathmaster.m
macpc = {'/','C:\'};
Data.Render_dir = [macpc{1+ispc},'Content'];  % Root for render engine
for dir = {Data.Render_dir,
[Data.Work_dir,macpc{1+ispc}(end),'Traverse_Coordinates']}
    if ~exist(dir{1},'file'), mkdir(dir{1}), end
end
Elevmap(Elevmap==ndt) = NaN;    % Recognize "no data" values
Data.Elevations = Elevmap;      % Elevation data, loaded in the section above
[Data.Rows,Data.Cols] = size(Elevmap);
Data.Resolution = res;          % Map values assigned in section above
Data.xllcorner = xll;
Data.yllcorner = yll;
Data.UTMzone = utm;
Data.NoData = -9999;            % Value entered in saved maps for no data
Data.EVAname = 'EVA1';          % Default mission name
Data.Explorers{1} = 'Astronaut';%Explorer type: Astronaut or Rover
Data.NumExp = 1;                % Number of explorers
Data.MaxSlope = 15;             % Maximum traversable slope
Data.Planet = 1;                % 1: 'earth', 2: 'moon', 3: 'mars'
Data.Weight(1) = 120;           % Suited explorer mass in kg
Datenow = datevec(now);         % Default time is right now
Data.Month = Datenow(2);
```

```
Data.Day = Datenow(3);
Data.Year = Datenow(1);
Data.Hour = Datenow(4);
Data.Minute = Datenow(5);
Data.TimeZone = 8;  % Use 8 for EST, 6 for CST, 14 for MST, 16 for PST
Data.Scrsize = get(0,'ScreenSize');
for tmap = {'Obstacles' 'SoilMech' 'SciReturn' 'Other'}
    if Data.(tmap{1})(1:4)) %Existing terrain cost maps found in section above
        TCM.(tmap{1}) = round(max(TCM.(tmap{1}),0)); %Clean up existing maps
        Data.(tmap{1}) =
min(TCM.(tmap{1}),1+3*(~strcmp(tmap{1},'Obstacles')));
    end
end
Data.Callback = false;  % Indicates if call is made from Mission Planner GUI
Data.Path = false;      % Indicates if command to run PATH was made
pathmaster('Map',0,Data)                                    % Call to 'Map'

%% ************ MAP INFO MENU ********************************************
case 'Map'
switch Select
    case 0  % Initialize Map Information menu
        MapIn = figure('Name','Pathmaster',...
                'Position',[round(Data.Scrsize(3)/4) round(Data.Scrsize(4)/3)
340 300],...
                'Color',[.92549 .913725 .847059],...
                'Resize','off',...
                'IntegerHandle','off',...
                'DockControls','off',...
                'MenuBar','none',...
                'NumberTitle','off',...
                'CloseRequestFcn','pathmaster(''Map'',''Close'',[])');

        utm = {'default',[1 1 1],'n/a',abs(Data.UTMzone)};
      % UIcontrols: {Handle,Style,Position,String,Value,...
      %              HorizontalAlignment,FontSize,BackgroundColor,Callback}
        Mui = {'na','text',[15 260 310 30],'Elevation Map Information',1,...
                    'center',16,[.58824 .96078 .86275],'';...
               'na','text',[17 220 306 25],sprintf('The entered map is %d x
%d',...
                    Data.Rows,Data.Cols),1,'center',16,[1 1 1],'';...
               'na','text',[15 175 155 25],'Map Resolution:',1,...
                    'left',16,'default','';...
               'ResolutionH','edit',[171 174 87
27],sprintf('%.3f',Data.Resolution),...
                    1,'center',18,[1 1 1],'pathmaster(''Map'',1,[])';...
               'na','text',[260 175 65 25],'meters',1,...
                    'left',16,'default','';...
               'na','text',[40 130 150 20],'UTM Zone (1-60):',1,...
                    'left',14,'default','';...
               'UTMzoneH','edit',[195 126 40 25],utm{3+(Data.UTMzone~=0)},...
                    1,'center',14,[1 1 1],'pathmaster(''Map'',4,[])';...
               'UTMnsH','popup',[238 150 57 1],{'North','South'},...

1+(Data.UTMzone<0),'left',10,utm{1+(Data.UTMzone~=0)},'';...
               'na','text',[15 100 135 15],'Lower-left X-coordinate:',1,...
                    'left',10,'default','';...
```

```
                'xllcornerH','edit',[15 80 145
20],sprintf('%.5f',Data.xllcorner),1,...

'center',12,utm{1+(Data.UTMzone~=0)},'pathmaster(''Map'',2,[])';...
                'na','text',[180 100 135 15],'Lower-left Y-coordinate:',1,...
                    'left',10,'default','';...
                'yllcornerH','edit',[180 80 145
20],sprintf('%.5f',Data.yllcorner),1,...

'center',12,utm{1+(Data.UTMzone~=0)},'pathmaster(''Map'',3,[])';...
                'na','push',[15 16 144 50],'Continue',0,...

'center',18,'default','pathmaster(''Map'',''Continue'',[])';...
                'NewmapH','push',[161 16 106 50],'New map...',0,...

'center',14,'default','pathmaster(''Map'',''NewMap'',[])';...
                'na','push',[269 16 56 50],'Quit',0,...
                    'center',14,'default','pathmaster(''Map'',''Close'',[])'};
        for n = 1:size(Mui,1)          % Create the GUI using the info above
            Data.(Mui{n,1}) = uicontrol('Style',Mui{n,2},...
                                'Position',Mui{n,3},...
                                'String',Mui{n,4},...
                                'Value',Mui{n,5},...
                                'HorizontalAlignment',Mui{n,6},...
                                'FontSize',Mui{n,7},...
                                'BackgroundColor',Mui{n,8},...
                                'CallBack',Mui{n,9});
        end
        Data.UTMzone = abs(Data.UTMzone); % < 0 UTM interpreted as South Hem
        Data.Do = {};  % Tasks to perform when called from Mission GUI
        if Data.Callback
            set(Data.NewmapH,'Enable','off')
            if Data.Planet ~= 1

set([Data.UTMzoneH,Data.xllcornerH,Data.yllcornerH,Data.UTMnsH],...
                    'Enable','off')
            end
        end
        set(MapIn,'UserData',Data,...                % Set GUI to store Data
                'HandleVisibility','callback')
                                                              % GUI controls
    case {1 2 3 4}  % Edit data entries
        MapIn = gcf;
        Data = get(MapIn,'UserData');
        Vars = {'Resolution', .001, 199.999, '%.3f';...
%{Variable,min,max,format}
                'xllcorner', 0, 9999999.99999, '%.5f';...
                'yllcorner', 0, 9999999.99999, '%.5f';...
                'UTMzone', 0, 61, '%.0f'};
        Newvalue = sscanf(get(Data.([Vars{Select,1},'H']),'String'),'%f');
        if ~isempty(Newvalue)  % If value too small or big, set to min/max
            Data.(Vars{Select,1}) =
min(max(Newvalue,Vars{Select,2}),Vars{Select,3});
            Data.Do = {Data.Do{:},'NewMaps','ClrPaths'};
            if Select==1  % Resolution edit ==> Don't use existing Obs
                Data.Obst = false;
                Data.Do = {Data.Do{:},'Scale','Obs'};
```

```
            end
        end
        set(Data.([Vars{Select,1},'H']),'String',...
            sprintf(Vars{Select,4},Data.(Vars{Select,1})))
        if Select==4  % UTM zone edit
            if Data.UTMzone >= 1 && Data.UTMzone <= 60
                Data.UTMzone = round(Data.UTMzone);
                set([Data.xllcornerH,Data.yllcornerH,Data.UTMnsH],...
                    'BackgroundColor',[1 1 1])
            else
                Data.UTMzone = 0;
                set(Data.UTMzoneH,'String','n/a')
                set([Data.xllcornerH,Data.yllcornerH,Data.UTMnsH],...
                    'BackgroundColor','default')
            end
        end
        set(MapIn,'UserData',Data)

    case {'Continue' 'NewMap' 'Close'}  % Buttons
        MapIn = gcf;
        Data = get(MapIn,'UserData');
        Data.UTMzone = Data.UTMzone*(3-2*get(Data.UTMnsH,'Value')); % +N,-S
        delete(MapIn)
        if Data.Callback
            task = {Data.Do,{}};                          % Call to 'Update'
            pathmaster('Update',task{1+strcmp(Select,'Close')},Data)
        elseif strcmp(Select,'Continue')
            pathmaster('Input',0,Data)                     % Call to 'Input'
        elseif strcmp(Select,'NewMap')
            eval(['pathmaster(',Data.Lite,')'])  % Restart
        end
end

%% ************ EVA INPUT MENU ********************************************
case 'Input'
switch Select
    case 0  % Initialize EVA Input menu
        chkbx =
25*(Data.Obst+(Data.Soil+Data.SciR+Data.Othe+Data.hasWP)*~Data.Callback);
        PathIn = figure('Name','Pathmaster',...
                'Position',[round(Data.Scrsize(3)/4) round(Data.Scrsize(4)/8)
400 525+chkbx],...
                'Color',[.92549 .913725 .847059],...
                'Resize','off',...
                'IntegerHandle','off',...
                'DockControls','off',...
                'MenuBar','none',...
                'NumberTitle','off',...
                'CloseRequestFcn','pathmaster(''Input'',''Close'',[])');

        macpc = {'/','\'};    % Append / or \
        Data.Work_dir = regexprep(Data.Work_dir,'\w$',['$0',macpc{1+ispc}]);
        Data.Render_dir =
regexprep(Data.Render_dir,'\w$',['$0',macpc{1+ispc}]);
      % UIcontrols: {Handle,Style,Position,String,Value,...
      %              HorizontalAlignment,FontSize,BackgroundColor,Callback}
```

```
        Gui = {'na','text',[15 485+chkbx 370 30],'Mission Planner EVA
Input',...
                   1,'center',16,[.58824 .96078 .86275],'';...
               'na','text',[15 440+chkbx 135 25],'Name of EVA:',1,...
                   'left',16,'default','';...
               'EVAnameH','edit',[155 440+chkbx 230 25],Data.EVAname,1,...
                   'left',16,[1 1 1],'pathmaster(''Input'',''Name'',[])';...
               'MaxSlopeH1','text',[15 395 105 25],'Max Slope:',1,...
                   'left',16,'default','';...
               'MaxSlopeH','edit',[127 395 48 25],Data.MaxSlope,1,...
                   'center',16,[1 1 1],'pathmaster(''Input'',1,[])';...
               'MaxSlopeH2','push',[175 407 15 15],'+',0,...

'center',12,'default','pathmaster(''Input'',''MxSlp'',[])';...
               'MaxSlopeH3','push',[175 393 15 15],'-',0,...

'center',12,'default','pathmaster(''Input'',''MxSlm'',[])';...
               'na','text',[15 323 75 25],'Planet:',1,...
                   'left',16,'default','';...
               'na','text',[105 345 50 25],'Earth',1,...
                   'left',14,'default','';...
               'PlanetH1','radio',[165 345 25 25],'',Data.Planet==1,...
                   'left',14,'default','pathmaster(''Input'',11,[])';...
               'na','text',[105 320 50 25],'Moon',1,...
                   'left',14,'default','';...
               'PlanetH2','radio',[165 320 25 25],'',Data.Planet==2,...
                   'left',14,'default','pathmaster(''Input'',12,[])';...
               'na','text',[105 295 50 25],'Mars',1,...
                   'left',14,'default','';...
               'PlanetH3','radio',[165 295 25 25],'',Data.Planet==3,...
                   'left',14,'default','pathmaster(''Input'',13,[])';...
               'na','text',[15 245 115 25],'Mass (kg):',1,...
                   'left',16,'default','';...
               'WeightH','edit',[127 245 53 25],Data.Weight(end),1,...
                   'center',16,[1 1 1],'pathmaster(''Input'',2,[])';...
               'na','text',[210 395 53 25],'Date:',1,...
                   'left',16,'default','';...
               'MonthH','edit',[263 395 27 25],Data.Month,1,...
                   'center',16,[1 1 1],'pathmaster(''Input'',3,[])';...
               'na','text',[290 395 8 25],'/',1,...
                   'center',16,'default','';...
               'DayH','edit',[298 395 27 25],Data.Day,1,...
                   'center',16,[1 1 1],'pathmaster(''Input'',4,[])';...
               'na','text',[325 395 8 25],'/',1,...
                   'center',16,'default','';...
               'YearH','edit',[333 395 52 25],Data.Year,1,...
                   'center',16,[1 1 1],'pathmaster(''Input'',5,[])';...
               'na','text',[210 345 55 25],'Time:',1,...
                   'left',16,'default','';...
               'HourH','edit',[275 345 35
25],sprintf('%d%d',zeros(Data.Hour==0),Data.Hour),...
                   1,'center',16,[1 1 1],'pathmaster(''Input'',6,[])';...
               'na','text',[310 345 8 25],':',1,...
                   'center',16,'default','';...
               'MinuteH','edit',[320 345 35
25],sprintf('%d%d',zeros(Data.Minute<10),Data.Minute),...
                   1,'center',16,[1 1 1],'pathmaster(''Input'',7,[])';...
```

```
                'na','text',[210 295 110 25],'Time Zone:',1,...
                    'left',16,'default','';...
                'TimeZoneH','listbox',[210 239 175 56],{' Alaska Daylight',...
                    ' Alaska Standard',' Atlantic Daylight',...
                    ' Atlantic Standard',' Central Daylight',...
                    ' Central Standard',' Eastern Daylight',...
                    ' Eastern Standard',' HawaiiAleutian Daylt.',...
                    ' HawaiiAleutian Std.',' Newfoundland Std.',...
                    ' Newfoundland Daylt.',' Mountain Daylight',...
                    ' Mountain Standard',' Pacific Daylight',...
                    ' Pacific Standard'},Data.TimeZone,...
                    'left',11,[1 1 1],'pathmaster(''Input'',''Tzone'',[])';...
                'ExplorerH1','toggle',[20 194 120
25],'Astronaut',strcmp(Data.Explorers{end},...

'Astronaut'),'center',14,'default','pathmaster(''Input'',''Astronaut'',[])';.
..
                'ExplorerH2','toggle',[140 194 120 25],'On
Rover',strcmp(Data.Explorers{end},...

'Rover'),'center',14,'default','pathmaster(''Input'',''Rover'',[])';...
                'ExplorerH3','toggle',[260 194 120
25],'Robot',strcmp(Data.Explorers{end},...

'Robot'),'center',14,'default','pathmaster(''Input'',''Robot'',[])';...
                'na','text',[90 149 165 25],'Explorer Number:',1,...
                    'left',16,'default','';...
                'ExpNumH','popup',[260 175 45 1],1:Data.NumExp,Data.NumExp,...
                    'center',14,[1 1
1],'pathmaster(''Input'',''SelExp'',[])';...
                'na','push',[255 80 125 45],'Add Explorer',0,...

'center',14,'default','pathmaster(''Input'',''AddExp'',[])';...
                'StartH','push',[20 80 235 45],'START',0,...

'center',18,'default','pathmaster(''Input'',''Start'',[])';...
                'na','text',[15 40 120 18],'Render Directory:',1,...
                    'left',11,'default','';...
                'Render_dirH','edit',[15 20 318 20],Data.Render_dir,1,...
                    'left',10,[1 1 1],'pathmaster(''Input'',21,[])';...
                'na','push',[334 18 51 25],'Browse',0,...
                    'center',10,'default','pathmaster(''Input'',22,[])'};...
        for tmap = {'Obstacles' 'SoilMech' 'SciReturn' 'Other'}
            if Data.(tmap{1}(1:4)) %Choice to use existing terrain cost maps
                Gui = vertcat(Gui,{'na','text',[50 409+chkbx 300 25],...
                    ['Use existing ',tmap{1},'
map:'],1,'left',16,'default','';...
                [tmap{1}(1:4),'H'],'check',[330 407+chkbx 25
25],'',Data.Callback,'left',...

16,'default',['pathmaster(''Input'',''',tmap{1}(1:4),''',[])']});
                Data.(tmap{1}(1:4)) = Data.Callback; %Default not use
existing maps
                if Data.Callback, break, end  % On callback only give Obst
box
                chkbx = chkbx-25;
            end
```

```
    end
    if Data.hasWP && ~Data.Callback
        Gui = vertcat(Gui,{'na','text',[50 409+chkbx 300 25],...
            'Load existing Waypoints:',1,'left',16,'default','';...
        'hasWPH','check',[330 407+chkbx 25 25],'',0,...
            'left',16,'default','pathmaster(''Input'',''hasWP'',[])'});
        Data.hasWP = false;  % Default is to not use existing waypoints
    end
    for n = 1:size(Gui,1)         % Create the GUI using the info above
        Data.(Gui{n,1}) = uicontrol('Style',Gui{n,2},...
                            'Position',Gui{n,3},...
                            'String',Gui{n,4},...
                            'Value',Gui{n,5},...
                            'HorizontalAlignment',Gui{n,6},...
                            'FontSize',Gui{n,7},...
                            'BackgroundColor',Gui{n,8},...
                            'CallBack',Gui{n,9});
    end
    Data.Do = {};  % Tasks to perform when called from Mission GUI
    if Data.Callback
        set([Data.MaxSlopeH,Data.MaxSlopeH1,Data.MaxSlopeH2,...
             Data.MaxSlopeH3],'Enable','off')
        set(Data.StartH,'String','Continue')
    end
    set(PathIn,'UserData',Data,...              % Set GUI to store Data
              'HandleVisibility','callback')
                                                      % GUI controls
case {1 2 3 4 5 6 7}  % Edit data entries
    PathIn = gcf;
    Data = get(PathIn,'UserData');
    Vars = {'MaxSlope', 0, 90;...   % {Variable name, min, max}
            'Weight', 1, 999;...
            'Month', 1, 12;...
            'Day', 1, 31;...
            'Year', 2008, 2030;...
            'Hour', 0, 23;...
            'Minute', 0, 59};   % i = Explorer# for Weight, 1 otherwise
    i = (Select==2)*(get(Data.ExpNumH,'Value')-1) + 1;
    Newvalue = sscanf(get(Data.([Vars{Select,1},'H']),'String'),'%f');
    if ~isempty(Newvalue)  % If value too small or big, set to min/max
        Newvalue = min(max(Newvalue,Vars{Select,2}),Vars{Select,3});
        Data.(Vars{Select,1})(i) = round(Newvalue);
        if Select == 1
            Data.Do = {Data.Do{:},'ClrPaths','Obs','NewMaps'};
        elseif Select == 2
            Data.Do = {Data.Do{:},i};
        else
            Data.Do = {Data.Do{:},'ClrPaths','Sun'};
        end
    end
    z = zeros(((Select==6 && Data.Hour==0) ||...   % Prepends a zero to
              (Select==7 && Data.Minute<10)),1);  % Hr/Min when needed
    set(Data.([Vars{Select,1},'H']),'String',...
             sprintf('%d%d',z,Data.(Vars{Select,1})(i)))
    set(PathIn,'UserData',Data)

case 'Name'  % Edit EVA name
```

```
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        Newname = get(Data.EVAnameH,'String');
        Newname = strrep(Newname,' ','');  % Eliminate spaces
        if ~isempty(Newname) &&
all(isstrprop(Newname,'alphanum')+(Newname=='_'))
            Data.EVAname = Newname;  % Must be alphanumeric or underscore
            Data.Do = {Data.Do{:},'NewFiles'};
        end
        set(Data.EVAnameH,'String',Data.EVAname)
        set(PathIn,'UserData',Data)

    case {'Astronaut' 'Rover' 'Robot'}  % Select explorer type
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        en = get(Data.ExpNumH,'Value');
        Data.Explorers{en} = Select;
        set(Data.ExplorerH1,'Value',strcmp(Select,'Astronaut'))
        set(Data.ExplorerH2,'Value',strcmp(Select,'Rover'))
        set(Data.ExplorerH3,'Value',strcmp(Select,'Robot'))
        Data.Do = {Data.Do{:},en};
        set(PathIn,'UserData',Data)

    case {'MxSlp' 'MxSlm'}  % Slope increment & decrement buttons
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        if strcmp(Select,'MxSlp')
            Data.MaxSlope = min(Data.MaxSlope+1,90);
        elseif strcmp(Select,'MxSlm')
            Data.MaxSlope = max(Data.MaxSlope-1,0);
        end
        set(Data.MaxSlopeH,'String',Data.MaxSlope)
        Data.Do = {Data.Do{:},'ClrPaths','Obs','NewMaps'};
        set(PathIn,'UserData',Data)

    case {11 12 13}  % Planet radio buttons
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        Data.Planet = Select-10;
        set(Data.PlanetH1,'Value',Select==11)
        set(Data.PlanetH2,'Value',Select==12)
        set(Data.PlanetH3,'Value',Select==13)
        Data.Do = {Data.Do{:},'ClrPaths','Planet'};
        set(PathIn,'UserData',Data)

    case 'Tzone'  % Select time zone
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        Data.TimeZone = get(Data.TimeZoneH,'Value');
        Data.Do = {Data.Do{:},'ClrPaths'};
        set(PathIn,'UserData',Data)

    case {21 22}  % Edit Render directory
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        if Select==21                                   % Manual edit
```

```
            Newdir = get(Data.Render_dirH,'String');
        else                                                % Browse button
            Newdir = uigetdir(Data.Render_dir,['Select directory for',...
                                               'external renderer data:']);
        end
        if ~isempty(Newdir) && ischar(Newdir)
            if ~exist(Newdir,'file')
                warndlg([Newdir,' is not a valid directory.'],...
                        'Edit Render Directory')
            else
                macpc = {'/','\'};      % Append / or \
                Data.Render_dir =
regexprep(Newdir,'\w$',['$0',macpc{1+ispc}]);
                Data.Do = {Data.Do{:},'NewFiles'};
            end
        end
        try set(Data.Render_dirH,'String',Data.Render_dir)
            set(PathIn,'UserData',Data), catch  %#ok<CTCH>
        end

    case {'SelExp' 'AddExp'}  % Select explorer & Add Explorer button
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        if strcmp(Select,'AddExp')  % Add explorer
            Data.NumExp = Data.NumExp+1;
            set(Data.ExpNumH,'String',1:Data.NumExp,'Value',Data.NumExp)
            Data.Explorers{Data.NumExp} = Data.Explorers{Data.NumExp-1};
            Data.Weight(Data.NumExp) = Data.Weight(Data.NumExp-1);
            Data.Do = {Data.Do{:},'AddExp'};
        end
        en = get(Data.ExpNumH,'Value');
        set(Data.ExplorerH1,'Value',strcmp(Data.Explorers{en},'Astronaut'))
        set(Data.ExplorerH2,'Value',strcmp(Data.Explorers{en},'Rover'))
        set(Data.ExplorerH3,'Value',strcmp(Data.Explorers{en},'Robot'))
        set(Data.WeightH,'String',Data.Weight(en))
        set(PathIn,'UserData',Data)

    case {'Obst' 'Soil' 'SciR' 'Othe' 'hasWP'}  % Use existing maps toggles
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        Data.(Select) = get(Data.([Select,'H']),'Value');
        if strcmp(Select,'Obst')
            state = {'on','off'};
            set([Data.MaxSlopeH,Data.MaxSlopeH1,Data.MaxSlopeH2,...
                 Data.MaxSlopeH3],'Enable',state{1+Data.Obst})
        end
        set(PathIn,'UserData',Data)

    case 'Start'  % START button
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        if exist([Data.Render_dir,Data.EVAname,'_Elevations.txt'],'file')
&&...
           ~Data.Callback
            Choice = questdlg(sprintf('%s already exists.\n\nOK to
overwrite?',...
```

```
                    Data.EVAname),'Overwrite Mission...','Yes','No','No');
            if strcmp(Choice,'No'), return, end
        end
        delete(PathIn)
        Data.UTMzone = Data.UTMzone*(Data.Planet==1); % UTMzone 0 off earth
        if ~Data.Callback
            message = ['Calculating obstacles,\n',...
                       ' writing map files,\n',...
                       ' preparing surface...'];
            Data.calcmsg = helpdlg(sprintf(message),'Pathmaster');
            pathmaster('CostMaps',[],Data)             % Call to 'CostMaps'
        else
            pathmaster('Update',Data.Do,Data)            % Call to 'Update'
        end

    case 'Close'  % Close GUI, exit pathmaster
        PathIn = gcf;
        Data = get(PathIn,'UserData');
        if ~Data.Callback
            Choice = questdlg(['Close ',Data.EVAname,' without saving?'],...
                              'Exit Pathmaster...','Yes','No','No');
            if strcmp(Choice,'Yes')
                delete(PathIn)
            end
        else
            delete(PathIn)
            pathmaster('Update',{},Data)                 % Call to 'Update'
        end
end

%% ************ SET SLOPES, OBSTACLES, SOIL MECH, SCI RETURN, OTHER *******
case 'CostMaps'
[gx,gy] = gradient(Data.Elevations,Data.Resolution);
Data.Slopes = atan(sqrt(gx.^2+gy.^2))*(180/pi);  % Slopes in degrees
Data.Slop = true;
if ~Data.Obst        % Find obstacles (not using an existing obstacles map)
    Data.Obstacles = Data.Slopes > Data.MaxSlope; % 1 if obstacle, else 0
end
if all(all(Data.Obstacles))  % Check if entire map is obstacle
    try delete(Data.calcmsg), catch end  %#ok<CTCH>
    message = ['The map has no travereable terrain\n',...
               'and is entirely obstacles.\n\n',...
               'Increase the maximum slope.'];
    waitfor(warndlg(sprintf(message),'Pathmaster'))
    pathmaster('Input',0,Data)  % Restart back at 'Input'
    return
end
Data.Obst = true;
for tmap = {'SoilMech' 'SciReturn' 'Other'}
    if ~Data.(tmap{1}(1:4))        % If no map loaded, default map all zero
        Data.(tmap{1}) = zeros(Data.Rows,Data.Cols);
    end
end
pathmaster('SaveMaps','',Data)                           % Call to 'SaveMaps'

%% ************ SAVE MAP FILES ***************************************
```

```
case 'SaveMaps'
form = [ones(Data.Cols,1)*'%.4f ';'   \n'].';             % Write map files
for tmap = {'Elevations' 'Obstacles' 'Slopes' 'SoilMech' 'SciReturn' 'Other'}
    if (Data.(tmap{1}(1:4)) && ~Data.Path) || (Data.Path &&
Data.([tmap{1}(1:4),'Ed']))
        try save([Data.EVAname,'_Data'],'-struct','Data',tmap{1},Select),
catch end %#ok<CTCH>
        cd(Data.Render_dir)
        for Outfile = {Data.EVAname,'Current'}
            mpf = fopen([Outfile{1},'_',tmap{1},'.txt'],'wt');
            fprintf(mpf,['ncols         %d\n',...
                         'nrows         %d\n',...
                         'xllcorner     %.8f\n',...
                         'yllcorner     %.8f\n'],...
                        Data.Cols,Data.Rows,Data.xllcorner,Data.yllcorner);
            if Data.UTMzone ~= 0
                fprintf(mpf,'UTMzone       %d\n',Data.UTMzone);
            end
            fprintf(mpf,['cellsize      %.3f\n',...
                         'NODATA_value  %d\n'],Data.Resolution,Data.NoData);
            fprintf(mpf,form,max(Data.(tmap{1}).',Data.NoData)); %Matrix,
NoData=-9999
            fclose(mpf);
        end
        cd(Data.Work_dir)
    end
    form = [ones(Data.Cols,1)*'%d ';' \n'].';
    Select = '-append';
end
if ~Data.Path
    try save([Data.EVAname,'_Data'],'-
struct','Data','Resolution','xllcorner',...
                       'yllcorner','UTMzone','NoData','Explorers','-append'),
catch %#ok<CTCH>
    end
end
if ~Data.Callback
    pathmaster('Mission',0,Data)                          % Call to 'Mission'
end

%% ************ MISSION PLANNER GUI ************************************
case 'Mission'
switch Select
    case 0  % Define waypoint, traverse path, and cost cell arrays
        [Data.Waypoints{1:Data.NumExp,1:2}] = deal([]);
        [Data.WayHandles{1:Data.NumExp,1:2}] = deal([]);
        [Data.Pathpoints{1:Data.NumExp,1:2}] = deal([]);
        [Data.PathHandles{1:Data.NumExp,1:2}] = deal([]);
        [Data.Distance{1:Data.NumExp,1:2}] = deal([]);
        [Data.MetCost{1:Data.NumExp,1:2}] = deal([]);
        [Data.Time{1:Data.NumExp,1:2}] = deal([]);
        [Data.ElevEd,Data.SlopEd,Data.ObstEd,...  % Info edited booleans
         Data.SoilEd,Data.SciREd,Data.OtheEd,Data.WayPEd] = deal(false);
        Data.R = '';           % Set to 'R' for "return home" path
        Data.TEsize = 1;       % Terrain edit size default
        Data.datadisp = 1;     % Determines which waypoint data to display
        Data.prevwp = [-1 -1]; % The previously selected waypoint coords
```

```
        Data.Callback = true;  % Calls to previous sections now "Callback"
                                        % Develop terrain surface for GUI
        Data.Elmin = min(min(Data.Elevations));
        Data.Eldiff = max(max(Data.Elevations))-Data.Elmin + 10^-3;
        Data.ColorLim = [Data.Elmin, Data.Elmin+Data.Eldiff*64/63;...
                Data.Elmin, Data.Elmin+Data.Eldiff*64/63; 0 2; 0 2; 0 2];
        Data.ColorObsRed = min(Data.Elevations+Data.Obstacles*10^6,...
                                Data.Elmin+Data.Eldiff*64/63);
        try delete(Data.calcmsg), catch end  %#ok<CTCH>
                                         % Initialize Mission Planner GUI
        Data.MPfig = figure('Name',['Pathmaster:  ',Data.EVAname,' - Mission
Planner'],...
            'Position',[round(Data.Scrsize(3)/32)
round(Data.Scrsize(4)/32),...
                        round(Data.Scrsize(3)*15/16)
round(Data.Scrsize(4)*7/8)],...
            'Color',[.92549 .913725 .847059],...
            'IntegerHandle','off',...
            'DockControls','off',...
            'NumberTitle','off',...
            'Renderer','OpenGL',...
            'ResizeFcn','pathmaster(''Mission'',''Resize'',[])',...
            'CloseRequestFcn','pathmaster(''Mission'',''Close'',[])');
        if isempty(strfind(system_dependent('getos'),'Vista'))
            set(Data.MPfig,'Pointer','fullcrosshair')
        end

        Data.MPaxes = axes('Units','pixels');          % Axes for surf plot
        try                                 % Surf plot with obstacles in red
            Data.MPsurf = surf(Data.MPaxes,0:Data.Cols-1,0:Data.Rows-1,...
                Data.Elevations,'CData',Data.ColorObsRed,...
                'FaceColor','interp',...
                'EdgeColor','none',...
                'ButtonDownFcn','pathmaster(''Mission'',''Click'',[])');
        catch  %#ok<CTCH>
            delete(Data.MPfig)
            message = ['\n-------------------------------------------',...
                '\nMatlab has encountered an error while trying\n',...
                'to create a surface rendering.\n\n',...
                'This is a bug caused by use of the HELP command.\n\n',...
                'Please exit and restart Matlab.\n',...
                '-------------------------------------------\n'];
            error('Help:Figure_or_Axes',message)
        end                                 % Axes scaling (initially meters)
        [xsize,ysize] =
deal(Data.Resolution*Data.Cols,Data.Resolution*Data.Rows);
        mapsize = max(xsize,ysize);
        mag = floor(log10(mapsize));
        scale = round(mapsize/10^mag)*10^(mag-1);
        zaspect = Data.Resolution*min(1,10*Data.Eldiff/mapsize);
        zmag = floor(log10(Data.Eldiff));
        zscale = round(Data.Eldiff/10^zmag)*10^(zmag-1)*2;
        set(Data.MPaxes,'YDir','reverse',...     % Set plot axes properties
                'View',[0 90],...
                'DataAspectRatio',[1 1 zaspect],...
                'CLim',Data.ColorLim(1,:),...
                'XLim',[0 Data.Cols-1],...
```

```
                'YLim',[0 Data.Rows-1],...
                'ZLim',[Data.Elmin-.01*Data.Eldiff,
Data.Elmin+1.25*Data.Eldiff],...
                'XTick',(scale:scale:xsize)/Data.Resolution,...
                'YTick',(mod(ysize-
.001,scale)+.001:scale:ysize)/Data.Resolution,...
                'ZTick',Data.Elmin:zscale:Data.Elmin+1.26*Data.Eldiff,...
                'XTickLabel',scale:scale:xsize,...
                'YTickLabel',ysize-(mod(ysize-.001,scale)+.001):-scale:0,...
                'ZTickLabel',0:zscale:1.25*Data.Eldiff,...
                'TickLength',[0 0],...
                'Color',[.92549 .913725 .847059])
        if isempty(Data.Lite)   % This code does not execute in 'lite' mode

set(Data.MPsurf,'FaceLighting','gouraud','BackFaceLighting','lit')
            material([.4 .8 0])        % Set surface reflectance properties
            Data.Sun =
light('Position',[sin((Data.Hour+Data.Minute/60)*pi/12),...
                -cos((Data.Hour+Data.Minute/60)*pi/12),...
                .014+.006*sin((Data.Hour+Data.Minute/60)*pi/24)],...
                'Style','infinite');  % Illumination, varies by time
        end
        if Data.UTMzone~=0  % Display compass when lat/long is active
           y = Data.Rows/20;  x = Data.Cols-2-y/10;
           z = max(max(Data.Elevations(1:ceil(y*5/4),floor(end-
y/2):end)))+.05*Data.Eldiff;
           line([x-y*9/32 x-y*9/32 x x],...
                [1+y*17/16 1+y*5/8 1+y*17/16 1+y*5/8],[z z z z],...
                'Color','k','LineWidth',2,'HitTest','off')
           line([x-y*9/32 x-y*9/64 x-y*9/64 x-y*9/64 x],...
                [1+y/4 1+y/16 1+y*9/16 1+y/16 1+y/4],[z z z z z],...
                'Color','k','LineWidth',2,'HitTest','off')
        end
                                                  % Render mode colormaps
        Mcolors = [[0 .3 .15; .9 .7 .4],[.2 .2 .25; .99 .99 1],[.25 .15 .1; 1
.8 .5]];
        colors1 = zeros(63,9);                    % Terrain cost colormaps
        Tcolors = [[.85 .85 .85; .5 .05 .1],[.85 .85 .85; .25 .1 .6],[.85 .85
.85; 0 0 .8]];
        colors2 = zeros(64,9);
        for i = 1:9
            colors1(:,i) = linspace(Mcolors(1,i),Mcolors(2,i),63);
            colors2(:,i) = linspace(Tcolors(1,i),Tcolors(2,i),64);
        end
        Data.Colors = [[colors1; .95 .02 .15 .95 .02 .15 .95 .02 .15],
colors2];
        colormap(Data.MPaxes,Data.Colors(:,3*Data.Planet-2:3*Data.Planet))
        Data.Ecolors = {[0 0 1],[.9 .9 0],[.5 0 .5],[1 .5 0],[0 1 1],... %
Explorer
                        [1 0 1],[.4 .2 0],[.8 .35 .35],[1 1 1],[0 0 0]}; %
colors

        Data.MPmenu = uipanel('Units','pixels');    % Panel for UI controls
      % UIcontrols: {Handle,Style,Position,String,Value,HorizontalAlignment,
      %                  FontSize,BackgroundColor,Enable,Callback}
        MPui = {'MPhelpH','push',[5 34 60 25],'HELP',0,'center',12,...
                   'default','on','pathmaster(''Mission'',''Help'',[])';...
```

```
                'MPmapiH','push',[65 34 85 25],'Map Info',0,'center',12,...
                    'default','on','pathmaster(''Mission'',''Map'',[])';...
                'MPevaiH','push',[150 34 85 25],'EVA Input',0,'center',12,...
                    'default','on','pathmaster(''Mission'',''Input'',[])';...
                'na','text',[240 34 50 22],'Axes: ',1,'right',12,...
                    'default','on','';...
                'MPscaleH','popup',[290 60 75
1],{'Meters','Km','Feet','Miles'},...
                    1,'left',12,[1 1
1],'on','pathmaster(''Mission'',''Scale'',[])';...
                'na','frame',[370 31 287 32],'',1,'left',12,...
                    'default','on','';...
                'MPwpmodeH','toggle',[374 34 90 25],'Waypoints',1,'center',...
                    12,'default','on','pathmaster(''Mission'',''WP'',[])';...
                'MPexptxtH','text',[468 34 75 22],'Explorer: ',1,'right',...
                    12,'default','on','';...
                'MPexpnumH','popup',[543 60 40 1],1:Data.NumExp,1,...
                    'center',12,[1 1
1],'on','pathmaster(''Mission'',''SelExp'',[])';...
                'MPwaytitleH','text',[588 34 64 23],'Start:',1,...
                    'center',14,'default','on','';...
                'na','frame',[662 31 358 32],'',1,'left',12,...
                    'default','on','';...
                'MPtermodeH','toggle',[666 34 65 25],'Terrain',0,'center',...
                    12,'default','on','pathmaster(''Mission'',''TER'',[])';...
                'MPtertypeH','popup',[736 60 100 1],{'Obstacles','Soil
mech','Sci return','Other'},...
                    1,'left',12,[1 1
1],'on','pathmaster(''Mission'',''TerON'',[])';...
                'MPtersizetxtH','text',[841 34 45 22],'Size: ',1,'right',...
                    12,'default','off','';...
                'MPtersizeH','edit',[886 34 30 25],1,1,'center',12,...
                    [1 1 1],'off','';...
                'MPtersizepH','push',[916 47 15 15],'+',0,'right',12,...
                    'default','off','pathmaster(''Mission'',''TSp'',[])';...
                'MPtersizemH','push',[916 32 15 15],'-',0,'right',12,...
                    'default','off','pathmaster(''Mission'',''TSm'',[])';...
                'MPterOnH','toggle',[936 34 40 25],'ON',1,'center',12,...
                    'default','on','pathmaster(''Mission'',''TerON'',[])';...
                'MPterOffH','toggle',[976 34 40 25],'OFF',0,'center',12,...
                    'default','on','pathmaster(''Mission'',''TerOFF'',[])';...
                'na','push',[5 1 133 30],'Run PATH',0,'center',14,...
                    'default','on','pathmaster(''Mission'',''PATH'',[])';...
                'na','text',[148 1 95 23],'Path         :',1,'left',14,...
                    'default','on','';...
                'MPpathenH','edit',[195 1 40 25],'',1,'center',14,...
                    [1 1 1],'inactive','';...
                'na','text',[250 1 50 23],'Dist: ',1,'right',14,...
                    'default','on','';...
                'MPdistNH','edit',[300 1 82 25],'',1,'right',14,...
                     [1 1 1],'inactive','';...
                'MPdistUH','edit',[385 1 60 25],'Meters',1,'center',12,...
                     [1 1 1],'inactive','';...
                'na','text',[455 1 55 23],'Cost: ',1,'right',14,...
                    'default','on','';...
                'MPcostNH','edit',[510 1 87 25],'',1,'right',14,...
                    [1 1 1],'inactive','';...
```

```
                'MPcostUH','popup',[600 28 60 1],{'Kcal','BTU','KJ'},1,...
                    'left',12,[1 1
1],'on','pathmaster(''Mission'',''CostU'',[])';...
                'na','text',[670 1 55 23],'Time: ',1,'right',14,...
                    'default','on','';...
                'MPtimeH','edit',[725 1 70 25],'',1,'center',14,...
                    [1 1 1],'inactive','';...
                'na','text',[805 1 65 22],'Render: ',1,'right',12,...
                    'default','on','';...
                'MPrendmH1','toggle',[870 1 50 25],'Earth',Data.Planet==1,...

'center',12,'default','on','pathmaster(''Mission'',1,[])';...
                'MPrendmH2','toggle',[920 1 50 25],'Moon',Data.Planet==2,...

'center',12,'default','on','pathmaster(''Mission'',2,[])';...
                'MPrendmH3','toggle',[970 1 50 25],'Mars',Data.Planet==3,...

'center',12,'default','on','pathmaster(''Mission'',3,[])'};
        for n = 1:size(MPui,1)          % Create the GUI using the info above
            Data.(MPui{n,1}) = uicontrol(Data.MPmenu,'Style',MPui{n,2},...
                                'Position',MPui{n,3},...
                                'String',MPui{n,4},...
                                'Value',MPui{n,5},...
                                'HorizontalAlignment',MPui{n,6},...
                                'FontSize',MPui{n,7},...
                                'FontWeight','bold',...
                                'BackgroundColor',MPui{n,8},...
                                'Enable',MPui{n,9},...
                                'CallBack',MPui{n,10});
        end
        set(Data.MPfig,'UserData',Data,...           % Set GUI to store Data
                  'HandleVisibility','callback')

        if Data.hasWP                                % Load existing waypoints
            Data.WayPEd = true;
            hold on
            for en = 1:min(Data.NumExp,size(Data.LoadWaypoints,1))
                if ~isempty(Data.LoadWaypoints{en})
                Data.Waypoints{en} =
[min(max(Data.LoadWaypoints{en}(:,1),0),Data.Cols-1),...

min(max(Data.LoadWaypoints{en}(:,2),0),Data.Rows-1)];
                for i = 1:size(Data.Waypoints{en},1)
                    Data.Waypoints{en}(i,3) =
Data.Elevations(Data.Waypoints{en}(i,2)+1,...

Data.Waypoints{en}(i,1)+1);
                end
                Data.WayHandles{en}(1) =
scatter3(Data.MPaxes,Data.Waypoints{en}(:,1),...

Data.Waypoints{en}(:,2),Data.Waypoints{en}(:,3)+.05*Data.Eldiff,...
                    120,Data.Ecolors{mod(en-1,10)+1},'filled',...

'ButtonDownFcn',sprintf('pathmaster(''Mission'',''Click'',%d)',en));
                txt = {'H',{'H';num2str((1:i-1).')}};
                WPtxt = text(Data.Waypoints{en}(:,1),...
```

```
                    Data.Waypoints{en}(:,2)-.007*Data.Rows,...
                    Data.Waypoints{en}(:,3)+.07*Data.Eldiff,txt{1+(i>1)},...
                    'HorizontalAlignment','center',...
                    'VerticalAlignment','bottom',...
                    'Color',[1 1 1],'FontWeight','bold','HitTest','off');
                Data.WayHandles{en} = [Data.WayHandles{en}(1);WPtxt];
                end
            end
            hold off
            set(Data.MPfig,'UserData',Data)    % Store the Waypoint data
            set(Data.MPexpnumH,'Value',en)
            pathmaster('Mission','SelExp',[]) % Select last explorer loaded
        end
                                                            % GUI Controls
    case 'Help'
        macpc = {'CTRL+','RIGHT-'};
        helpmsg = ['MISSION PLANNER GUI\n','_____________________\n\n',...
                   'LEFT-CLICK:     Add waypoints or terrain
characteristics\n\n',...
                   'SHIFT+CLICK:  Clear waypoints or terrain
characteristics\n\n',...
                   'SHIFT+CLICK ON PATH: Find return home path\n\n',...
                   'DOUBLE-CLICK:  Heighten terrain characteristics\n\n',...
                   macpc{1+ispc},'CLICK:  Display terrain or waypoint data,
select paths\n',...
                   '_____________________________________________\n\n',...
                   'Map Info & EVA Input buttons:  Reopen menus to change
mission data\n\n',...
                   'Scale menu:  Update axes scaling with selected
units\n\n',...
                   'Waypoints & Terrain buttons:  Select edit mode\n\n',...
                   'Explorer menu:  Select the current explorer\n\n',...
                   'Terrain menu:   Select the terrain characteristic to
display\n\n',...
                   'Size Control:   Adjust the size of the terrain edit
rectangle\n\n',...
                   'Terrain ON & OFF:  Turn terrain characteristic display on
& off\n\n',...
                   'Run PATH button:  Run the PATH-based optimization to find
traverse paths\n\n',...
                   'Render buttons:  Select the terrain render mode'];
        questdlg(sprintf(helpmsg),'Pathmaster Help','OK','OK');

    case {'Map' 'Input'}
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        set(MPfig,'Visible','off')
        try delete(Data.minfo), catch end  %#ok<CTCH> % Clear map data text
if exists
        pathmaster(Select,0,Data)  % Call to 'Map' OR 'Input'

    case 'Scale'  % Change axes scale
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        distU = get(Data.MPscaleH,'Value');
        distR = {1,'%.0f','Meters'; .001,'%.2f','Km';...
```

```
                3.28084,'%.0f','Feet'; .0006213712,'%.2f','Miles'};
        [xsize,ysize] =
deal(Data.Resolution*Data.Cols,Data.Resolution*Data.Rows);
        mapsize = max(xsize,ysize);
        mag = floor(log10(mapsize*distR{distU,1}));
        scale = round(mapsize*distR{distU,1}/10^mag)*10^(mag-1);
        tscale = scale/distR{distU,1}/Data.Resolution;
        zmag = floor(log10(Data.Eldiff*distR{distU,1}));
        zscale = round(Data.Eldiff*distR{distU,1}/10^zmag)*10^(zmag-1)*2;
        set(Data.MPaxes,'XTick',tscale:tscale:Data.Cols,...
                        'YTick',mod(Data.Rows-1,tscale)+1:tscale:Data.Rows,...

'ZTick',Data.Elmin:zscale/distR{distU,1}:Data.Elmin+1.25*Data.Eldiff,...
                        'XTickLabel',scale:scale:xsize*distR{distU,1},...
                        'YTickLabel',ysize*distR{distU,1}-(mod((ysize-
.001)*distR{distU,1},...
                                             scale)+.001*distR{distU,1}):-
scale:0,...
                        'ZTickLabel',0:zscale:1.25*Data.Eldiff*distR{distU,1})
        set(Data.MPdistUH,'String',distR{distU,3})
        if ~isempty(get(Data.MPdistNH,'String'))
            path = get(Data.MPpathenH,'String');
            en = get(Data.MPexpnumH,'Value') + Data.NumExp*(path(end)=='R');
            set(Data.MPdistNH,'String',sprintf(distR{distU,2},...

Data.Distance{en}(end)*distR{distU,1}))
        end

    case {'WP' 'TER'}  % Select edit mode
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        set(Data.MPwpmodeH,'Value',strcmp(Select,'WP'))
        set(Data.MPtermodeH,'Value',strcmp(Select,'TER'))
        state = {'off','on','inactive'};
        set([Data.MPexptxtH,Data.MPexpnumH,Data.MPwaytitleH,Data.MPterOnH,...
             Data.MPterOffH],'Enable',state{1+strcmp(Select,'WP')})
        set([Data.MPtersizetxtH,Data.MPtersizepH,Data.MPtersizemH],...
            'Enable',state{1+strcmp(Select,'TER')})
        set(Data.MPtersizeH,'Enable',state{1+2*strcmp(Select,'TER')})
        if strcmp(Select,'TER')
            pathmaster('Mission','TerON',[])
        end

    case 'SelExp'  % Select explorer for waypoint edits & data display
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        enR = get(Data.MPexpnumH,'Value');
        ecolors = {Data.Ecolors{1:8},'k','k'};
        set(Data.MPexpnumH,'ForegroundColor',ecolors{mod(enR-1,10)+1})
        txt = {'Start:',sprintf('WP %d:',size(Data.Waypoints{enR},1))};
        title = txt{1 + ~isempty(Data.Waypoints{enR})};
        set(Data.MPwaytitleH,'String',title)
        if ~isempty(Data.Pathpoints{enR})  % Set the cost displays
            en = enR + ~isempty(Data.R)*Data.NumExp;
            set(Data.MPpathenH,'String',sprintf('%d%s',enR,Data.R),...
                               'ForegroundColor',ecolors{mod(enR-1,10)+1})
```

```
            distU = get(Data.MPscaleH,'Value');
            distR = {1,'%.0f'; .001,'%.2f'; 3.28084,'%.0f';
.0006213712,'%.2f'};
            set(Data.MPdistNH,'String',sprintf(distR{distU,2},...

Data.Distance{en}(end)*distR{distU,1}))
            costR = [.2521644 1 1.055056];
            set(Data.MPcostNH,'String',sprintf('%.1f',...
                Data.MetCost{en}(end)*costR(get(Data.MPcostUH,'Value'))))
            hourmin = [floor(Data.Time{en}(end)/3600),...
                       round(rem(Data.Time{en}(end),3600)/60)];
            set(Data.MPtimeH,'String',sprintf('%d:%d%d',...

hourmin(1),zeros(hourmin(2)<10),hourmin(2)))
        else

set([Data.MPpathenH,Data.MPdistNH,Data.MPcostNH,Data.MPtimeH],'String','')
        end

    case {'TerON' 'TerOFF'}  % Select terrain map to display
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        set(Data.MPterOnH,'Value',strcmp(Select,'TerON'))
        set(Data.MPterOffH,'Value',strcmp(Select,'TerOFF'))
        state = {'off' 'on'};
        set(Data.MPtertypeH,'Enable',state{1+strcmp(Select,'TerON')})
        ter = get(Data.MPtertypeH,'Value');
        tcm = {'Elevations','ColorObsRed','SoilMech','SciReturn','Other'};
        set(Data.MPaxes,'CLim',Data.ColorLim(1+ter*strcmp(Select,'TerON'),:))
        set(Data.MPsurf,'CData',Data.(tcm{1+ter*strcmp(Select,'TerON')}))
        if strcmp(Select,'TerOFF') || ter==1
            rendm = find([get(Data.MPrendmH1,'Value'),...
                get(Data.MPrendmH2,'Value'),get(Data.MPrendmH3,'Value')]);
        else
            rendm = ter+2;
        end
        colormap(Data.MPaxes,Data.Colors(:,3*rendm-2:3*rendm))

    case {'TSp' 'TSm'}  % Change terrain map edit rectangle size
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        sizes = [.1 .2 .3 .4 .5 .6 .7 .8 .9 1 1.5 2 2.5 3 4 5 6 7 8 9 10];
        ces = find(sizes==Data.TEsize);
        if strcmp(Select,'TSp') && ces < 21
            Data.TEsize = sizes(ces+1);
        elseif strcmp(Select,'TSm') && ces > 1
            Data.TEsize = sizes(ces-1);
        end
        set(Data.MPtersizeH,'String',Data.TEsize)
        set(MPfig,'UserData',Data)

    case 'CostU'
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        if ~isempty(get(Data.MPcostNH,'String'))
            costR = [.2521644 1 1.055056];  % Ratios: Kcal, BTU, KJ
```

```
            set(Data.MPcostNH,'String',sprintf('%.1f',...
                Data.MetCost{get(Data.MPexpnumH,'Value')}(end) * ...
                costR(get(Data.MPcostUH,'Value'))))
        end

    case {1 2 3}  % Change render mode
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        set(Data.MPrendmH1,'Value',Select==1)
        set(Data.MPrendmH2,'Value',Select==2)
        set(Data.MPrendmH3,'Value',Select==3)
        if get(Data.MPterOffH,'Value') || get(Data.MPtertypeH,'Value')==1
            colormap(Data.MPaxes,Data.Colors(:,3*Select-2:3*Select))
        end

    case 'PATH'  % Run PATH button or return home path
        paths2do = Data;  % For return home path, this is the explorer #
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        set([Data.MPhelpH,Data.MPmapiH,Data.MPevaiH],'enable','off')
        try delete(Data.minfo), catch end  %#ok<CTCH>
        if isempty(paths2do)  % Nominal case (non return home)
            Data.MPfigH = uisuspend(MPfig);
            [numWP,hasPath] = deal(zeros(Data.NumExp,1));
            for i = 1:Data.NumExp        % Find which explorers need paths
                numWP(i) = size(Data.Waypoints{i},1);
                hasPath(i) = ~isempty(Data.Pathpoints{i});
            end
            paths2do = find(numWP>=2 & ~hasPath).'; % Array of explorer #'s
        end
        if ~isempty(paths2do)    % Save all edits & calculate the new paths
            Data.Path = true;
            try save([Data.EVAname,'_Data'],'-struct','Data','Waypoints','-
append'), catch end %#ok<CTCH>
            pathmaster('SaveMaps','-append',Data)      % Call to 'SaveMaps'
            [Data.ObstEd,Data.SoilEd,Data.SciREd,...  % Reset edit booleans
             Data.OtheEd,Data.WayPEd] = deal(false);
            pathmaster('PATH',paths2do,Data)               % Call to 'PATH'
            Data = get(MPfig,'UserData');  % Path data saved
            if ~isempty(Data.Newpaths)
                hold on
                for en = Data.Newpaths(end:-1:1)       % Plot the new paths
                    if isempty(Data.R)  % Make waypoints big & green
                        set(Data.WayHandles{en}(1),'SizeData',200,'CData',[0
1 0])
                    end
                    C = Data.Ecolors{mod(mod(en-1,Data.NumExp),10)+1};
                    axes(Data.MPaxes)  % Make this GUI's axes current
                    Data.PathHandles{en}(1) =
line(Data.Pathpoints{en}(:,1),...
                        Data.Pathpoints{en}(:,2),...
                        Data.Pathpoints{en}(:,3)+.05*Data.Eldiff,...
                        'Color',C,'LineWidth',4,'Marker','o',...

'MarkerEdgeColor',C,'MarkerFaceColor',C,'MarkerSize',5,...
```

```
'ButtonDownFcn',sprintf('pathmaster(''Mission'',''Click'',%d)',en));
                end
                hold off
                if ~isempty(Data.R)  % Set to dotted line for return home
path
                    set(Data.PathHandles{en}(1),'LineStyle','--')
                end
                set(Data.MPexpnumH,'Value',mod(Data.Newpaths(1)-
1,Data.NumExp)+1)
            end
            Data.Path = false;
            set(MPfig,'UserData',Data)          % Store data
            pathmaster('Mission','SelExp',[])  % Set cost displays
        else
            message = 'There are no new traverse paths to find.';
            if any(numWP==1)
                message = [message,'\n\nA path requires a start and at least
1 waypoint.'];
            end
            Data.minfo = helpdlg(sprintf(message),'Pathmaster');
            set(MPfig,'UserData',Data)
        end
        set([Data.MPhelpH,Data.MPmapiH,Data.MPevaiH],'enable','on')
        uirestore(Data.MPfigH)
        if ~isempty(Data.errpath)  % If error on any path
            message = ['An error occured on path%s: ',...
                       num2str(Data.errpath,' %d,')];
            message = [message(1:end-1),'%s\n\nMake sure waypoints are
not\n',...
                       'enclosed by obstacles.'];
            s = {'','s'};

errordlg(sprintf(message,s{1+(length(Data.errpath)>1)},Data.R),'Path Error');
        end

    case 'Click'  % Mouse click: Waypoints, terrain edits, paths, data
display
        ClOnPath = Data;  % If a path clicked on, this is the explorer #
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        Task = get(MPfig,'SelectionType');       % Click type
        clpt = get(Data.MPaxes,'CurrentPoint');  % Click location
        clx = max(min(round((clpt(1,1)+clpt(2,1))/2),Data.Cols-1),0);
        cly = max(min(round((clpt(1,2)+clpt(2,2))/2),Data.Rows-1),0);
        try delete(Data.minfo), catch end  %#ok<CTCH> % Clear map text
        switch Task
            case {'normal' 'extend' 'open'} % Left-Click,Shift+Click,Double
                if strcmp(Task,'extend') && ~isempty(ClOnPath) &&
numel(ClOnPath)==1 && ...
                        ClOnPath<=Data.NumExp &&
~isempty(Data.Pathpoints{ClOnPath}) && ...
                        ~Data.Obstacles(cly+1,clx+1) % Shift+Click path:
return home
                    en = ClOnPath+Data.NumExp;
                    hold on
                    Data.WayHandles{en} = scatter3(Data.MPaxes,clx,cly,...
```

```
                        Data.Elevations(cly+1,clx+1)+.05*Data.Eldiff,300,[0 0
0],'filled');
                    hold off
                    Data.MPfigH = uisuspend(MPfig);
                    Choice = questdlg(sprintf('Calculate return home path for
Explorer %d?',...
                                              ClOnPath),'Return
Home','Yes','No','Yes');
                    if strcmp(Choice,'No')
                        delete(Data.WayHandles{en})
                        Data.WayHandles{en} = [];
                        uirestore(Data.MPfigH)
                    else
                        delete(Data.PathHandles{en}) %Clear prev return path
                        Data.Waypoints{en} = [clx cly
Data.Elevations(cly+1,clx+1);
                                                Data.Waypoints{ClOnPath}(1,:)];
                        Data.PathHandles{en}(2) = Data.WayHandles{en};
                        Data.WayHandles{en} = [];
                        Data.R = 'R';  % Indicates "return home" path
                        set(MPfig,'UserData',Data)
                        pathmaster('Mission','PATH',en)  % Call to 'PATH'
option
                        Data = get(MPfig,'UserData');
                        Data.R = '';
                    end
                                                        % Waypoint edit mode
                elseif get(Data.MPwpmodeH,'Value') && ~strcmp(Task,'open')
                    en = get(Data.MPexpnumH,'Value');  % Explorer #
                    if ~isempty(Data.Pathpoints{en})   % Check if path exists
                        MPfigH = uisuspend(MPfig);
                        Choice = questdlg('Editing waypoints will clear the
traverse path.',...
                                          sprintf('Edit Explorer
%d',en),'OK','Cancel','OK');
                        uirestore(MPfigH)
                        if strcmp(Choice,'Cancel'), return, end
                        delete(Data.PathHandles{[en,en+Data.NumExp]})
                        set(Data.WayHandles{en}(1),'SizeData',120,...
                            'CData',Data.Ecolors{mod(en-1,10)+1})
                        Data.Waypoints{en+Data.NumExp} = [];
                        for vars = {'Pathpoints' 'PathHandles' 'Distance'
'MetCost' 'Time'}
                            [Data.(vars{1}){[en,en+Data.NumExp]}] = deal([]);
                        end
                        set([Data.MPpathenH,Data.MPdistNH,Data.MPcostNH,...
                             Data.MPtimeH],'String','')
                        set(MPfig,'UserData',Data)
                    end
                    Data.WayPEd = true;
                    if strcmp(Task,'normal')  % Left-Click: add waypoint
                        if Data.Obstacles(cly+1,clx+1) ||...   %Click on obs
                           (~isempty(Data.Waypoints{en}) &&... %or prev waypt
                            all(Data.Waypoints{en}(end,1:2)==[clx cly]))
                            return
                        end                     % Append new waypoint
                        Data.Waypoints{en} = [Data.Waypoints{en}; clx cly,...
```

```
                                                Data.Elevations(cly+1,clx+1)];
                        else                    % Shift+Click: erase waypoint
                            if isempty(Data.Waypoints{en}), return, end
                            Data.Waypoints{en} = Data.Waypoints{en}(1:end-1,:);
                        end
                        delete(Data.WayHandles{en})      % Clear prev waypoints
                        if ~isempty(Data.Waypoints{en})  % Plot waypoints in GUI
                            hold on
                            Data.WayHandles{en}(1) = scatter3(Data.MPaxes,...

Data.Waypoints{en}(:,1),Data.Waypoints{en}(:,2),...
                                Data.Waypoints{en}(:,3)+.05*Data.Eldiff,...
                                120,Data.Ecolors{mod(en-1,10)+1},'filled',...

'ButtonDownFcn',sprintf('pathmaster(''Mission'',''Click'',%d)',en));
                            txt =
{'H',{'H';num2str((1:size(Data.Waypoints{en},1)-1).')}};
                            WPtxt = text(Data.Waypoints{en}(:,1),...
                                Data.Waypoints{en}(:,2)-.007*Data.Rows,...
                                Data.Waypoints{en}(:,3)+.07*Data.Eldiff,...
                                txt{1+(size(Data.Waypoints{en},1)>1)},...
                                'HorizontalAlignment','center',...
                                'VerticalAlignment','bottom',...
                                'Color',[1 1
1],'FontWeight','bold','HitTest','off');
                            Data.WayHandles{en} = [Data.WayHandles{en}(1);WPtxt];
                            hold off
                            set(Data.MPwaytitleH,'String',sprintf('WP %d:',...
                                                 size(Data.Waypoints{en},1)))
                        else
                            set(Data.MPwaytitleH,'String','Start:')
                            Data.WayHandles{en} = [];
                        end
                                                          % Terrain edit mode
                    elseif get(Data.MPtermodeH,'Value')
                        for en = 1:Data.NumExp
                            haspaths = ~isempty(Data.Pathpoints{en});
                            if haspaths
                                MPfigH = uisuspend(MPfig);
                                Choice = questdlg(['Editing the terrain will',...
                                                   'clear all traverse
paths.'],...
                                                  'Edit
Terrain','OK','Cancel','OK');
                                uirestore(MPfigH)
                                if strcmp(Choice,'Cancel'), return, end
                                delete(Data.PathHandles{:})
                                [Data.Pathpoints{:},Data.PathHandles{:},...

Data.Distance{:},Data.MetCost{:},Data.Time{:},...
                                 Data.Waypoints{Data.NumExp+1:end}] = deal([]);
                                set([Data.MPpathenH,Data.MPdistNH,...
                                     Data.MPcostNH,Data.MPtimeH],'String','')
                                for i = 1:Data.NumExp
                                    if ~isempty(Data.Waypoints{i})

set(Data.WayHandles{i}(1),'SizeData',120,...
```

```
                                        'CData',Data.Ecolors{mod(i-1,10)+1})
                                end
                            end
                            break
                        end
                    end
                    tmap = {'Obstacles' 'SoilMech' 'SciReturn' 'Other'};
                    Terrain = tmap{get(Data.MPtertypeH,'Value')};
                    Data.(Terrain(1:4)) = true;          % Cost map exists
                    Data.([Terrain(1:4),'Ed']) = true;  % Map edited, to be
saved
                    er = round(Data.Rows*Data.TEsize/200);
                    ec = round(Data.Cols*Data.TEsize/200); % Edit rectangle
                    [lr,ur,lc,uc] = deal(max(cly+1-
er,1),min(cly+1+er,Data.Rows),...
                                         max(clx+1-
ec,1),min(clx+1+ec,Data.Cols));
                    % Left-Click sets all values in the edit rectangle to 1
                    % Double Click sets all values to 2 (besides Obstacles)
                    % Shift+Click sets all values to zero
                    Data.(Terrain)(lr:ur,lc:uc) = (strcmp(Task,'normal') +
...

(1+~strcmp(Terrain,'Obstacles'))*strcmp(Task,'open'));
                    if strcmp(Terrain,'Obstacles')
                        Data.ColorObsRed =
min(Data.Elevations+Data.Obstacles*10^6,...
                                                Data.Elmin+Data.Eldiff*64/63);
                        Terrain = 'ColorObsRed';
                    end
                    set(Data.MPsurf,'CData',Data.(Terrain))
                end

            case 'alt'                           % Right-Click: data display
                dtext = '';  info = [];  en = [];  wp = [];
                distU = get(Data.MPscaleH,'Value');
                distR = {1,'%.0f','m'; .001,'%.2f','km';...
                         3.28084,'%.0f','ft'; .0006213712,'%.2f','mi'};
                costU = get(Data.MPcostUH,'Value');
                costR = {.2521644,'Kcal'; 1,'BTU'; 1.055056,'KJ'};
                if ~isempty(ClOnPath)    % If a path was clicked on
                    en = ClOnPath(1);    % explorer# or #+Data.NumExp for
return
                    enR = mod(en-1,Data.NumExp)+1; % explorer#
                    if en~=enR  % Return home path clicked on
                        Data.R = 'R';
                        set(MPfig,'UserData',Data)
                    end
                    set(Data.MPexpnumH,'Value',enR)      % Set overhead path
                    pathmaster('Mission','SelExp',[])    % cost displays
                    Data.R = '';
                    wp = ClOnPath(end);  % waypoint # if passed
                    if numel(ClOnPath)==1  % If no waypt passed, find nearest
                        [D,wp] = min((Data.Waypoints{en}(:,1)-clx).^2+...
%wp=Nearest
                                     (Data.Waypoints{en}(:,2)-cly).^2);    %
waypoint
```

```
                    end
                    clx = Data.Waypoints{en}(wp,1);  % Move clx,cly to the
                    cly = Data.Waypoints{en}(wp,2);  % waypoint coordinates
                    if all([clx,cly]==Data.prevwp)
                        Data.datadisp = mod(Data.datadisp,5)+1;
                    end
                    Data.prevwp = [clx,cly];
                    numwp = size(Data.Waypoints{en},1);
                    if ~isempty(Data.Pathpoints{en})
                        pp = find((Data.Pathpoints{en}(:,1)==clx) & ...
%pp=Point along
                                  (Data.Pathpoints{en}(:,2)==cly),1);   %
traverse path
                        if isempty(pp), pp=1; end
                    else
                        Data.datadisp = 5;
                    end
                    txt = {'Start Point:',sprintf('Waypoint %d:',wp-1)};
                    dtext = sprintf([txt{1+(wp>1)},'\n']);
                    while Data.datadisp<=4  % Display cost data
                        hasdata = true;
                        if Data.datadisp==1 && wp>1      % Cost from start
                            header = 'Cost from start';
                            dist = Data.Distance{en}(pp);
                            mcost = Data.MetCost{en}(pp);
                            time = Data.Time{en}(pp);
                        elseif Data.datadisp==2 && wp>1 && numwp>2
                            header = 'Cost from prev WP'; % Cost from prev
waypt
                            prevwp = Data.Waypoints{en}(wp-1,1:2);
                            prevpp =
find((Data.Pathpoints{en}(:,1)==prevwp(1))&...

(Data.Pathpoints{en}(:,2)==prevwp(2)),1);
                            if isempty(prevpp), prevpp=1; end
                            dist = Data.Distance{en}(pp)-
Data.Distance{en}(prevpp);
                            mcost = Data.MetCost{en}(pp)-
Data.MetCost{en}(prevpp);
                            time = Data.Time{en}(pp)-Data.Time{en}(prevpp);
                        elseif Data.datadisp==3 && wp<numwp && numwp>2
                            header = 'Cost to next WP';  % Cost to next
waypoint
                            nextwp = Data.Waypoints{en}(wp+1,1:2);
                            nextpp =
find((Data.Pathpoints{en}(:,1)==nextwp(1))&...

(Data.Pathpoints{en}(:,2)==nextwp(2)),1);
                            dist = Data.Distance{en}(nextpp)-
Data.Distance{en}(pp);
                            mcost = Data.MetCost{en}(nextpp)-
Data.MetCost{en}(pp);
                            time = Data.Time{en}(nextpp)-Data.Time{en}(pp);
                        elseif Data.datadisp==4 &&
wp<size(Data.Waypoints{en},1)
                            header = 'Cost to end';      %Cost to end
```

```
                            dist = Data.Distance{en}(end)-
Data.Distance{en}(pp);
                            mcost = Data.MetCost{en}(end)-
Data.MetCost{en}(pp);
                            time = Data.Time{en}(end)-Data.Time{en}(pp);
                        else
                            Data.datadisp = Data.datadisp+1;
                            hasdata = false;
                        end
                        if hasdata
                            dtext = [dtext,header,...
                                     '\nDist:  ',distR{distU,2},'
',distR{distU,3},...
                                     '\nCost: %.1f',' ',costR{costU,2},...
                                     '\nTime: %d:%d%d']; %#ok<AGROW>
                            hrmin = [floor(time/3600)
round(rem(time,3600)/60)];
                            info = [dist*distR{distU,1},
mcost*costR{costU,1},...
                                    hrmin(1), zeros(hrmin(2)<10), hrmin(2)];
                            break
                        end
                    end
                end
                if isempty(ClOnPath) || Data.datadisp == 5  % Display general
info
                    elu = {'m', 'ft', 1, 3.28084};
                    dtext = [dtext,'Elev:   %.2f',elu{round(distU/2)},...
                                   '\nSlope: %.2f°']; %#ok<AGROW>
                    info =
[Data.Elevations(cly+1,clx+1)*elu{2+round(distU/2)},...
                            Data.Slopes(cly+1,clx+1)];
                    if Data.UTMzone~=0                          % Get Lat/Long
                    % Lat/Long: uwgb.edu/dutchs/UsefulData/UTMFormulas.htm
                    x   = Data.xllcorner+(clx+.5)*Data.Resolution-500000;
                    y   = Data.yllcorner+(Data.Rows-1-
(cly+.5))*Data.Resolution-...
                             10000000*(Data.UTMzone<0);
                    e   = (1-6356752.314^2/6378137^2)^(1/2);
                    mu  = y/(.9996*6378137*(1-e^2/4-3/64*e^4-5/256*e^6));
                    e1  = (1-(1-e^2)^(1/2))/(1+(1-e^2)^(1/2));
                    J   = [3/2*e1-27/32*e1^3, 21/16*e1^2-55/32*e1^4,...
                           151/96*e1^3,       1097/512*e1^4];
                    fp =
mu+J(1)*sin(2*mu)+J(2)*sin(4*mu)+J(3)*sin(6*mu)+J(4)*sin(8*mu);
                    e2  = e^2/(1-e^2);
                    C   = e2*cos(fp)^2;
                    T   = tan(fp)^2;
                    R   = 6378137*(1-e^2)/(1-e^2*sin(fp)^2)^(3/2);
                    N   = 6378137/(1-e^2*sin(fp)^2)^(1/2);
                    D   = x/(.9996*N);
                    Qa  = [N*tan(fp)/R, D^2/2, (5+3*T+10*C-4*C^2-
9*e2)*D^4/24,...
                           (61+90*T+298*C+45*T^2-3*C^2-252*e2)*D^6/720];
                    Qo  = [D, (1+2*T+C)*D^3/6, (5-2*C+28*T-
3*C^2+8*e2+24*T^2)*D^5/120];
                    lat   = (fp-Qa(1)*(Qa(2)-Qa(3)+Qa(4)))*180/pi;
```

```
                    long = abs(Data.UTMzone)*6-183+((Qo(1)-
Qo(2)+Qo(3))/cos(fp))*180/pi;
                    LAT  = [fix(lat)  fix(rem(lat,1)*60)
rem(rem(lat,1)*60,1)*60];
                    LONG = [fix(long) fix(rem(long,1)*60)
rem(rem(long,1)*60,1)*60];
                    % ***************************************************
                    dtext = [dtext,'\nLat:   %d° %d'' %.2f"',...
                                   '\nLong: %d° %d'' %.2f"'];
                    info =
[info,LAT(1),abs(LAT(2:3)),LONG(1),abs(LONG(2:3))];
                end
                for tmap = {'SoilMech' 'SciReturn' 'Other'}
                    if Data.(tmap{1}(1:4))
                        dtext = [dtext,'\n',tmap{1}(1:5),':  %d'];
%#ok<AGROW>
                        info = [info,Data.(tmap{1})(cly+1,clx+1)];
%#ok<AGROW>
                    end
                end
            end
            aln = {'left','right','bottom','top'};   % Text alignments
            hold on
            Data.minfo(1) = scatter3(Data.MPaxes,clx,cly,...
                  Data.Elevations(cly+1,clx+1)+.05*Data.Eldiff,60,[0 1
0],'filled');
            Data.minfo(2) = text(clx,cly,...
                Data.Elevations(cly+1,clx+1)+.2*Data.Eldiff,...
                sprintf(dtext,info),...
                'BackgroundColor',[.92 .865 .7],...
                'HorizontalAlignment',aln{1+(clx>.8*Data.Cols)},...
                'VerticalAlignment',aln{3+(cly<.1*Data.Rows)},...

'ButtonDownFcn',sprintf('pathmaster(''Mission'',''Click'',[%d %d])',en,wp));
            hold off
        end
        set(MPfig,'UserData',Data)

    case 'Resize'  % Resize GUI window
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        figsize = get(MPfig,'Position');
        set(Data.MPmenu,'Position',[0 figsize(4)-64 figsize(3)+2 66]);
        set(Data.MPaxes,'Position',[50 24 figsize(3)-65 figsize(4)-90]);

    case 'Close'  % Close GUI, save edits, exit pathmaster
        MPfig = gcf;
        Data = get(MPfig,'UserData');
        if any([Data.ObstEd,Data.SoilEd,Data.SciREd,Data.OtheEd,Data.WayPEd])
            Choice = questdlg(sprintf([Data.EVAname,' has been
edited\n\n',...
                        'Exit without running PATH?']),'Exit
Pathmaster...',...
                        'Save edits','Don''t save','Cancel','Cancel');
        else
            Choice = questdlg(['Finished with ',Data.EVAname,'?'],...
```

```
                                'Exit Pathmaster...','Yes','Cancel','Cancel');
        end
        if ~strcmp(Choice,'Cancel')
            delete(MPfig)
            if strcmp(Choice,'Save edits')
                try save([Data.EVAname,'_Data'],'-
struct','Data','Waypoints','-append'), catch end %#ok<CTCH>
                Data.Path = true;
                pathmaster('SaveMaps','-append',Data)      % Call to
'SaveMaps'
            end
        end
end

%% ************ UPDATE DATA & FILES & MISSION GUI ************************
case 'Update'        % Runs after callback to Map Info or EVA Data GUI
if ~isempty(Select)
message = '';
header = ['Changing these data values may clear\n',...
          'or re-write the following data:\n\n'];
for task = {{'ClrPaths','All traverse paths'} {'Obs','Obstacles'} ...
            {'NewMaps','Map data files'}}
    if any(strcmp(task{1}{1},Select))
        message = [header,message,'- ',task{1}{2},'\n']; %#ok<AGROW>
        header = '';
    end
end
AddEx = sum(strcmp('AddExp',Select));  % Number of added explorers
NumExp = Data.NumExp-AddEx;            % Previous number of explorers
if ~any(strcmp('ClrPaths',Select))
    ClrPath = [];
    for task = Select
        if ~ischar(task{1}) && task{1}<=NumExp && ...
           ~isempty(Data.Pathpoints{task{1}}) && ~any(ClrPath==task{1})
            ClrPath = [ClrPath, task{1}]; %#ok<AGROW>
        end
    end
    if ~isempty(ClrPath)
        message = [header,message,'- Traverse path(s): ',...
                   num2str(sort(ClrPath),' %d'),'\n'];
    end
else
    ClrPath = 1:NumExp;
end
if any(strcmp('NewFiles',Select))
    message = [message,'\nOnly paths created after this point will be\n',...
                       'written with the new name and/or directories.'];
end
if ~isempty(message)           % Asks if it's OK to make applicable changes
    Choice = questdlg(sprintf(message),'Change Data
Values...','OK','Cancel','OK');
    if strcmp(Choice,'Cancel')  % Cancel without saving changes
        set(Data.MPfig,'Visible','on')
        return
    end
end
if any(strcmp('Scale',Select))             % Rescale map
```

```
    [gx,gy] = gradient(Data.Elevations,Data.Resolution);
    Data.Slopes = atan(sqrt(gx.^2+gy.^2))*(180/pi);  % Slopes in degrees
    mapsize = Data.Resolution*max(Data.Rows,Data.Cols);
    zaspect = Data.Resolution*min(1,10*Data.Eldiff/mapsize);
    set(Data.MPaxes,'DataAspectRatio',[1 1 zaspect])
    set(Data.MPfig,'UserData',Data)
    pathmaster('Mission','Scale',[])  % Call scaling routine
end
if ~Data.Obst && any(strcmp('Obs',Select)) %Recalculate obstacles
    Data.Obstacles = Data.Slopes > Data.MaxSlope; % 1 if obstacle, else 0
    Data.Obst = true;
    Data.ColorObsRed = min(Data.Elevations+Data.Obstacles*10^6,...
                           Data.Elmin+Data.Eldiff*64/63);
    if get(Data.MPtertypeH,'Value')==1
        set(Data.MPsurf,'CData',Data.ColorObsRed)
    end
end
if any(strcmp('NewMaps',Select)) || any(strcmp('NewFiles',Select)) %New map
files
    pathmaster('SaveMaps','',Data)         % Call to 'SaveMaps'
    set(Data.MPfig,'Name',['Pathmaster:  ',Data.EVAname,' - Mission
Planner'])
end
for en = ClrPath                           % Clear paths
    delete(Data.PathHandles{[en,en+NumExp]})
    [Data.Pathpoints{[en,en+NumExp]},Data.PathHandles{[en,en+NumExp]},...
     Data.Distance{[en,en+NumExp]},Data.MetCost{[en,en+NumExp]},...
     Data.Time{[en,en+NumExp]},Data.Waypoints{en+NumExp}] = deal([]);
    if ~isempty(Data.Waypoints{en})

set(Data.WayHandles{en}(1),'SizeData',120,'CData',Data.Ecolors{mod(en-
1,10)+1})
    end
end
if ~isempty(ClrPath)
    try save([Data.EVAname,'_Data'],'-
struct','Data','Pathpoints','Distance',...
                                           'MetCost','Time','-append'), catch
%#ok<CTCH>
    end
end
if any(ClrPath==get(Data.MPexpnumH,'Value'))  % Clear cost displays

set([Data.MPpathenH,Data.MPdistNH,Data.MPcostNH,Data.MPtimeH],'String','')
end
if AddEx                                   % Add explorer
    [newexp{1:AddEx,1:2}] = deal([]);
    for vars = {'Waypoints' 'WayHandles' 'Pathpoints' 'PathHandles',...
                'Distance' 'MetCost' 'Time'}
        Data.(vars{1}) = vertcat(Data.(vars{1}),newexp);
    end
    set(Data.MPexpnumH,'String',1:Data.NumExp)
    try save([Data.EVAname,'_Data'],'-struct','Data','Explorers','-append'),
catch end %#ok<CTCH>
end
if any(strcmp('Planet',Select))            % Change planet
    pathmaster('Mission',Data.Planet,[])
```

```
end
if isempty(Data.Lite) && any(strcmp('Sun',Select))  % Move sun
    set(Data.Sun,'Position',[sin((Data.Hour+Data.Minute/60)*pi/12),...
                -cos((Data.Hour+Data.Minute/60)*pi/12),...
                .014+.006*sin((Data.Hour+Data.Minute/60)*pi/24)])
end
set(Data.MPfig,'UserData',Data)            % Save Data changes
end
set(Data.MPfig,'Visible','on')             % Re-open Mission GUI

%% ************ RUN THE PATH OPTIMIZATION *******************************
case 'PATH'
enR = mod(Select-1,Data.NumExp)+1;  % enR is explorer# even for return home
message = ['Running traverse optimization...\n',...
           '\nTraverse path%s: ',num2str(enR,' %d,')];
s = {'','s'};
pathmsg = helpdlg([sprintf(message(1:end-
1),s{1+(length(Select)>1)}),Data.R],'Pathmaster');
                                  % Make sparse copies of terrain cost maps
[obstacles,soilmech,scireturn,other] = deal(sparse(Data.Obstacles),...
    sparse(Data.SoilMech),sparse(Data.SciReturn),sparse(Data.Other));
%#ok<NASGU>
elevation = Data.Elevations;
resolution = Data.Resolution;
obstacles(isnan(obstacles)) = 1;  % Clean up any NaNs in the maps
[elevation(isnan(elevation)),soilmech(isnan(soilmech)),...
    scireturn(isnan(scireturn)),other(isnan(other))] = deal(0); %#ok<NASGU>
grav = [1,1/6,1/3];
gravity = 9.8*grav(Data.Planet);  % Set the planet gravity
[dimr,dimc] = deal(Data.Rows,Data.Cols);
E = sparse([5,ones(1,dimc-2),6;4*ones(dimr-2,1),zeros(dimr-2,dimc-
2),2*ones(dimr-2,1);8,3*ones(1,dimc-2),7]);
path_err = false;  % Signals an error in finding traverse paths
Data.errpath = [];
for en = Select
    enR = mod(en-1,Data.NumExp)+1;
    mass = Data.Weight(enR);
    waypoints_x_y = Data.Waypoints{en}(:,1:2)+1; %(x,y) coords to matrix
index
    waypoints = Data.Waypoints{en}(:,1)*dimr+Data.Waypoints{en}(:,2)+1;
    % /////// Optimization Routine: Brandon Johnson, August 5, 2008 ///////
    [Distance,Cost,Time] = deal(0);
    smoothed_route = waypoints(1);
    [skipped,skip_waypoint,straight_line] = deal(false);
    altwaypoints = ones(1,length(waypoints));
    altwaypoints(2:2:end) = 0;  % Used to alternate which matrices are for
start or finish
    % Waypoint Loop
    for k = 1:(length(waypoints)-1)
        start = waypoints(k);
        finish = waypoints(k+1);
        if skip_waypoint  % Used if next waypoint already in visited range
            skip_waypoint = false;
            continue
        end
        % Initialization, matrices have to be alternatively reset after each
waypoint
```

```
        if  k~=1 && ~altwaypoints(k) && ~skipped
            I = finish;
            previousa = zeros(dimr,dimc);
            testcosta = sparse(dimr,dimc);
            Fcosta = sparse(dimr,dimc);
            costa = zeros(dimr,dimc);
            timea = zeros(dimr,dimc);
            distancea = zeros(dimr,dimc);
            costa(finish) = 0.00001;
        elseif k~=1 && altwaypoints(k) && ~skipped
            J = finish;
            previousb = zeros(dimr,dimc);
            testcostb = sparse(dimr,dimc);
            Fcostb = sparse(dimr,dimc);
            costb = zeros(dimr,dimc);
            timeb = zeros(dimr,dimc);
            distanceb = zeros(dimr,dimc);
            costb(finish) = 0.00001;
        elseif k==1 || skipped
            I = start;  J = finish;
            previousa = zeros(dimr,dimc);
            previousb = zeros(dimr,dimc);
            testcosta = sparse(dimr,dimc);
            testcostb = sparse(dimr,dimc);
            Fcosta = sparse(dimr,dimc);
            Fcostb = sparse(dimr,dimc);
            costa = zeros(dimr,dimc);
            costb = zeros(dimr,dimc);
            timea = zeros(dimr,dimc);
            timeb = zeros(dimr,dimc);
            distancea = zeros(dimr,dimc);
            distanceb = zeros(dimr,dimc);
            costa(start) = 0.00001;  costb(finish) = 0.00001;
            if skipped
                if ~altwaypoints(k)
                    altwaypoints = ~altwaypoints;
                end
            end
        end
        % Main Loop
        while I ~= J
        % Determines neighboring nodes of I
            x = ceil(I/dimr);
            y = mod(I-1,dimr)+1;  % Find (x,y) coorinates of I
            if E(I) == 0  % Middle
                II = [I-1 I+dimr I+1 I-dimr I+dimr-1 I+dimr+1 I-dimr+1 I-
dimr-1]; %all
                IIs = [x y-1; x+1 y; x y+1; x-1 y; x+1 y-1; x+1 y+1; x-1 y+1;
x-1 y-1];
            elseif E(I) == 1  % Top edge
                II = [I+dimr I+1 I-dimr I+dimr+1 I-dimr+1]; %right down left
bottomright bottomleft
                IIs = [x+1 y; x y+1; x-1 y; x+1 y+1; x-1 y+1];
            elseif E(I) == 2  % Right edge
                II = [I-1 I+1 I-dimr I-dimr+1 I-dimr-1]; %up down left
bottomleft topleft
                IIs = [x y-1; x y+1; x-1 y; x-1 y+1; x-1 y-1];
```

```
            elseif E(I) == 3  % Bottom edge
                II = [I-1 I+dimr I-dimr I+dimr-1 I-dimr-1]; %up right left
topright topleft
                IIs = [x y-1; x+1 y; x-1 y; x+1 y-1; x-1 y-1];
            elseif E(I) == 4  % Left edge
                II = [I-1 I+dimr I+1 I+dimr-1 I+dimr+1]; %up right down
topright bottomright
                IIs = [x y-1; x+1 y; x y+1; x+1 y-1; x+1 y+1];
            elseif E(I) == 5  % Top left corner
                II = [I+dimr I+1 I+dimr+1]; %right down bottomright
                IIs = [x+1 y; x y+1; x+1 y+1];
            elseif E(I) == 6  % Top right corner
                II = [I+1 I-dimr I-dimr+1]; %down left bottomleft
                IIs = [x y+1; x-1 y; x-1 y+1];
            elseif E(I) == 7  % Bottom right corner
                II = [I-1 I-dimr I-dimr-1]; %up left topleft
                IIs = [x y-1; x-1 y; x-1 y-1];
            else  % Bottom left corner
                II = [I-1 I+dimr I+dimr-1]; %up right topright
                IIs = [x y-1; x+1 y; x+1 y-1];
            end
            remov_val = ~(~costa(II) & ~obstacles(II));
            II(remov_val) = [];
            IIs(remov_val,:) = [];
        % ******************* COST FUNCTION: METABOLIC ********************
            % Calculate the local costs for I using the Metabolic Cost
function
            % Distance from I to II
            diag = (II==I-dimr-1 | II==I+dimr-1 | II==I+dimr+1 | II==I-
dimr+1); %diagonals
            cart = (II==I-1 | II==I+dimr | II==I+1 | II==I-dimr); %up, right,
down, left
            dist = zeros(1,length(II));
            dist(diag) = resolution*sqrt(2); % Diagonal dist is sqrt(2)
greater
            dist(cart) = resolution;
            % Slope from I to II
            if altwaypoints(k)  % Correct direction always applied
                slope = 180/pi*atan((elevation(II)-elevation(I))./dist);
            else
                slope = 180/pi*atan((elevation(I)-elevation(II))./dist);
            end
            % Velocity as a function of slope
            V = zeros(1,length(II));
            a = (slope<=-20 | slope>=15);
              V(a) = 0.05;
            b = (slope>-20 & slope<=-10);
              V(b) = 0.095*slope(b)+1.95;
            c = (slope>-10 & slope<0);
              V(c) = 0.06*slope(c)+1.6;
            d = (slope>=0 & slope<6);
              V(d) = -0.02*slope(d)+1.6;
            e = (slope>=6 & slope<15);
              V(e) = -0.039*slope(e)+0.634;
            % Power
            Rfactor = 0.661*V.*cos(pi*slope/180)+0.115;
            power = zeros(1,length(Rfactor));
```

```
            f = (slope<0);
              power(f) =
(2.4*mass*gravity*V(f).*sin(pi*slope(f)/180).*(0.3.^(abs(slope(f))/7.65)))+((
3.28*mass+71.1)*Rfactor(f));
            g = (slope>0);
              power(g) =
(3.5*mass*gravity*V(g).*sin(pi*slope(g)/180))+((3.28*mass+71.1)*Rfactor(g));
            h = (slope==0);
              power(h) = (3.28*mass+71.1)*Rfactor(h);
            % Metabolic Cost
            Metcost = power.*dist./V;
            C = Metcost*0.00094781712;  % Metcost converted to BTUs
            % Heuristic Function for A* Algorithm
            % P. Amit, see
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
            H_straight = zeros(1,length(C));
            H_diagonal = zeros(1,length(C));
            for i = 1:size(IIs,1)
                if altwaypoints(k)
                    xx = abs(IIs(i,1)-waypoints_x_y(k+1,1));
                    yy = abs(IIs(i,2)-waypoints_x_y(k+1,2));
                else
                    xx = abs(IIs(i,1)-waypoints_x_y(k,1));
                    yy = abs(IIs(i,2)-waypoints_x_y(k,2));
                end
                H_straight(i) = xx+yy;
                if xx > yy
                    H_diagonal(i) = yy;
                else
                    H_diagonal(i) = xx;
                end
            end
            H = (resolution*544.9/1.6*0.00094781712)*(H_straight-
2*H_diagonal)...
                +(sqrt(2)*resolution*544.9/1.6*0.00094781712)*H_diagonal;
            D = (~testcosta(II) | costa(I)+C<testcosta(II));
            % Store the costs
            testcosta(II(D)) = costa(I)+C(D);  % Total metabolic cost
            Fcosta(II(D)) = costa(I)+C(D)+H(D);  % Metabolic + Heuristic
estimate
            timea(II(D)) = timea(I)+dist(D)./V(D);  % Total time
            distancea(II(D)) = distancea(I)+dist(D);  % Total distance
            previousa(II(D)) = I;  % Used to back-track for finding the route
        % ********************** END OF COST FUNCTION **********************
            % Determines neighboring nodes of J
            x = ceil(J/dimr);
            y = mod(J-1,dimr)+1;
            if E(J) == 0  % Middle
                JJ = [J-1 J+dimr J+1 J-dimr J+dimr-1 J+dimr+1 J-dimr+1 J-
dimr-1]; %all
                JJs = [x y-1; x+1 y; x y+1; x-1 y; x+1 y-1; x+1 y+1; x-1 y+1;
x-1 y-1];
            elseif E(J) == 1  % Top edge
                JJ = [J+dimr J+1 J-dimr J+dimr+1 J-dimr+1]; %right down left
bottomright bottomleft
                JJs = [x+1 y; x y+1; x-1 y; x+1 y+1; x-1 y+1];
            elseif E(J) == 2  % Right edge
```

```
                JJ = [J-1 J+1 J-dimr J-dimr+1 J-dimr-1]; %up down left
bottomleft topleft
                JJs = [x y-1; x y+1; x-1 y; x-1 y+1; x-1 y-1];
            elseif E(J) == 3  % Bottom edge
                JJ = [J-1 J+dimr J-dimr J+dimr-1 J-dimr-1]; %up right left
topright topleft
                JJs = [x y-1; x+1 y; x-1 y; x+1 y-1; x-1 y-1];
            elseif E(J) == 4  % Left edge
                JJ = [J-1 J+dimr J+1 J+dimr-1 J+dimr+1]; %up right down
topright bottomright
                JJs = [x y-1; x+1 y; x y+1; x+1 y-1; x+1 y+1];
            elseif E(J) == 5  % Top left corner
                JJ = [J+dimr J+1 J+dimr+1]; %right down bottomright
                JJs = [x+1 y; x y+1; x+1 y+1];
            elseif E(J) == 6  % Top right corner
                JJ = [J+1 J-dimr J-dimr+1]; %down left bottomleft
                JJs = [x y+1; x-1 y; x-1 y+1];
            elseif E(J) == 7  % Bottom right corner
                JJ = [J-1 J-dimr J-dimr-1]; %up left topleft
                JJs = [x y-1; x-1 y; x-1 y-1];
            else  % Bottom left corner
                JJ = [J-1 J+dimr J+dimr-1]; %up right topright
                JJs = [x y-1; x+1 y; x+1 y-1];
            end
            remov_val = ~(~costb(JJ) & ~obstacles(JJ));
            JJ(remov_val) = [];
            JJs(remov_val,:) = [];
        % ******************* COST FUNCTION: METABOLIC *******************
            % Calculate the local costs for J using the Metabolic Cost
Function
            % Distance from J to JJ
            diag = (JJ==J-dimr-1 | JJ==J+dimr-1 | JJ==J+dimr+1 | JJ==J-
dimr+1); %diagonals
            cart = (JJ==J-1 | JJ==J+dimr | JJ==J+1 | JJ==J-dimr); %up, right,
down, left
            dist = zeros(1,length(JJ));
            dist(diag) = resolution*sqrt(2);  % Diagonal dist is sqrt(2)
greater
            dist(cart) = resolution;
            % Slope from J to JJ
            if ~altwaypoints(k)
                slope = 180/pi*atan((elevation(JJ)-elevation(J))./dist);
            else
                slope = 180/pi*atan((elevation(J)-elevation(JJ))./dist);
            end
            % Velocity as a function of slope
            V = zeros(1,length(JJ));
            a = (slope<=-20 | slope>=15);
              V(a) = 0.05;
            b = (slope>-20 & slope<=-10);
              V(b) = 0.095*slope(b)+1.95;
            c = (slope>-10 & slope<0);
              V(c) = 0.06*slope(c)+1.6;
            d = (slope>=0 & slope<6);
              V(d) = -0.02*slope(d)+1.6;
            e = (slope>=6 & slope<15);
              V(e) = -0.039*slope(e)+0.634;
```

```
            % Power
            Rfactor = 0.661*V.*cos(pi*slope/180)+0.115;
            power = zeros(1,length(Rfactor));
            f = (slope<0);
              power(f) =
(2.4*mass*gravity*V(f).*sin(pi*slope(f)/180).*(0.3.^(abs(slope(f))/7.65)))+((
3.28*mass+71.1)*Rfactor(f));
            g = (slope>0);
              power(g) =
(3.5*mass*gravity*V(g).*sin(pi*slope(g)/180))+((3.28*mass+71.1)*Rfactor(g));
            h = (slope==0);
              power(h) = (3.28*mass+71.1)*Rfactor(h);
            % Metabolic Cost
            Metcost = power.*dist./V;
            C = Metcost*0.00094781712;  % Metcost converted to BTUs
            H_straight = zeros(1,length(C));
            H_diagonal = zeros(1,length(C));
            for i = 1:size(JJs,1)
                if altwaypoints(k)
                    xx = abs(JJs(i,1)-waypoints_x_y(k,1));
                    yy = abs(JJs(i,2)-waypoints_x_y(k,2));
                else
                    xx = abs(JJs(i,1)-waypoints_x_y(k+1,1));
                    yy = abs(JJs(i,2)-waypoints_x_y(k+1,2));
                end
                H_straight(i) = xx+yy;
                if xx > yy
                    H_diagonal(i) = yy;
                else
                    H_diagonal(i) = xx;
                end
            end
            H = (resolution*544.9/1.6*0.00094781712)*(H_straight-
2*H_diagonal)...
                +(sqrt(2)*resolution*544.9/1.6*0.00094781712)*H_diagonal;
            D = (~testcostb(JJ) | costb(J)+C<testcostb(JJ));
            % Store the costs
            testcostb(JJ(D)) = costb(J)+ C(D);  % Total metabolic cost
            Fcostb(JJ(D)) = costb(J)+C(D)+H(D);  % Metabolic + Heuristic
estimate
            timeb(JJ(D)) = timeb(J)+dist(D)./V(D);  % Total time
            distanceb(JJ(D)) = distanceb(J)+dist(D);  % Total distance
            previousb(JJ(D)) = J;  % Used to back-track for finding the route
        % ********************* END OF COST FUNCTION **********************
            testcosta(I) = 0;
            testcostb(J) = 0;
            Fcosta(I) = 0;
            Fcostb(J) = 0;
        % Find minimum value in both Fcosts
            K = find(Fcosta);
            L = find(Fcostb);
            [v,N] = min(Fcosta(K));
            [v,M] = min(Fcostb(L));
            I = K(N);
            J = L(M);
        % Update costs & check if paths intersect
            costa(I) = testcosta(I);
```

```
            costb(J) = testcostb(J);
            if costb(I);  % Tests for intersection of paths
                j = I;
                break;
            end
            if costa(J);
                j = J;
                break;
            end
        end  % End of main loop
        % Traverse path is solved!  Now find the route
        if ~exist('j','var'), path_err = true; break, end  % Signals error,
break
        route = [j,zeros(1,dimr+dimc)];
        count = 1;
        if altwaypoints(k)
            i = previousa(j);
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousa(i);
            end
            route1 = route(find(route,1,'last'):-1:1);
            route = [j,zeros(1,dimr+dimc)];
            count = 1;
            i = previousb(j);
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousb(i);
            end
            route2 = route(1:find(route,1,'last'));
        else
            i = previousb(j);
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousb(i);
            end
            route1 = route(find(route,1,'last'):-1:1);
            route = [j,zeros(1,dimr+dimc)];
            count = 1;
            i = previousa(j);
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousa(i);
            end
            route2 = route(1:find(route,1,'last'));
        end
        testline = Midpoint(route1(1),route2(end)); %Tests if route is
straight to simplify smoothing
        if length(testline)==length([route1(2:end-1) route2]) && all(testline
== [route1(2:end-1) route2])
            new_route = route2(end);
            straight_line = 1;
        else
```

```
            new_route = Smooth([route1(1:end-1) route2]); % Call smoothing
function
            new_route = new_route(2:end);
        end
        smoothed_route = [smoothed_route,new_route]; %#ok<AGROW>
        Update_lists  % Update stored costs
        straight_line = 0;
        clear j
        % Check if next waypoint has already been visited, and finds path
        if (waypoints(k+1)~=waypoints(end)) && ~altwaypoints(k+1) &&
costb(waypoints(k+2))
            route = [waypoints(k+2),zeros(1,dimr+dimc)];
            i = previousb(waypoints(k+2));
            count = 1;
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousb(i);
            end
            route1 = route(find(route,1,'last'):-1:1);
            route1 = Smooth(route1);
            smoothed_route = [smoothed_route,route1(2:end)]; %#ok<AGROW>
            skipped = true;
            skip_waypoint = true;
            new_route = route1(2:end);
            Update_lists  % Update stored costs
        elseif (waypoints(k+1)~=waypoints(end)) && altwaypoints(k+1) &&
costa(waypoints(k+2))
            route = [waypoints(k+2),zeros(1,dimr+dimc)];
            i = previousa(waypoints(k+2));
            count = 1;
            while i ~= 0
                count = count+1;
                route(count) = i;
                i = previousa(i);
            end
            route1 = route(find(route,1,'last'):-1:1);
            route1 = Smooth(route1);
            smoothed_route = [smoothed_route,route1(2:end)];  %#ok<AGROW>
            skipped = true;
            skip_waypoint = true;
            new_route = route1(2:end);
            Update_lists  % Update stored costs
        end
    end
    % ////////////////// END OF OPTIMIZATION ROUTINE ///////////////////////
    if path_err  % Handle any path errors
        Data.errpath = [Data.errpath,enR];  %#ok<AGROW>
        Select = Select(Select~=en);  % Purge from new path list
        path_err = false;
        continue
    end
    Data.Pathpoints{en} = [floor((smoothed_route-.5)/dimr);
mod(smoothed_route-1,dimr)].';
    [Data.Distance{en},Data.MetCost{en},Data.Time{en}] =
deal(Distance.',Cost.',Time.');
end
```

```
try delete(pathmsg), catch end  %#ok<CTCH>

%% ************ STORE PATHS, WRITE RENDER & COORD FILES ******************
for en = Select
    cd(Data.Render_dir)
    enR = mod(en-1,Data.NumExp)+1;
    for Outfile = {Data.EVAname,'Current'}    % Write Waypoint render files
        wrf =
fopen(sprintf([Outfile{1},'_Waypoints%d%s.txt'],enR,Data.R),'wt');
        fprintf(wrf,'way%d %d %d\n',[(1:size(Data.Waypoints{en},1)).',...
            Data.Waypoints{en}(:,1),Data.Rows-1-Data.Waypoints{en}(:,2)].');
        fclose(wrf);                              % Write Traverse render files
        trf =
fopen(sprintf([Outfile{1},'_Traverse%d%s.txt'],enR,Data.R),'wt');
        fprintf(trf,'path%d %d %d\n',[(1:length(Data.Distance{en})).',...
            Data.Pathpoints{en}(:,1),Data.Rows-1-
Data.Pathpoints{en}(:,2)].');
        fclose(trf);                              % Write Cost render files
        crf = fopen(sprintf([Outfile{1},'_Costs%d%s.txt'],enR,Data.R),'wt');
        fprintf(crf,'cost%d %.2f %.2f
%.2f\n',[(1:length(Data.Distance{en})).',...
            Data.Distance{en},Data.Time{en},Data.MetCost{en}].');
        fclose(crf);
    end
    for i = 1:length(Data.Distance{en}); % Append elevations at path coords
        Data.Pathpoints{en}(i,3) =
Data.Elevations(Data.Pathpoints{en}(i,2)+1,...

Data.Pathpoints{en}(i,1)+1);
    end

    % The following is an example of exporting traverse Lat/Long data in text
files
    % This may be deleted if not desired
    if Data.UTMzone ~= 0
        [LAT,LONG] = deal(zeros(i,3));
        for k = 1:i  % Lat/Long at each Pathpoint
            % Lat/Long: uwgb.edu/dutchs/UsefulData/UTMFormulas.htm
            x  =
Data.xllcorner+(Data.Pathpoints{en}(k,1)+.5)*Data.Resolution-500000;
            y  = Data.yllcorner+(Data.Rows-1-
(Data.Pathpoints{en}(k,2)+.5))*...
                                  Data.Resolution-10000000*(Data.UTMzone<0);
            e  = (1-6356752.314^2/6378137^2)^(1/2);
            mu = y/(.9996*6378137*(1-e^2/4-3/64*e^4-5/256*e^6));
            e1 = (1-(1-e^2)^(1/2))/(1+(1-e^2)^(1/2));
            J  = [3/2*e1-27/32*e1^3, 21/16*e1^2-55/32*e1^4,...
                  151/96*e1^3,       1097/512*e1^4];
            fp =
mu+J(1)*sin(2*mu)+J(2)*sin(4*mu)+J(3)*sin(6*mu)+J(4)*sin(8*mu);
            e2 = e^2/(1-e^2);
            C  = e2*cos(fp)^2;
            T  = tan(fp)^2;
            R  = 6378137*(1-e^2)/(1-e^2*sin(fp)^2)^(3/2);
            N  = 6378137/(1-e^2*sin(fp)^2)^(1/2);
            D  = x/(.9996*N);
```

```
            Qa = [N*tan(fp)/R, D^2/2, (5+3*T+10*C-4*C^2-9*e2)*D^4/24,...
                  (61+90*T+298*C+45*T^2-3*C^2-252*e2)*D^6/720];
            Qo = [D, (1+2*T+C)*D^3/6, (5-2*C+28*T-
3*C^2+8*e2+24*T^2)*D^5/120];
            lat  = (fp-Qa(1)*(Qa(2)-Qa(3)+Qa(4)))*180/pi;
            long = abs(Data.UTMzone)*6-183+((Qo(1)-
Qo(2)+Qo(3))/cos(fp))*180/pi;
            LAT(k,:)  = [fix(lat)  fix(rem(lat,1)*60)
rem(rem(lat,1)*60,1)*60];
            LONG(k,:) = [fix(long) fix(rem(long,1)*60)
rem(rem(long,1)*60,1)*60];
        end
        cd([Data.Work_dir,'Traverse_Coordinates'])
        pcf =
fopen(sprintf([Data.EVAname,'_Coords%d%s.txt'],enR,Data.R),'wt');
        fprintf(pcf,'Explorer %d%s  Lat , Long:\n',enR,Data.R);
        fprintf(pcf,'point%d  %.0f %.0f %.2f , %.0f %.0f %.2f\n',...

[(1:length(Data.Distance{en})).',LAT(:,1),abs(LAT(:,2:3)),LONG(:,1),abs(LONG(
:,2:3))].');
        fclose(pcf);
    end

end
cd(Data.Work_dir)  % Append paths & costs to Matlab data file
try save([Data.EVAname,'_Data'],'-struct','Data','Pathpoints','Distance',...
     'MetCost','Time','-append'), catch  %#ok<CTCH>
end
Data.Newpaths = Select;           % Note all successful paths
set(Data.MPfig,'UserData',Data)   % Store all path and cost data

%% ************ INCORRECT 3 ARGUMENT CALL TO PATHMASTER ******************
otherwise
disp('Error: Incorrect call to pathmaster')

%% End of Progress switch
end

%% ************ PATH OPTIMIZATION SUBFUNCTIONS ***************************
% Update Cost, Distance, & Time lists
function Update_lists
    prev_cost = Cost(end);
    prev_distance = Distance(end);
    prev_time = Time(end);
    if ~straight_line
        if altwaypoints(k)
            for p = new_route
                if ~costb(p) && ~skipped
                    Cost(end+1) = prev_cost+costa(p); %#ok<AGROW>
                    Distance(end+1) = prev_distance+distancea(p); %#ok<AGROW>
                    Time(end+1) = prev_time+timea(p); %#ok<AGROW>
                elseif ~skipped
                    Cost(end+1) = prev_cost+((costa(j)+costb(j))-costb(p));
%#ok<AGROW>
                    Distance(end+1) =
prev_distance+((distancea(j)+distanceb(j))-distanceb(p)); %#ok<AGROW>
```

```
                    Time(end+1) = prev_time+((timea(j)+timeb(j))-timeb(p));
%#ok<AGROW>
                else
                    Cost(end+1) = prev_cost+costb(p); %#ok<AGROW>
                    Distance(end+1) = prev_distance+distanceb(p); %#ok<AGROW>
                    Time(end+1) = prev_time+timeb(p); %#ok<AGROW>
                end
            end
        else
            for p = new_route
                if ~costa(p)
                    Cost(end+1) = prev_cost+costb(p); %#ok<AGROW>
                    Distance(end+1) = prev_distance+distanceb(p); %#ok<AGROW>
                    Time(end+1) = prev_time+timeb(p); %#ok<AGROW>
                elseif ~skipped
                    Cost(end+1) = prev_cost+((costa(j)+costb(j))-costa(p));
%#ok<AGROW>
                    Distance(end+1) =
prev_distance+((distancea(j)+distanceb(j))-distancea(p)); %#ok<AGROW>
                    Time(end+1) = prev_time+((timea(j)+timeb(j))-timea(p));
%#ok<AGROW>
                else
                    Cost(end+1) = prev_cost+costa(p); %#ok<AGROW>
                    Distance(end+1) = prev_distance+distancea(p); %#ok<AGROW>
                    Time(end+1) = prev_time+timea(p); %#ok<AGROW>
                end
            end
        end
    else
        Cost(end+1) = prev_cost+costa(j)+costb(j);
        Distance(end+1) = prev_distance+distancea(j)+distanceb(j);
        Time(end+1) = prev_time+timea(j)+timeb(j);
    end
end

% Path Smoothing
function [smooth_route] = Smooth(route)
    smooth_route = route;
    remove_val = zeros(1,length(smooth_route));
    remove_val(1) = 1; %find(remove_val,1,'first') wont give an empty matrix
    p = 1;
    while p < length(route)
        u = route(p);
        for ip = (2+p):length(route)
            v = route(ip);
            line1 = Midpoint(u,v);  % Calls Midpoint function
            if length(line1)==length(route(p:ip)) && all(line1==route(p:ip))
%If route(p:ip) is straight, remove points between
                remove_val(find(remove_val,1,'last')+1) = ip-1;
            elseif v==u-dimr-1 || v==u+dimr-1 || v==u+dimr+1 || v==u-dimr+1
                remove_val(find(remove_val,1,'last')+1) = ip-1;
            else
                break;
            end
        end
        p = ip-1;
    end
```

```
    remove_val(1) = [];
    remove_val = remove_val(1:find(remove_val,1,'last'));
    smooth_route(remove_val) = [];
end

% Midpoint Algorithm
% Modified from N. Chattrapiban's version of Bresenham's Algorithm
% Used to find the straightest path between two points
function [myline] = Midpoint(a,b)
    x = ceil([a b]/dimr);
    y = mod([a b]-1,dimr)+1;
    XX = x(1);
    YY = y(1);
    steep = (abs(y(2)-y(1)) > abs(x(2)-x(1)));
    if steep
        t = x; x = y; y = t;
    end
    if x(1) > x(2)
        t = x(1); x(1) = x(2); x(2) = t;
        t = y(1); y(1) = y(2); y(2) = t;
    end
    delx = x(2)-x(1);
    dely = abs(y(2)-y(1));
    err = 0;
    x_n = x(1);
    y_n = y(1);
    if y(1) < y(2), ystep = 1; else ystep = -1; end
    myline = zeros(1,delx+1);
    for nn = 1:delx+1
        if steep
            myline(nn) = 1+dimr*(y_n-1)+(x_n-1);
        else
            myline(nn) = 1+dimr*(x_n-1)+(y_n-1);
        end
        x_n = x_n + 1;
        err = err + dely;
        if bitshift(err,1) >= delx, % same as -> if 2*err >= delx,
            y_n = y_n + ystep;
            err = err - delx;
        end
    end
    if myline(1) ~= (XX-1)*dimr+YY
        myline = myline(end:-1:1);
    end
end
% End of pathmaster
end
```

APPENDIX D: SUPPLEMENTARY INFORMATION FOR THE EXPLORATION LAB FIELD TEST

2.00AJ/16.00AJ Lab 1: Exploration on the Moon (well, Killian Court): Mission Planning for EVA and Geology

Background:

Several days ago, a rover sent through the Killian terrain identified various regions of distinct geological formations. Within each region, the rover mapped the locations of several sites where highly interesting geological samples may be collected. In response to this exciting discovery, a team of astronauts and rovers nearby on the surface has been re-directed to the Killian region in order to examine and bring back these samples. Nearing the end of their scheduled surface mission duration, the team has enough resources remaining for roughly 8 hours of work toward exploring Killian before they must return to the lunar module. In response to this change of plans, Mission Control must now develop a strategy to maximize the scientific return from Killian before returning the surface team home safely.

Mission Detail:

As shown in the map on the next page, Killian has been segmented into three distinct zones. Within each zone are the marked locations of sites of potential geological interest. Although various samples are expected to be encountered at each site, geological data will provide the identity of "samples of interest" that are to be collected by the astronauts. Different zones may have different "samples of interest", and not every site is necessarily "interesting".

Objectives:

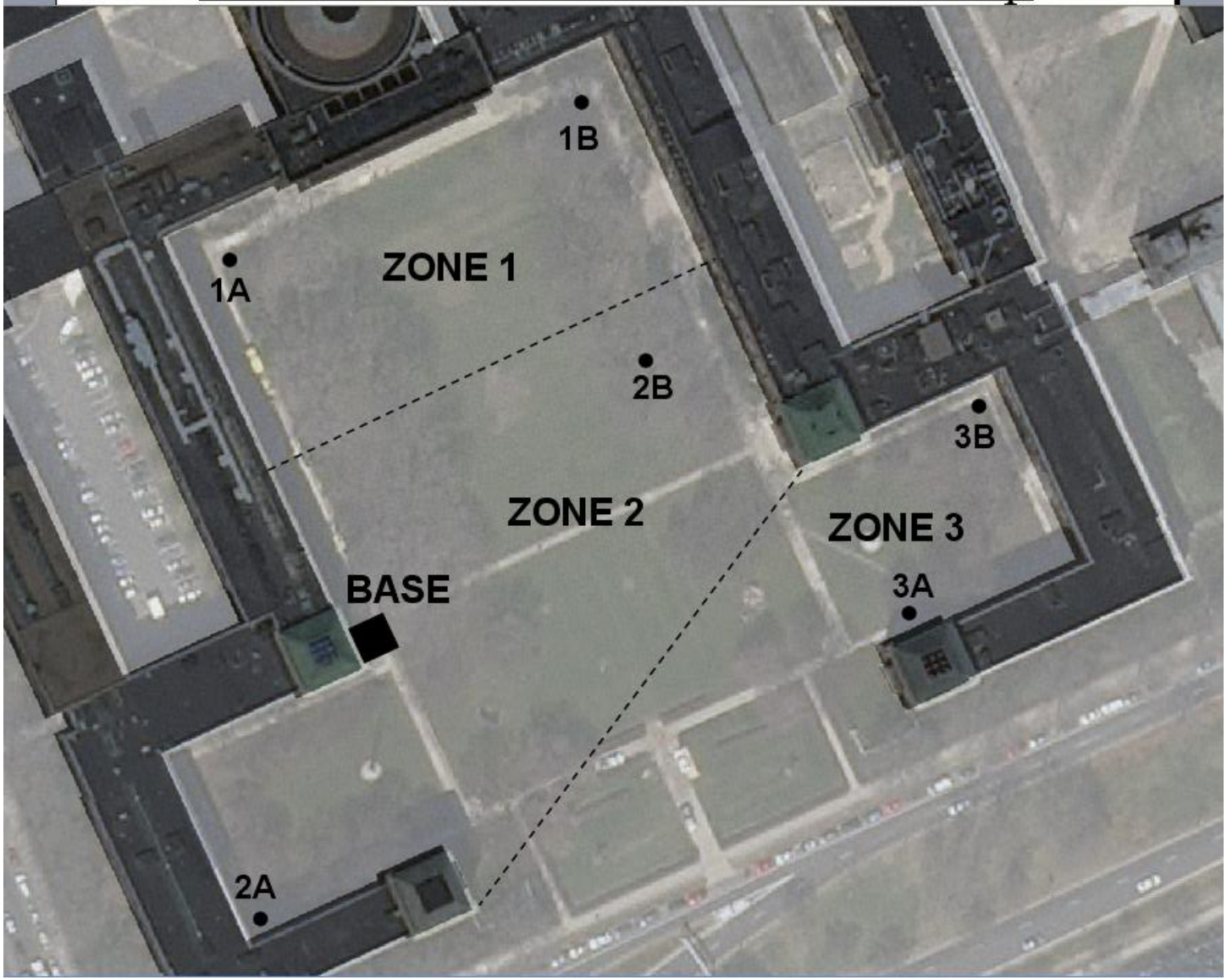
The suited astronauts and rovers will explore the Killian terrain beginning at the starting base. The mission objectives, listed in order of priority, are as follows:

1) Safely return all astronauts and rovers to the base
2) Collect a sample of interest from as many zones as possible
3) Collect as many samples of interest as possible

Schedule:

2:00 Introduction & choosing team positions
2:20 Mission planning
3:00 Team Shackleton EVA1
3:20 Discussion, team switch, real-time planning for EVA2
3:35 Team EARLE EVA2
3:55 Debrief, demos, cleanup

Killian Terrain: Marked Aerial Map



Resources and Limitations:

General

- The EVA mission will be run on a 1/30 time scale, so every minute in real-time is weighted as a half-hour.
- The Exploration Surface Team will have a continuous audio link with Mission Control via the Communicator; however, the Surface Team will have no access to maps or data and must rely on Mission Control to guide them. Likewise, Mission Control will not be able to see the Surface Team and must rely upon the communications link.

Astronauts

- The astronauts have enough oxygen for about 9 hours of *light* activity. To conserve oxygen, astronauts should attempt to remain relatively still and should lope[1] or walk and may not run while exploring. Carrying a load also increases oxygen consumption.
  Monitored by the Medical Officer
- The astronauts are limited to a cumulative traverse distance of 1,000 meters.
  Monitored by the Positioning Officer
- The astronauts must stick together and travel as a group.
- Each astronaut may carry one sample (or rover) at a time.

Rovers

- The rovers have battery power for an expected 5-10 hours of use. This is highly dependent upon the rover and the level of activity.
  Monitored by the Rover Technician and rover operators
- The rovers may travel independently
- Rovers cannot carry samples

[1] Loping (a form of run with increased ariel phase) is more energy efficient than walking, per unit distance and mass, in environments with gravity reduced more than 50% relative to Earth (<0.5G).
Newman, D.J., Alexander, H.L. and B.W. Webbon, "Energetics and Mechanics for Partial Gravity Locomotion," *Aviat Space and Environ Med*, 65: 815-823, 1994; C.E. Carr and D.J. Newman. "When is running in a space suit more efficient than walking in a space suit?", Society of Automotive Engineers, Inc., Warrendale, Pennsylvania, USA. SAE paper 2005-01-2970, 2005; Carr, C. E., Newman, D. J., "Space Suit Bioenergetics: Cost of Transport During Walking and Running", *Journal of Aviation, Space Environmental Medicine*, 78:1093-1102, 2007.

Team Format:

Lunar Exploration Surface Team

Astronauts (2-4) The astronauts will be physically moving ("loping" is the preferred means of locomotion by lunar astronauts) about Killian, wearing Apollo "space suits". Astronauts must stick together. The lunar astronauts will communicate with Mission Control via walkie-talkie. Each astronaut will be able to carry one "sample" at a time.

Rovers (3) The lunar rovers will be three RC robots, and are part of the exploration team in Killian. Unlike astronauts, rovers do not need to stick together and can move wherever commanded. The rovers may be equipped with wireless cameras fed back to Mission Control. RC rover motion will be controlled by human operators out in the field who will receive commands from Mission Control. Operators (considered part of Mission Control) should not physically assist the rovers; however, an astronaut can move or carry a rover should it get stuck or run out of battery power. Rovers cannot carry "samples." *See rover detail, page 5*

Lunar Exploration Mission Control Team

Director The Director oversees the mission and makes final decisions regarding how to proceed. All other Mission Control positions report to the Director.

Communicator(s) The Communicator is an astronaut, and the only person who may communicate with the exploration astronauts via walkie-talkie. The Communicator is also responsible for sending rover commands.

Positioning (1-2) The Positioning Officer(s) will update and display current astronaut and rover positioning on a real-time map display. The Positioning Officer will also need to keep track of distance traveled, inform the Director of astronaut constraints (i.e. distance to return 'home', etc.), and report if/when the exploration activity needs to end. *See Positioning Officer detail, page 6*

Medical The Medical Officer will update and monitor the astronaut heart rate and oxygen levels and detect any problems. They will inform the Director of the astronaut status (heart rate & oxygen remaining) and if/when the exploration activity must end based upon data. *See Medical Officer detail, page 7*

Rover Tech. (1-2) The Rover Technician(s) will monitor the rover video feed and track battery life. Rover positioning commands will need to be given to the Communicator, and visual data provided to the Geologist. They will inform the Director of rover battery status and if/when the activities must end based upon data. *See Rover Technician detail, page 8*

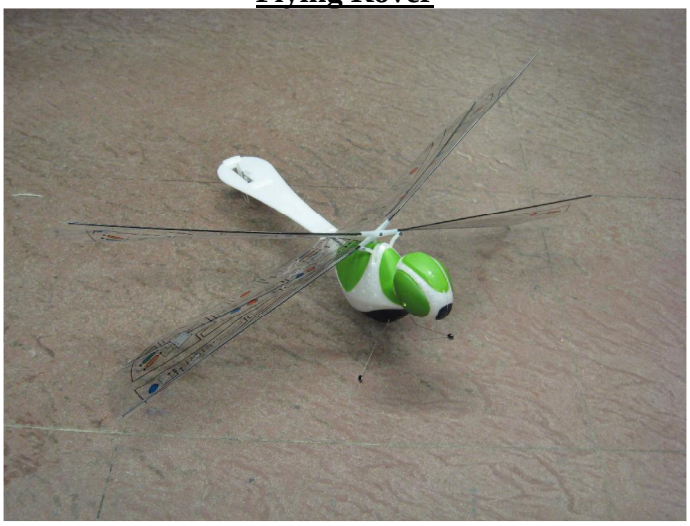
Geologist The Geologist in Mission Control will be provided with data regarding samples of interest to be collected by the field astronauts. Based upon this data, the Geologist will advise the Director as to which sites are most valuable for scientific return.

Rover Detail

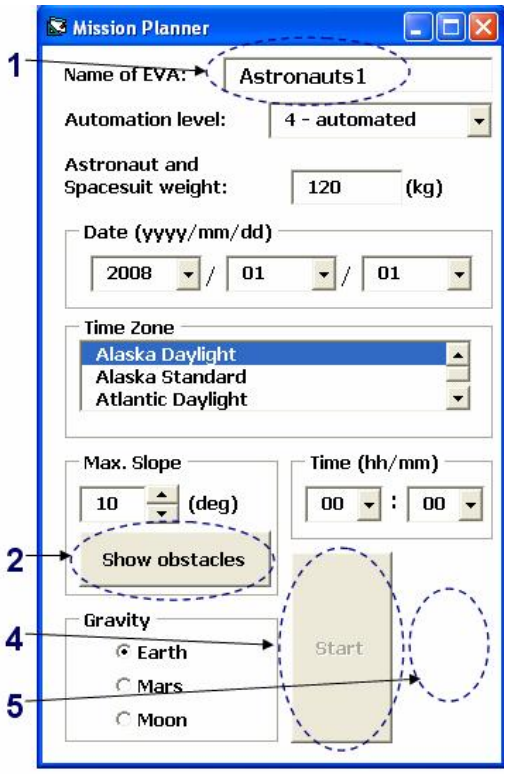
Rolling Rover

Crawling Rover

Flying Rover



Positioning Officer Detail



1) Identify the traverse path segment

2) Click "Show obstacles"

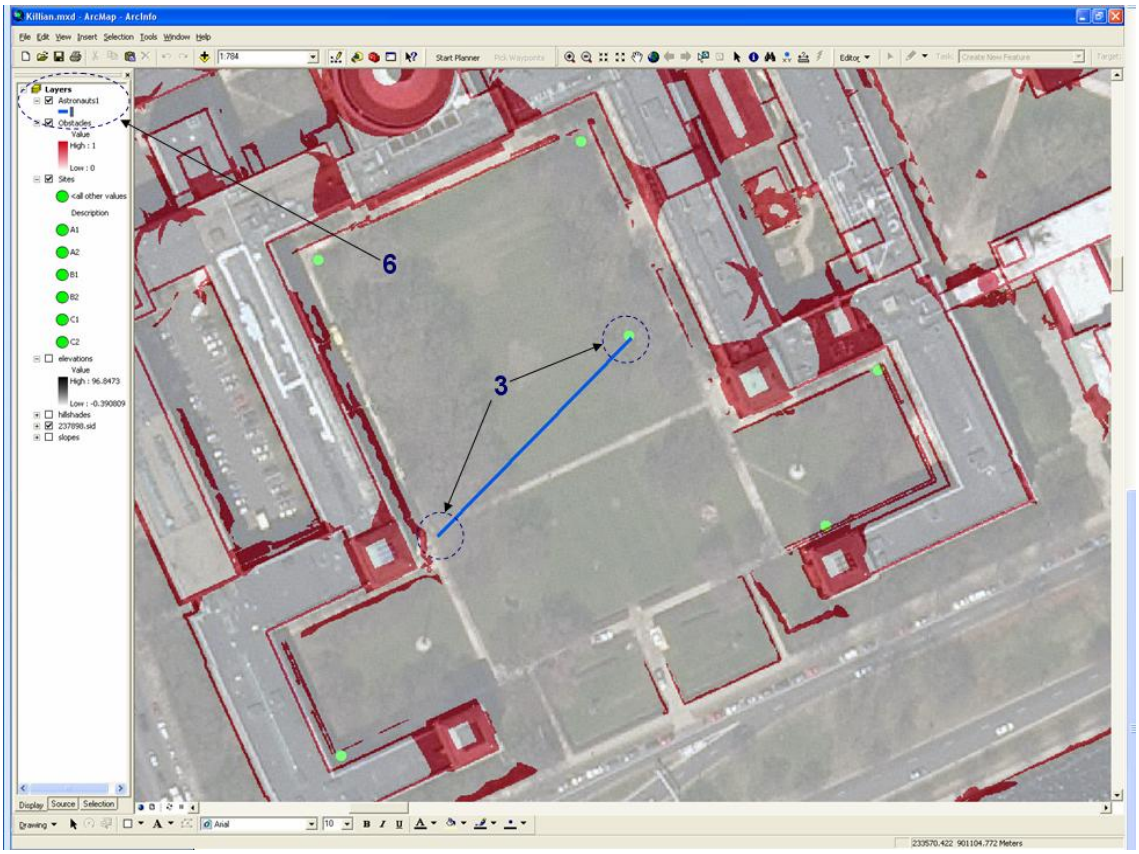
3) Click on start then end point on map

4) Click "Start"

5) Record traverse distance data

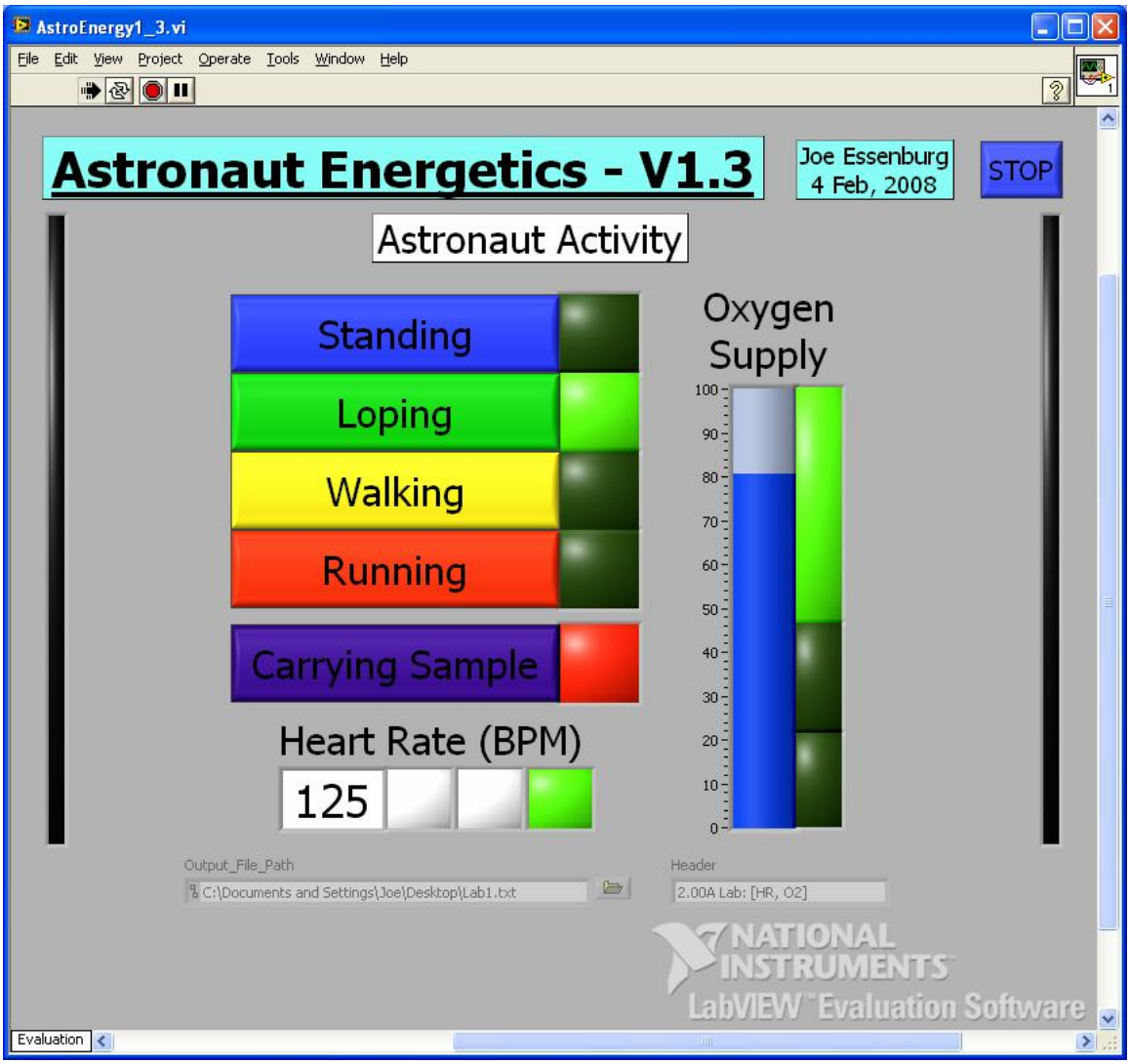
6) Edit path segment appearance on map

7) Repeat from (1)



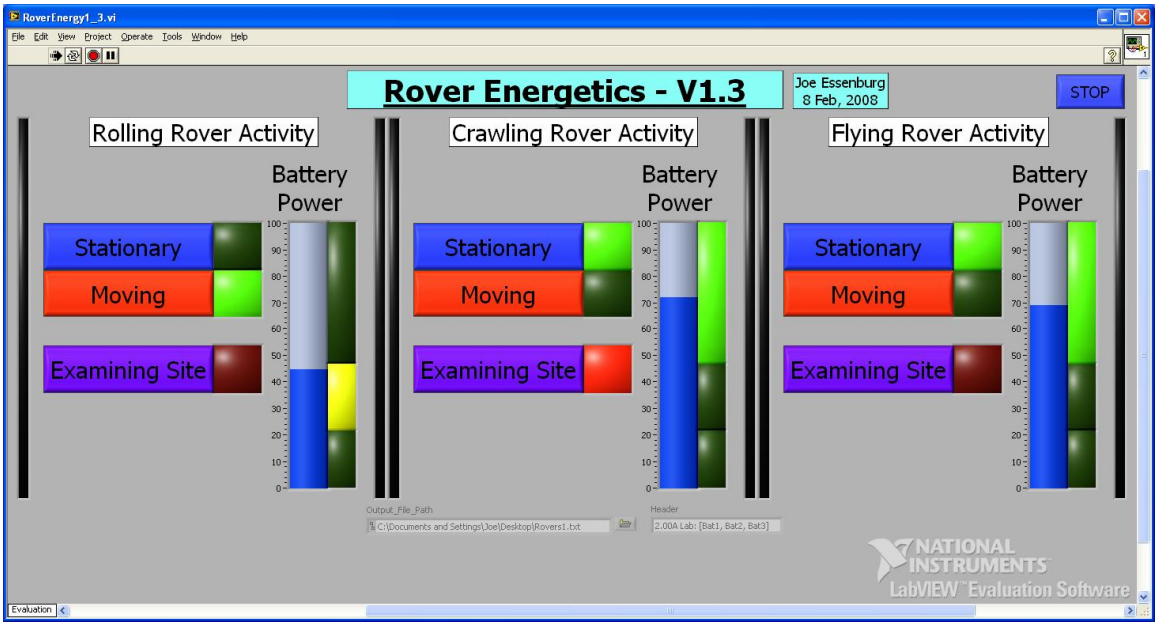
Medical Officer Detail

- Point and click to record current astronaut activity
- Monitor astronaut heart rate and Oxygen supply
- Notify team of astronaut status and/or problems



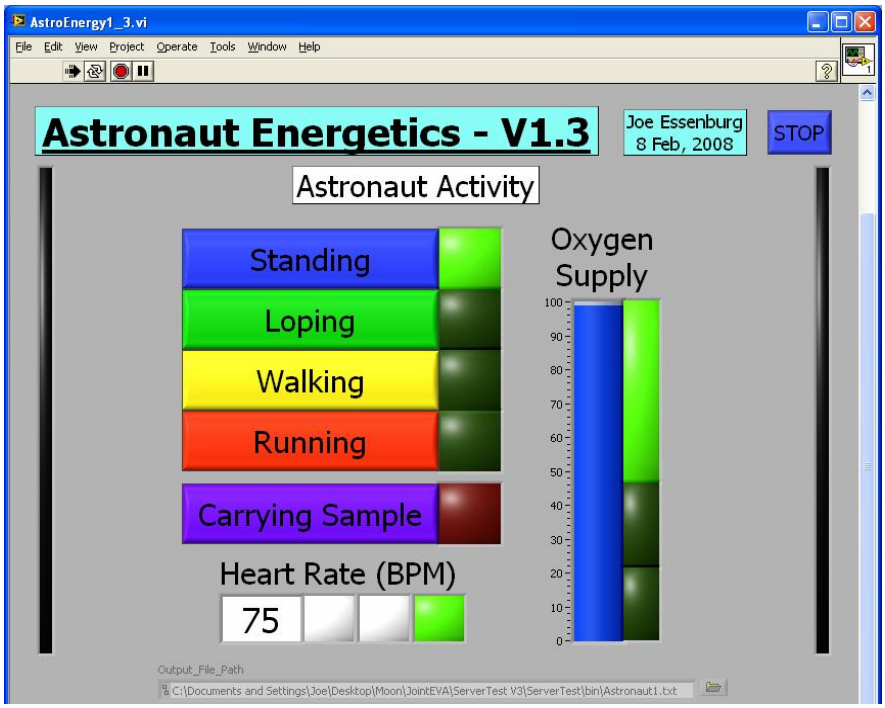
Rover Technician Detail

- Monitor rover video feed
- Inform Geologist of data from sites
- Point and click to record current rover activities
- Monitor rovers' battery power
- Notify team of rover status and/or problems

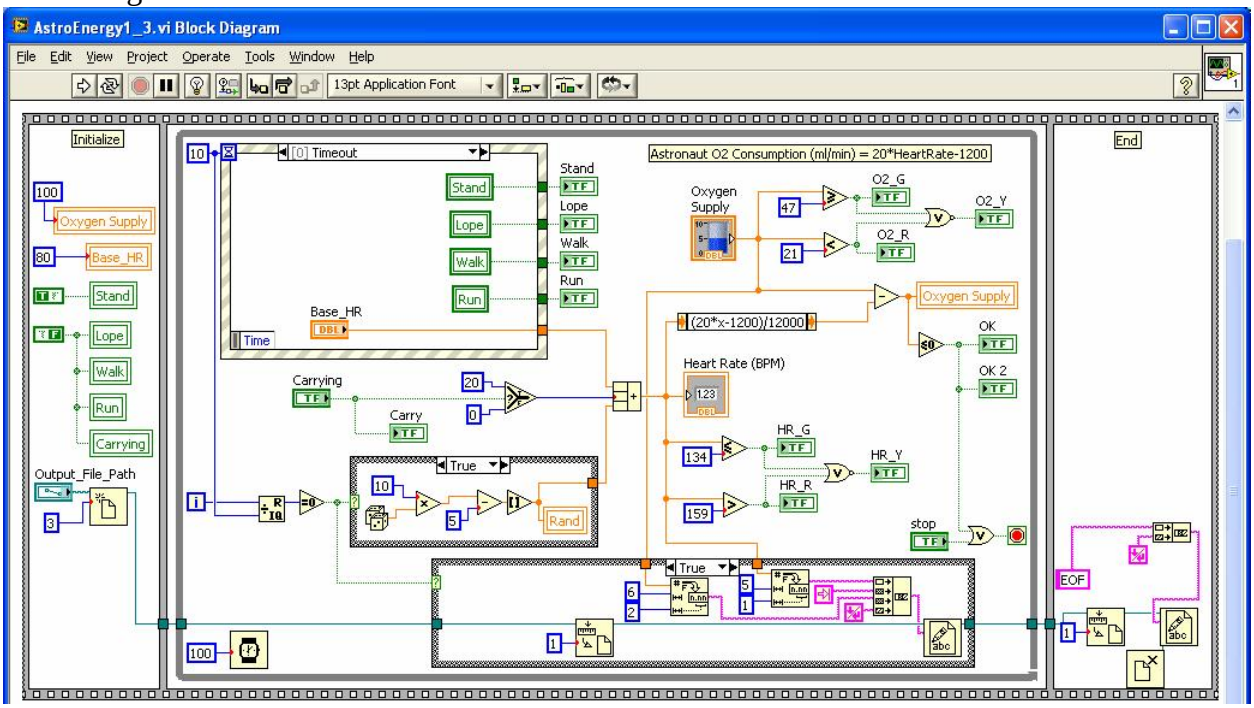


APPENDIX E: LABVIEW ENERGETICS MODELS

Astronaut energetics model for the Exploration Lab field test:
Front panel:

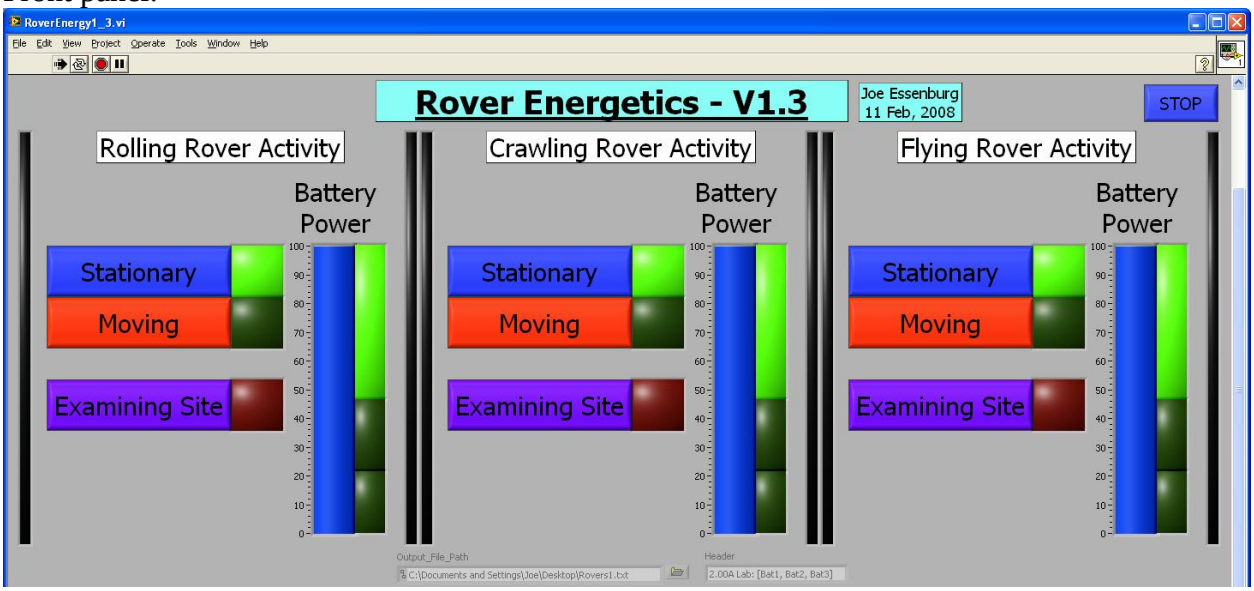


Block diagram:

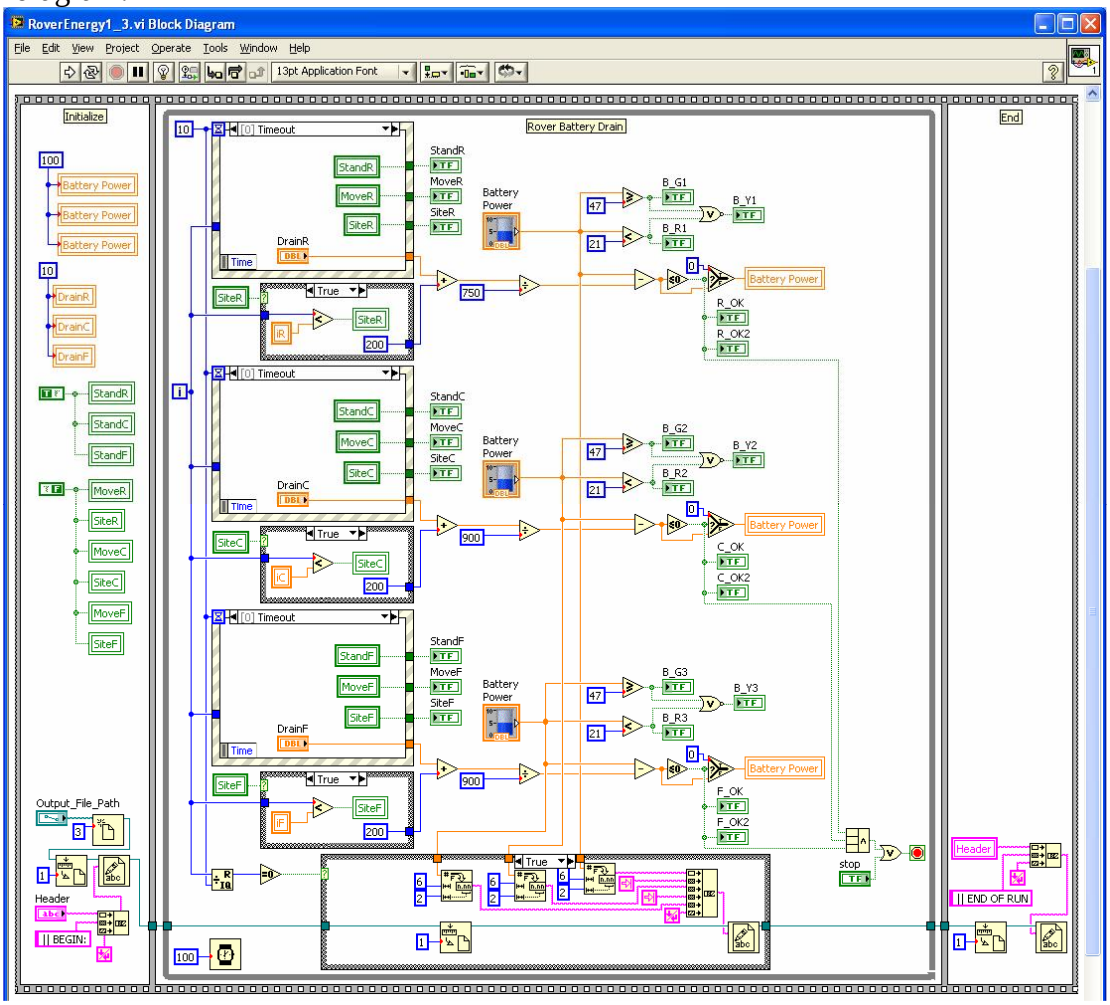


Robot energetics model for the Exploration Lab field test:

Front panel:

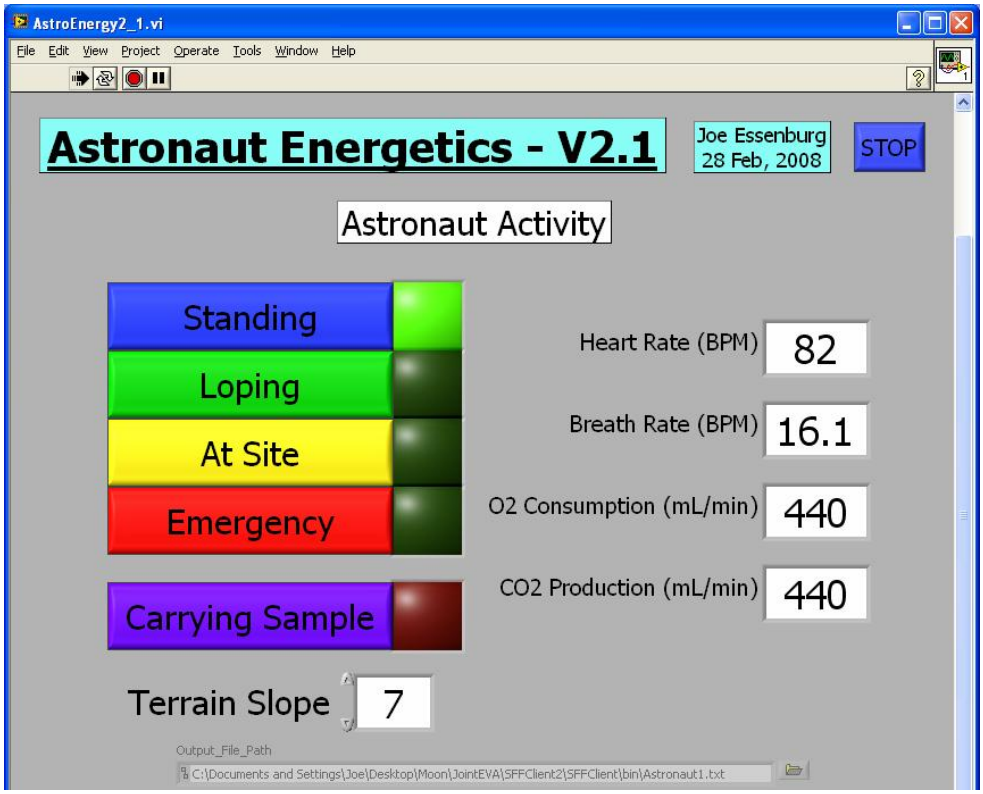


Block diagram:

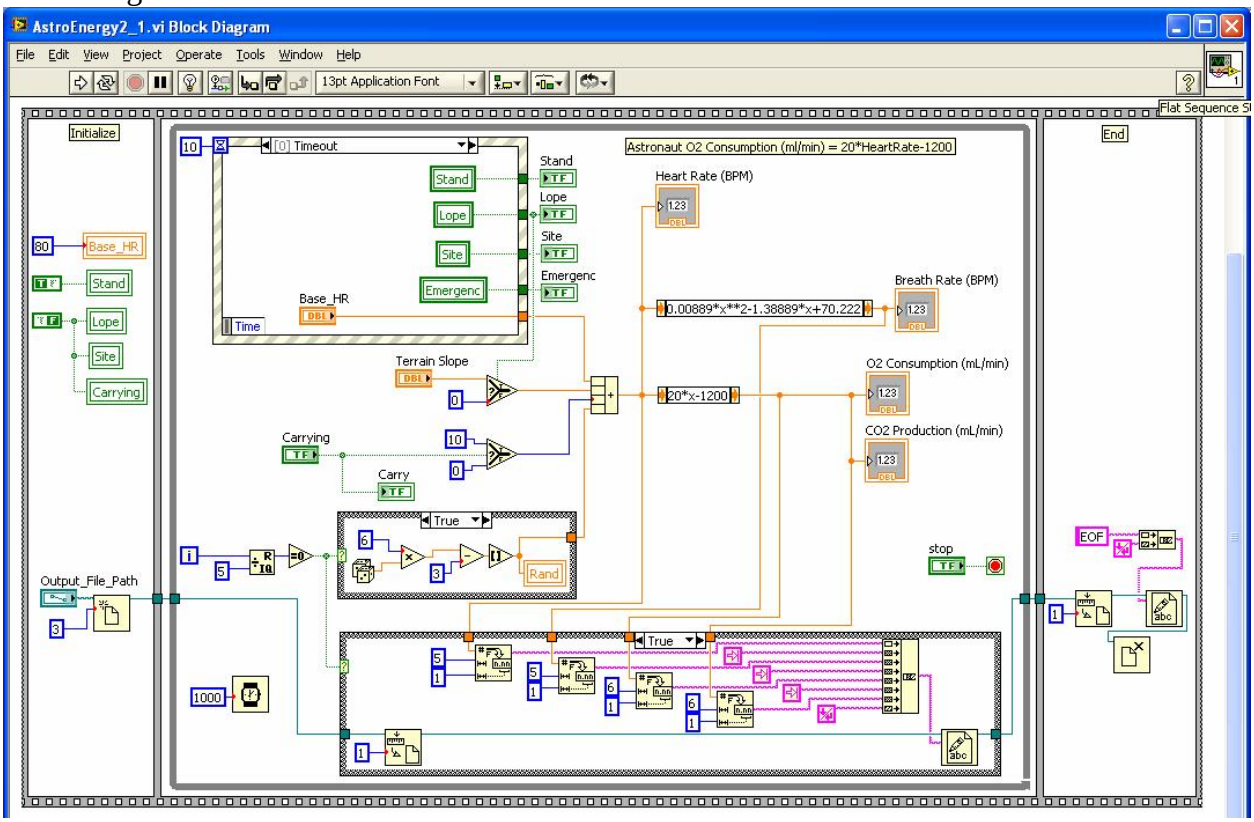


Astronaut energetics model for the joint EVA simulations:

Front panel:



Block diagram:



REFERENCES

Acuña, M. H. (2003). The magnetic field of Mars. *The Leading Edge,* August 2003, 769 – 771.

Arnett, B. (2005). The Moon. *The Nine Planets*. Retrieved June 23, 2008, from the World Wide Web: http://www.nineplanets.org/luna.html

Asaravala, A. (2004). A Black Box for Human Health. *Wired*. Retrieved July 28, 2008, from the World Wide Web: http://www.wired.com/science/discoveries/news/2004/04/63034

Bagherzadeh, N., Chou, P. H., Kurdahi, F., & Liu, J. (2001). Power-Aware Scheduling Under Timing Constraints for Mission-Critical Embedded Systems. *38th Conference on Design Automation,* 840 – 845.

BBC. (2008). *1969: Man takes first steps on the Moon*. BBC On This Day. Retrieved June 17, 2008, from the World Wide Web: http://news.bbc.co.uk/onthisday/hi/dates/stories/july/21/newsid_2635000/2635845.stm

Biesiadecki, J. J., Leger, P. C., & Maimone, M. W. (2005). *Tradeoffs between directed and autonomous driving on the Mars Exploration Rovers.* Paper presented at the International Symposium of Robotics Research, San Francisco, CA.

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal,* 4(1), 25 – 30.

Bush, G. W. (2004). "The Vision for Space Exploration." Speech given to NASA Headquarters, Washington, D.C.

Cabrol, N. A., Kosmo, J. J., Trevino, R. C., Thomas, H., Eppler, D., Bualat, M. G., Baker, K., Huber, E., Sierhuis, M., Grin, E. A., Stoker, C. R., Schreiner, J. A., Sims, M. H., Gulick, V. C., & Cockell, C. S. (1999). Results of the First Astroanut-Rover (ASRO) Field Experiment: Lessons and Directions for the Human Exploration of Mars. *Proceedings of the 18th Digital Avionics Systems Conference,* 2, 7.C.3-1 – 7.C.3-8

Carney, M. D., Willis, B., Marchant, C., Miskin, J., Andersen, B., Schicker, J., & Jolley, P., (2005). *AEGIS: A Navigation and Communications Infrastructure for the Moon and Mars.* College of Engineering, Utah State University.

Carr, C. E., Newman, D. J., & Hodges, K. V. (2003). *Geologic Traverse Planning for Planetary EVA.* Paper presented to the 33rd International Conference on Environmental Systems, Vancouver, B.C., Canada.

Casper, J. & Murphy, R. R. (2003). Human-Robot Interactions During the Robot-Assisted Urban Search and Rescue Response at the World Trade Center. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 33(3), 367 – 385.

Christy, R. (2008). Luna17 – Carrier for Lunokhod 1. *Zarya: Soviet, Russian and International Spaceflight*. Retrieved June 23, 2008, from the World Wide Web: http://www.zarya.info/Diaries/Luna/Luna17.php

Clement, J. L., Boyd, J. E., Kanas, N., Saylor, S. (2007). Leadership challenges in ISS operations: Lessons learned from junior and senior mission control personnel. *Acta Astronautica,* 61(1-6), 2 – 7.

Cooke, D., Yoder, G., Leshin, L., & Gernhardt, M. (2007). *Exploration Systems Mission Directorate Lunar Architecture Update*. Presentation at the AIAA Space 2007 Conference & Exposition, Long Beach, CA.

Cummings, M. L. (2006). *Human Interaction with Automated Planners*. National Science Foundation Proposal.

Engle, M. (2004). *Operational Considerations for Manned Lunar Landing Missions – Lessons Learned from Apollo*. Paper Presented at the AIAA Space 2004 Conference and Exhibit, San Diego, CA.

Eppler, D. (2004). *Conduct of Geologic Field Work During Planetary Exploration: Implications for EVA and Robotic Systems Interaction*. Presentation to SAE.

GM Working on 'Smart' Windshields to Guide Drivers. *Fox News Press Release.* Retrieved July 18, 2008, from the World Wide Web: http://www.foxnews.com/story/0,2933,385302,00.html

Gorder, P. F. (2008). New Project to Develop GPS-Like System for the Moon. *Ohio State University Press Release*. Retrieved July 28, 2008, from the World Wide Web: http://www.moontoday.net/news/viewpr.html?pid=26025

Hart, P. E., Nilsson, N. J., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems, Science, and Cybernetics SSC4(2),* 100 – 107.

Iagnemma, K., Kang, S., Shibly, H., & Dubowsky, S. (2004). Online Terrain Parameter Estimation for Wheeled Mobile Robots With Application to Planetary Rovers. *IEEE Transactions on Robotics,* 20(5), 921 - 927.

Johnson, B. J. (2008). *Optimization of a Planning Support System for Planetary Exploration Extravehicular Activities*. Paper presented to the Massachusetts Institute of Technology Summer Research Program.

Jones, E. M. (1995). *Apollo Lunar Surface Journal*. Retrieved June 21, 2008, from the World Wide Web: http://www.hq.nasa.gov/alsj/

Jones, E. M. (2006). *Apollo 16 Lunar Surface Journal.* Retrieved June 21, 2008, from the World Wide Web: http://history.nasa.gov/alsj/a16/a16.lsp.html

LaPiana, F. (1971). *A Summary of the LRV Navigation System*. (NASA-CR-121354). Washington, D.C.

Leger, P. C., Deen, R. G., & Bonitz, R. G. (2005). *Remote Image Analysis for Mars Exploration Rover Mobility and Manipulation Operations*. Paper presented at the Systems, Man, and Cybernetics 2005 IEEE Conference, Big Island, HI.

Lindqvist, L. V. J. (2008). *Multidisciplinary Extravehicular Activity Mission Optimization for Lunar Exploration*. Master Thesis, Technishe Universität München.

Márquez, J. J. (2007). *Human-Automation Collaboration: Decision Support for Lunar and Planetary Exploration*. Ph.D. Thesis, Massachusetts Institute of Technology.

Muehlberger, W. R. (1981). Apollo 16 Traverse Planning and Field Procedures. In G.E. Ulrich & C. A. Hodges & W.R. Muehlberger (Eds.), *Geological Survey Professional Paper 1048: Geology of the Apollo 16 Area, Central Lunar Highlands* (pp. 10 - 20). Washington, D.C., US Government Printing Office.

NASA. (1971). *Apollo 14 Mission Report (MSC-04112).* Houston, TX: Manned Spacecraft Center.

NASA. (2004). *Apollo 11 at 35: Celebrating the Past with a Vision for the Future*. NASA. Retrieved June 15, 2008, from the World Wide Web: http://www.nasa.gov/vision/space/features/apollo11_35th.html

NASA. (2007). *Exploration: NASA's Plan to Explore the Moon, Mars and Beyond*. NASA. Retrieved July 19, 2008, from the World Wide Web: http://www.nasa.gov/mission_pages/exploration/mmb/why_moon.html

Ney, Z. A., & Looper, C.A. (2005). *Integrated Planning and Logistics of Exploration Initiatives for Long Term Maintenance EVA/EVR Operations*. Paper presented at the AIAA 1st Space Exploration Conference, Orlando, FL.

Norris, J. S., Powell, M. W., Vona, M. A., Backes, P. G., & Wick, J. V. (2005). *Mars Exploration Rover Operations with the Science Activity Planner*. Paper presented at the IEEE International Conference on Robotics and Automation.

O'Keefe, K., Lachapelle, G., Skone, S. (2004). GPS goes martian: nav/com for a red planet. *GPS World.* Retrieved July 20, 2008, from the World Wide Web: http://findarticles.com/p/articles/mi_m0BPW/is_6_15/ai_n6175511/pg_1?tag=artBody;col1

Parasuraman, R., Sheridan, T. B., & Wickens, C. D. (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans,* 30, 286 - 297

Perko, H. A., Nelson, J. D., Green, J. R. (2006). Mars Soil Mechanical Properties and Sustainability of Mars Soil Simulants. *Journal of Aerospace Engineering*, 19(3), 169-176.

Rader, A. A., Newman, D. J., Carr, C. E. (2007). *Loping: A Strategy for Reduced Gravity Human Locomotion?* Paper presented at the International Conference On Environmental Systems, Chicago, IL.

Riesterer, J. (2008). UTM – Universal Transverse Mercator Geographic Coordinate System. *Geospatial Training and Analysis Cooperative Introduction to Topographic Maps*. Retrieved July 19, 2008, from the World Wide Web: http://geology.isu.edu/geostac/Field_Exercise/topomaps/utm.htm

Santee, W. R., Allison, W. F., Blanchard, L. A., & Small, M. G. (2001). A proposed model for load carriage on sloped terrain. *Aviation, Space, and Environmental Medicine,* 72(6).

Sarter, N., & Schroeder, B. (2001). Supporting decision making and action selection under time pressure and uncertainty: The case of in-flight icing. *Human Factors*, 43(4), 573 – 583.

Squires, S. (2008). "NASA's Plans To Return To The Moon In Preparation For A Manned Flight To Mars." Interview with 60 Minutes, CBS.

Titterton, D. H., & Weston, J. L. (2004) *Strapdown Inertial Navigation Technology*. New York: American Institute of Aeronautics & Astronautics.

UNBC GIS & Remote Sensing Lab. (2008). *Geo-Referencing*. Lecture notes for Introduction to Geographic Information System. Retrieved July 20, 2008, from the World Wide Web: http://www.gis.unbc.ca/courses/geog300/lectures/lect3/index.php

Wade, M. (2008). Apollo LRV. *Encyclopedia Astronautica*. Retrieved June 20, 2008, from the World Wide Web: http://www.astronautix.com/craft/apololrv.htm