FootFall: A Ground Based Operations Toolset Enabling Walking for the ATHLETE Rover

Vytas SunSpiral[*]
NASA Ames Research Center, Moffett Field, CA, 94035, USA

Daniel Chavez-Clemente[†]
Stanford University, Stanford, CA, 94305, USA

Michael Broxton[‡], Leslie Keely[§]
NASA Ames Research Center, Moffett Field, CA, 94035, USA

Patrick Mihelich[**]
Willow Garage, Inc, Menlo Park, CA, 94025, USA

David Mittman[††], Curtis Collins[‡‡]
NASA Jet Propulsion Laboratory/California Institute of Technology, Pasadena, CA, 91109, USA

The ATHLETE (All-Terrain Hex-Limbed Extra-Terrestrial Explorer) vehicle consists of six identical, six degree of freedom limbs. FootFall is a ground tool for ATHLETE intended to provide an operator with integrated situational awareness, terrain reconstruction, stability and safety analysis, motion planning, and decision support capabilities to enable the efficient generation of flight software command sequences for walking. FootFall has been under development at NASA Ames for the last year, and having accomplished the initial integration, it is being used to generate command sequences for single footfalls. In this paper, the architecture of FootFall in its current state will be presented, results from the recent Human Robotic Systems Project's Integrated Field Test (Moses Lake, Washington, June, 2008) will be discussed, and future plans for extending the capabilities of FootFall to enable ATHLETE to walk across a boulder field in real time will be described.

I. Introduction

ATHLETE (All-Terrain Hex-Limbed Extra-Terrestrial Explorer) is a large (3 meters in diameter) six-legged robot developed at the Jet Propulsion Laboratory (Fig. 1). ATHLETE is a flexible platform designed to serve multiple roles during manned and unmanned missions to the moon, including transportation, construction and exploration. ATHLETE is to be operated by personnel on Earth or by astronauts on the Moon either located inside a habitat shirtsleeve environment or outside the habitat in spacesuits.

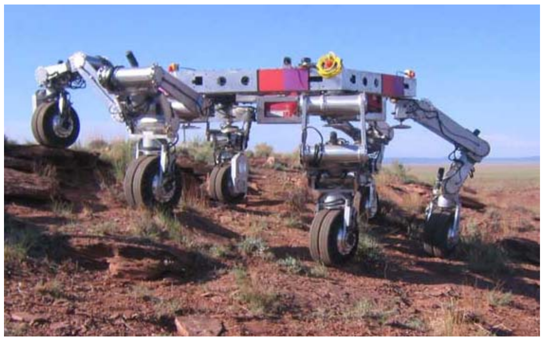


Figure 1. ATHLETE walking on rough terrain

In the two years since prototype third-scale ATHLETE robots became operational, a wide array of capabilities have been demonstrated.[1,2] ATHLETE can roll on smooth terrain,

[*] Robotics Researcher, Carnegie Mellon University, NASA Ames, M/S 269-3, Vytas.Sunspiral@nasa.gov
[†] PhD Candidate, Aerospace Robotics Laboratory, Durand Building Rm 017, dchavez@stanford.edu
[‡] Robotics Researcher, Carnegie Mellon University, NASA Ames, M/S 269-3, Michael.Broxton@nasa.gov
[§] Computer Engineer, Code TI, NASA Ames, M/S 269-3, Leslie.Keely@nasa.gov
[**] Software Engineer, 68 Willow Road, mihelich@willowgarage.com
[††] Senior Technical Staff, Planning Software Systems, Mail Stop 301-250D, David.S.Mittman@jpl.nasa.gov
[‡‡] Member of Technical Staff, Mobility and Robotic Systems, Mail Stop 82-105, Curtis.L.Collins@jpl.nasa.gov

combine walking with rolling to traverse uneven terrain and even climb ledges. It can manipulate tools, rappel down a steep slope, and coordinate with other robots. All of these activities involve sequences of commands selected by human operators with limited software aid. Rolling can be commanded efficiently because a single command can direct the robot to travel long distances. If commanded, active compliance can be enforced when rolling; this means the robot will keep its chassis level and wheels coordinated while rolling over uneven ground. However, when stepping is involved, active compliance of the robot's posture is not currently available and low level joint space and task space motion commands must be sent, making operation tedious.

The goal of this project is to make ATHLETE operation safer, faster, and more efficient by developing a decision support ground tool (the FootFall software) capable of generating sequences of commands for stepping and walking, along with any required on-board software. The multi-year goal is to enable ATHLETE to traverse complex boulder fields, steep slopes, and ledges. FootFall takes telemetry from the robot, such as joint angles and stereo camera image pairs, and generates 3D terrain maps colored by stability and reachability metrics to provide walking-specific situational awareness to the operator. The operator may inspect a model of the robot within the reconstructed terrain using Viz, a fully integrated 3D terrain and robot modeling system. The current FootFall system is designed to plan leg motions for a single leg step at a time, with extensions to multi-step planning under development. Once the operator chooses the next location to move a leg, a motion planner computes a collision free path to the location that respects kinematics and safety constraints. The output is a sequence of flight software commands that can be visualized and modified before being sent to the robot.

In the rest of this section we will give an overview of the ATHLETE robot itself, and review other walking robots. Section II will cover the architecture and details of the FootFall system. Section III will discuss the results from NASA's recent Human and Robotic Systems Integrated Robotics Field Test held at Moses Lake, Washington, June 2008, and will include a discussion of open issues and plans for future work. Finally, we will provide some high level insights gained from this effort in the conclusion (Section IV).

A. Overview of the ATHLETE Robot

ATHLETE is a technology development project managed by NASA's Jet Propulsion Laboratory in California (JPL). ATHLETE is capable of rolling over relatively flat terrain and "walking" over extremely rough or steep terrain. The current ATHLETE vehicle is able to travel at speeds of 3 km/h, climb vertical steps of 1.7 m, and carry payloads of up to 300 kg. Additionally, ATHLETE can fold into a flat ring, allowing for easier transport to the lunar surface.

The ATHLETE vehicle, shown in Figures 1 and 2, consists of six identical six-degree-of-freedom limbs. Attached to the end of each limb is a wheel that can be used for mobility in the form of driving over benign terrain. Alternatively, the wheels can be locked rotationally so that the limbs can be used for walking over rough terrain. The rover body is shaped as a hexagon, giving six flat faces that can be used to dock to similar ATHLETE vehicles, or to other systems such as refueling stations, rappelling winches, etc. One unique advantage of the wheel-on-leg ATHLETE concept is that it combines the high mobility of legged vehicles with the energy efficiency of wheeled vehicles.

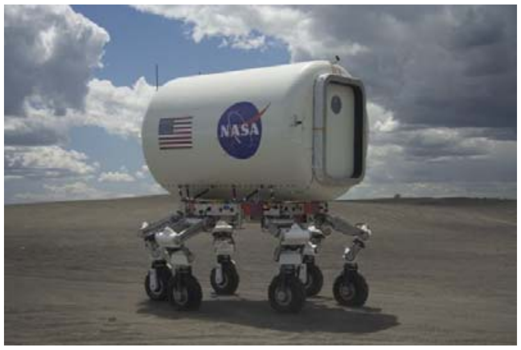


Figure 2. Athlete at the Moses Lake Field Test, carrying a mock habitat

For sensing its environment, ATHLETE has a number of stereo camera pairs. Each face of the hexagon has a pair (NavCams) that provides the long distance views required for driving. There are three more pairs (HazCams) mounted on the inner vertices of the hex that give visibility to the area below the robot and between the legs. Finally, some of the legs have cameras mounted above the wheels (ToolCams) that are designed to assist with the deployment and use of tools. For the purposes of walking, we mainly rely upon the HazCams because the NavCams do not see the ground within reach of the legs. Other sensors on the vehicle include an IMU for pose recovery, and dual absolute and relative encoders on all the joints, which enable the calculation of torques.[3]

B. Related Work

The control and planning for legged walking robots is an extremely active field where research can be generally grouped into two categories: dynamically controlled robots, and statically stable robots. A large body of the current research efforts are focused on dynamically controlled robots,[4,5] where the center of gravity (CG) is allowed to move outside of the polygon of ground contacts. This approach, which uses closed loop controllers to dynamically react to experienced forces and positions, has lead to the success of well-known robots such as BigDog,[6] which is capable of recovering its balance after slipping on ice. Implementing such a controller requires modeling the dynamics of individual motors and power systems,[7] and having an actuation system with a power to weight ratio capable of generating the instantaneous forces necessary to bring the CG back into the support polygon. Many of these systems rely upon high-pressure hydraulic actuators, which are unlikely be used in a vacuum due to the potential for leaks in the pressure system. ATHLETE uses highly geared energy efficient motors, and as a result cannot produce the instantaneous power outputs required for dynamic control approaches. Instead, ATHLETE is a statically stable robot, where the CG is always maintained within the support polygon.

Many statically stable robots have been built,[8] as their design and control is better understood than dynamically stable robots. Of course, most of these robots have not left the labs, or at least highly controlled environments. Of the few that have been deployed into natural unstructured terrains, possibly the most famous is Dante II,[9] which descended into the mouth of an active volcano. The design of these robots has generally been optimized to make the task of walking as simple as possible. Dante II, for example, is a frame walker, which means that it has two sets of rigidly connected 4-leg frames that translate with respect to each other. Thus planning was greatly simplified as the problem was highly constrained. Even amongst walkers with independently articulated legs, most robots employ legs with only three or four joints each, which also simplifies the task of planning a single leg step because there is a fairly direct mapping between the task space and the control of the joint parameters.

ATHLETE is unique among this class of multi-legged statically stable walkers due to the complexity of its legs, which have six degrees of freedom (DOF) each. While most walkers have the luxury of optimizing their mechanical design to facilitate walking, ATHLETE is constrained by many complex factors such as driving, tool use, launch vehicle considerations, load carrying, docking, and maintaining a horizontal load platform. These considerations lead to the general-purpose six-DOF legs used on ATHLETE, which in turn makes the act of planning a single step more complicated.

II. FootFall

A. Architectural Overview

The system architecture of FootFall, which is depicted in Figure 3, consists of telemetry integration and processing modules that feed an interactive 3D user interface to enable improved operator situational awareness. The camera data is sent through the Ames Stereo Pipeline for terrain reconstruction, and the pose and orientation telemetry is combined with the resulting terrain morphology and a mass model of the rover to establish the current stability of the rover. This is accomplished by calculating the center of gravity of the robot in its current configuration and the Conservative Polygon of Support, which ensures that the robot will stay stable even in the event of the total failure of any single leg. All this information, along with experienced forces and torques, are displayed in Viz, a three dimensional visualization environment developed at Ames which enables the operator to effortlessly understand the current situation of the rover, and to then select locations on the terrain for the next foot placement.

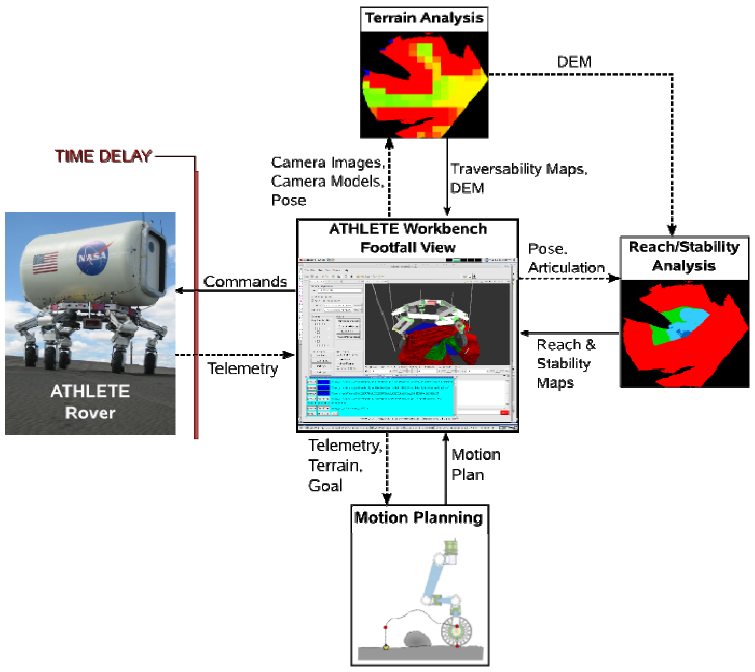


Figure 3. Overall architecture of FootFall Software

Once a desired foot placement is chosen by the operator, a motion planning module is invoked to plan a trajectory for the leg, which has taken all stability and collision constraints (both with itself and the environment) into account. This planned motion can be visually inspected and modified by the operator in the integrated 3D viewing environment. Currently, multiple motion planning algorithms have been implemented in order to explore the trade-off between them. The output plans can be used to directly command the robot.

B. Advanced ATHLETE Ground Software

The Planning Software Systems Group at the Jet Propulsion Laboratory has developed a suite of software applications that support the development of advanced operations technologies for the ATHLETE rover. Each of these applications is built upon NASA's Ensemble software application framework, an Eclipse-based platform for graphically rich application software development. Many of the application components are derived from capabilities developed and tested on current and recent NASA surface exploration missions, such as the Mars Exploration Rovers and the Phoenix Mars Lander. The ATHLETE suite of ground software applications is designed to mange the complexity of the ATHLETE rover and to increase the efficiency of the operator in conducting prototypical lunar-style tasks. The FootFall software has been designed to interoperate with this suite of ground tools, using the ATHLETE Telemetry Bridge and is encapsulated as a perspective available to the AthleteWorkBench, both of which will be described next.

Managing the telemetry data produced by a lunar research robot poses many of the same challenges as one finds in a flight operations environment. Telemetry data must be captured, stored and made available to the processes that monitor vehicle health and safety. However, less funding, fewer personnel, and an evolving test vehicle are common in the research environment. The ATHLETE Telemetry Bridge (AthleteBridge) software application addresses the problem of providing a realistic ground data telemetry processing system in a research environment.

The AthleteBridge has been developed using industry-standard Java development techniques, including object-oriented design and open-source tools. The resulting application provides mechanisms for telemetry capture, broadcast and storage, while enabling realistic time-delay operations of a research robot. The application design enables rapid deployment of telemetry format changes not only to the AthleteBridge application, but also to related tools in the ATHLETE application suite.

The AthleteBridge receives telemetry from the ATHLETE robot and provides commands to the same. The received telemetry is logged to a persistent storage medium and is made available to interested clients through a broadcast over a computer network. Telemetry is converted from a robot-specific format into a Java standard object format and the Java objects are then broadcast asynchronously via the industry-standard Java Messaging Service (JMS) over a network connection and made available to any listening client applications. The AthleteBridge also further refines the telemetry through a series of pipelines and provides the refined products through the same broadcast mechanism. The AthleteBridge allows for separate time delay of command and telemetry data streams to simulate a lunar time-of-flight distance between operator and robot. The AthleteBridge stores telemetry messages in a database and makes the messages available for playback and analysis. Although the AthleteBridge does not have a graphical user interface, a user can interact with the AthleteBridge through a Java Management Extensions (JMX) web interface for remote control and debugging.

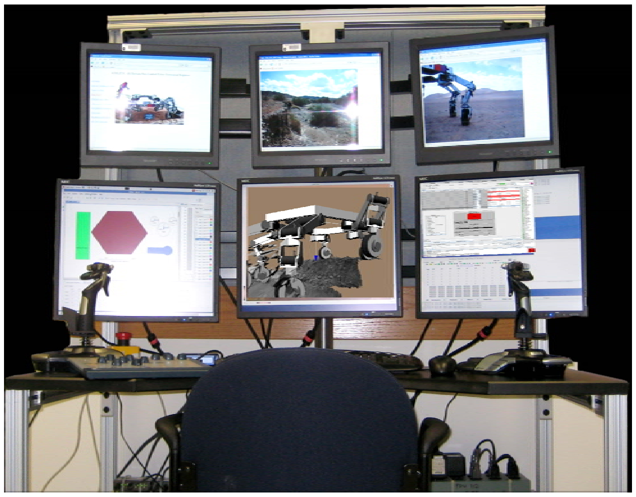


Figure 4. The ATHLETE ground control cockpit

The ATHLETE Application Workbench (AthleteWorkbench) provides ground-based command and telemetry monitoring displays for the ATHLETE robot. The AthleteWorkbench combines several different operations tasks into a single integrated application, including commanding, telemetry monitoring, image browsing, mapping, database annotation, and advanced user input mechanisms. To operate the ATHLETE robot, the driver sits at an immersive cockpit that supplies the computer and display hardware necessary to support the advanced operations software, including a stereo image viewer that provides the operator with depth perception without the need to wear special glasses.

C. Terrain Reconstruction

Terrain reconstruction is initiated when the operator requests that a new terrain be built. Given that ATHLETE has nine stereo camera pairs (not including the ToolCams), we allow the operator to choose which cameras will be used to build the terrain model so that processing time and communications bandwidth are not unnecessarily consumed. Each returned stereo pair is processed through the Ames Stereo Pipeline, which is part of the Ames Vision Workbench (see below). The resulting digital elevation models (DEMs) are then handed to the Traversability module, which classifies terrain traversability based on characteristics of the terrain itself. This approach was used in the Morphin software[10] to find paths for planetary rovers through natural terrain. We are currently using a derivative of the Morphin software in the FootFall system. A similar extension to Morphin created at JPL is part of GESTALT,[11] which is currently being used by the MER rovers on Mars as a means to plan local traversability.[12] This traversability stage discretizes the DEM and analyses each cell for basic characteristics such as slope and evenness. The metrics are then used to color a map of the terrain indicating good places to place a foot (green) and those that would be too rough or risky (red). Finally, the individual DEMs, image textures, and traversability maps are put through a process of GeoSpatial Blending, which combines the data into a single DEM to be displayed in Viz and used for motion planning purposes. The different maps and textures can be laid over the DEM so that operators can examine the results of the various analyses.

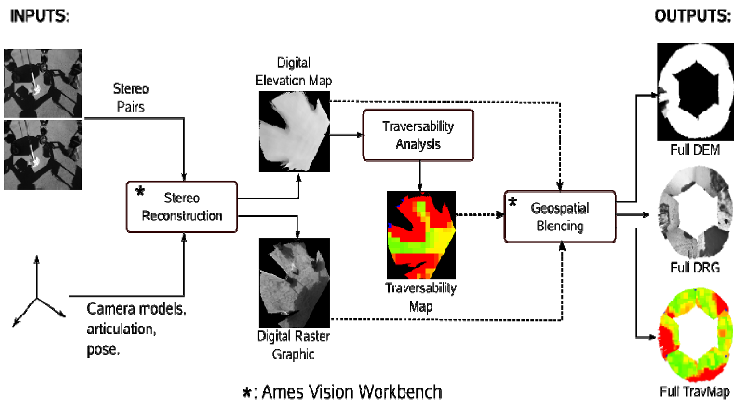


Figure 5. Terrain Reconstruction Module Schematic

The NASA VisionWorkbench[13] is a modular, extensible computer vision framework that supports a variety of space exploration tasks including automated science and engineering analysis, robot perception, and 2D/3D terrain reconstruction. The NASA VisionWorkbench is the synthesis of many computer vision tools that have been under development at Ames for more than a decade. The NASA VisionWorkbench has been used to develop a wide range of NASA computer vision applications, such as creating gigapixel (images with more than $>10^6$ pixels) 2D panoramas (Global Connection Project), 3D terrain modeling from orbital images (Mars Orbiter Camera and Apollo Panoramic Camera), high-dynamic-range images for visual inspection, and texture-based image content matching and retrieval (MER microscopic imager dataset).

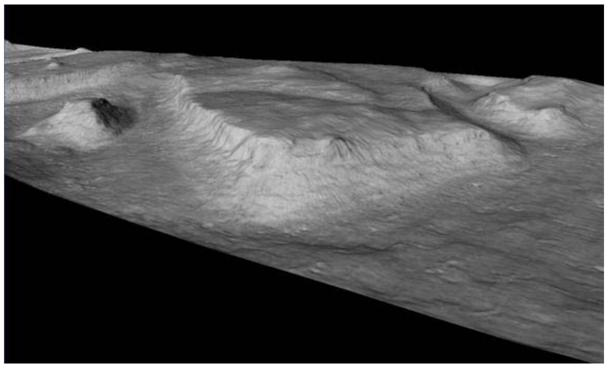
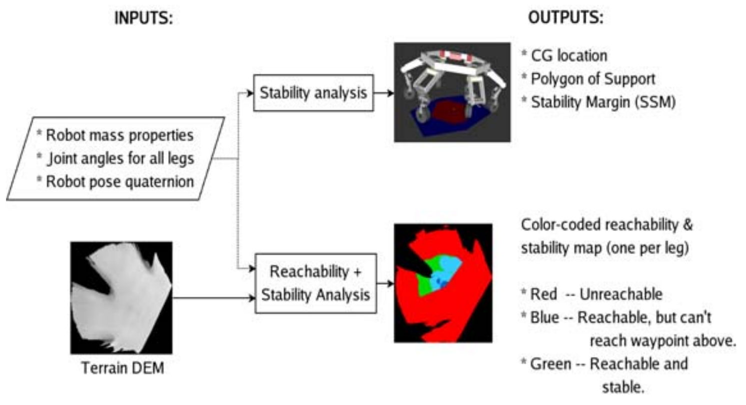


Figure 6. Lunar Terrain Maps Generated by Vision Workbench from Apollo Panoramic Camera data.

D. Reachability Analysis

Once the DEMs have been generated, they can be combined with kinematics and mass models of the rover to find the reachable and stable locations a leg could be moved to. As this takes a fair bit of computation, it is again designed to be done on a leg-by-leg basis, only when the operator requests it. Based on user preference, stability is computed by finding either the regular polygon of support (which is simply the polygon created by all the points where the feet make contact with the ground), or the conservative polygon of support (which is the intersection of all the n-1 polygons of support – i.e. it assumes that any one leg might slip). The center of mass can then be calculated and the Absolute Stability Margin (ASM – a metric of stability) can be calculated by measuring the distance of the CG from the closest edge of the polygon. Reachability is computed by taking the terrain patch within a circle with radius equal to the length of the leg, and checking if an inverse kinematic solution exists at each point. The terrain map is painted green or red if there is a solution or not. Currently the inverse kinematic solution is calculated in an analytic manner, and one must specify full orientation information, so only one solution is provided, if it exists. A future line of work would be to allow some orientation parameters to be left unspecified (such as rotation around the z axis), thus finding if there is any solution for placing the leg at that location. Finally, for all the locations for which a solution exists, we then compute the stability of the robot, were the leg placed in that position. Positions for which there is a solution, but which would not be stable are then colored orange on the terrain map. In order to compensate for unmodeled compliance in the robot (which will be discussed later), we also require all motions to go through waypoints 20cm directly above the start and goal positions. These points must also be checked for reachability, and the map is colored blue if the terrain is reachable, but the compliance compensation waypoint is not.



INPUTS:
OUTPUTS:
* Robot mass properties
* Joint angles for all legs
* Robot pose quaternion
Stability analysis
* CG location
* Polygon of Support
* Stability Margin (SSM)
Terrain DEM
Reachability + Stability Analysis
Color-coded reachability & stability map (one per leg)
* Red -- Unreachable
* Blue -- Reachable, but can't reach waypoint above.
* Green -- Reachable and stable.

Figure 7. Reachability Analysis Architecture

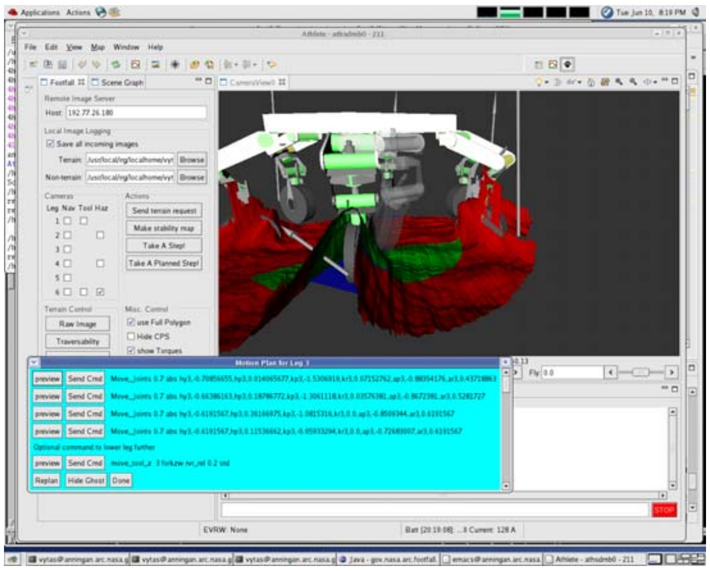
E. Viz (3D Viewing Environment) and the User Interface

Viz[14] is a 3D photorealistic immersive display program for visualizing 3D terrain models generated from rover stereo camera panoramas and robot models. Viz was originally developed at Ames for the 1997 Mars Pathfinder mission, and successfully used by MER for a variety of geo-morphological measurements and virtual exploration of the area surrounding the rovers. Additionally, the Phoenix mission science team is currently using Viz as a decision making aid for planning activities. Viz has undergone many revisions and has been ported to the Eclipse software platform for use on a wide variety of missions and projects where it will be used both as a terrain display and as part of the user interface for controlling different robots.

The FootFall user interface (UI) is mainly concerned with providing an integrated view of data in Viz to facilitate situational awareness by the operator and to interact with the motion planners and the generation of commands. The heart of the UI is a Viz window which shows a model of the robot which is parameterized in real time by the articulation and pose telemetry received from the robot. Likewise, the torque and force telemetry received is used to color the joints (green, yellow, red) and to display a force vector at the wheels. Experience has shown that when walking, the joint torques and forces are critical to monitor and maintain within tolerable ranges.

As terrain reconstructions are requested and other forms of derived data are generated (traversability texture and reachability maps), the operator can select which texture is displayed on the underlying DEM. This allows them to inspect the environment until they find a footfall location that meets requirements for direction of motion, feasibility, and safety. They can then select one of the planning modules (see below) and request a motion plan to that location. If the planner finds a valid plan, it will display the individual commands, and allow the operator to preview each step by adding a "ghost" leg to the display. The operator can (but should rarely need to) hand edit the generated plan steps, and when satisfied can send each command to the rover.

Since FootFall is designed as a plug-in to the AthleteWorkBench application, all the other ground control tools normally used by operators using the AthleteWorkBench can also be activated, such as the ability to manually enter commands, detailed telemetry monitoring, and an emergency stop button.



```
Move_Joints 0.7 abs hy3,-0.70856655,hp3,0.014065677,kp3,-1.5306919,kr3,0.07152762,ap3,-0.88354176,ar3,0.43718863
Move_Joints 0.7 abs hy3,-0.66386163,hp3,0.18786772,kp3,-1.3061118,kr3,0.03576380,ap3,-0.8672380,ar3,0.5281727
Move_Joints 0.7 abs hy3,-0.6191567,hp3,0.36166875,kp3,-1.0815316,kr3,0.0,ap3,-0.8509344,ar3,0.6191567
Move_Joints 0.7 abs hy3,-0.6191567,hp3,0.11536662,kp3,-0.95933294,kr3,0.0,ap3,-0.72683007,ar3,0.6191567
move_tool_z 3 forkzw rvr_rel 0.2 std
```

Figure 8. The User Interface, with a motion plan, a "ghost" of the next step shown for preview, and force arrows and joint torques displayed. The current terrain texture is colored to show reachability.

F. Motion Planning

Motion planning for walking robots can be broken down into two different components: how to take a single step, and how to sequence steps to successfully walk across the terrain. Much of the literature focuses on the second problem, which is often called gait planning. As discussed in the related works section, this is possible because most walking robots have fewer degrees of freedom than ATHLETE and the mechanisms have been specifically designed to simplify the task of walking. ATHLETE is a much more general-purpose robot, with the result that more effort must be expended to plan for a single step. The FootFall software has been largely focused on this part of the problem until now, and we will be starting to tackle the gait-planning problem over the next year. A core concept to understand for motion planning is the difference between task space and configuration space. The task space is the normal 3 dimensional space of our world. This is the space our terrain data is in, and intuitively we want to plan our motion trajectories and obstacle avoidance in this space. The configuration space (C-space) is defined such that the range of allowable motion for each joint is represented as one of the dimensions of the C-space. Thus, the C-space of ATHLETE is a 36-dimensional space, or just six-dimensional if one focuses on a single leg. For some robot designs, especially those with three degrees of freedom per leg, it is possible to define a clear mapping between the task space and the C-space. One way to understand the challenge for ATHLETE is to imagine defining an obstacle free path in task space for the foot of the ATHLETE robot. Since that is a three-dimension path, you are left with three unconstrained dimensions, which correspond to the orientation of the wheel. If you then pick a specific orientation you have a complete set of joint angles to command the robot with. Unfortunately, while you may know that the wheel is collision free, you know nothing about other possible collisions, such as between the knee and some rock. This is not an issue with a simpler robot such as a frame walker, because you can easily define the space that the whole leg will move through. But on a complex robot like ATHLETE, you need to ensure that the entire kinematic chain stays collision free. Clearly one can then iterate through different possible orientations of the wheel until a collision free path is found (assuming it exists), but this is no longer a trivial solution and can rapidly become inefficient to calculate. A further complication is that even in the absence of obstacles, it may not be possible for the robot to follow a given straight line path in task space, even if the start and end points are legal configurations. It is possible for parts of the path to traverse holes in the reachable workspace or to encounter singularities. In an effort to try the simplest approaches first, our first motion planner generated simple task space linear plans. We found this solution to be extremely fragile as most plans violated joint limits or encountered singularities.

For high-DOF robots, it is generally assumed that the best approach is to plan directly in the C-space, while testing each motion segment for task space collisions. There are may approaches to doing this planning, and the right one to use is very dependant upon properties of the specific problem being solved, but most approaches share a few common components. Our approach has been to implement a number of the core modules in a flexible framework, which allows for easy testing of different planning algorithms so that we can find the best solution for ATHLETE. We have also taken the simplifying step of only planning for a single leg at a time so that we are only looking through a six-dimensional space, rather than the full 36-dimensional space of the full robot. The core components, which are shared across different planning approaches, are the models of the robot and environment, the collision checker, and a post-process plan smoothing and optimization algorithm. Next, we will discuss these in detail, including a discussion of the SBL motion planning algorithm we have implemented, and some comments on our initial experiments with a task space A* planner.

1. Models of the Terrain and Robot

While the DEM model of the terrain and a simplified VRML model of the robot are adequate for display in Viz, they are not the right format for motion planning, and thus we need to transform them. As the collision detection algorithm is based on finding intersections between triangles (see below), we need to turn the DEM into a triangulated mesh. This is currently accomplished by using the VisionWorkbench to create a triangulated terrain. Currently we are keeping all the points, but clearly a future optimization would be to simplify the terrain into fewer triangles. Likewise, for computational efficiency we need to transform the VRML model of the robot into one that uses the fewest possible triangles. As it stands, we already simplify the model of the robot for display in Viz so that the 3D rendering can be responsive to user interaction. Currently we use a one-time process for manually building simplified models of the robot. The Viz model is then further simplified for the collision checking, since the runtime is directly dependant on the number of triangles. Clearly a concern here is that the various models of the robot can get out of synch with each other, especially as components and tools are added and removed and the robot is modified. A future line of work will be to have a single central model of the robot, ideally taken from the fabrication CAD designs, and then enable the automatic generation of the various derived models (Viz and collision avoidance models).

2. *Collision Detection*

Two kinds of collisions are checked for ATHLETE: self-collisions and environmental collisions. As discussed above, both of these rely on triangulated-mesh representations of the robot and terrain, which are extracted from the CAD model and Digital Elevation Map respectively.

Our application uses the Proximity Query Package (PQP) library[15,16] from the University of North Carolina. PQP is able to efficiently detect collisions by representing triangulated mesh models as hierarchical trees of oriented bounding boxes (OBB-Trees). Given a robot or terrain model consisting of a group of polygons (most commonly triangles), PQP pre-computes a hierarchy of tightly fitting rectangular oriented bounding boxes (OBB) and organizes them in a tree. The boxes near the root of the tree span a large number of polygons. To form the next level down the tree, these boxes are divided in two along the longest axis as possible ("top-down" approach). The division and addition of levels continues until polygons can no longer be divided. The idea then is that by traversing the tree from top to bottom collisions are checked with progressively higher levels of resolution. A "separating-axis" theorem is used to efficiently identify collisions between any two bounding boxes as the tree is explored, often without traversing the whole tree. Because OBB-Trees are able to fit the model tightly, they allow for fewer levels of the tree to be explored as compared to approaches using other volumes (such as spheres).

For the purpose of *self-collisions* the ATHLETE model is passed to the collision checker and decomposed into a series of components. These include the chassis, as well as six each of hip yaw, hip pitch+thigh, knee pitch, knee roll+calf, ankle pitch and ankle roll+wheel, corresponding to all six legs. PQP provides the option of signaling a collision when two objects come within some distance $\delta \geq 0$ from each other. Our initial testing showed that the appropriate value of δ is not uniform throughout the robot. In particular, parts that naturally come in close proximity due to the construction of the robot require a small δ, such as the chassis-thigh pair. On the other hand, components that are ordinarily far from each other, such as wheel-chassis pairs or parts of different legs, need larger δ*'s* for safety because we have less certainty about their actual relative positions due to compliance and encoder noise. With this in mind, a lookup table has been manually generated indicating values of δ for each pair of components of the robot. In the case of *environment collisions*, the value of δ is the same for all parts of the robot.

For ATHLETE, PQP provides the lowest-level collision checker, which verifies any given individual configuration for collisions. At a higher level, the configuration space planners have a need to check edges, which are segments connecting two configurations. Exact and approximate techniques have been presented in the literature for this purpose.[17] We currently use an approximate technique that iteratively bisects a linear edge between the two configurations, until a collision is detected or a maximum, pre-specified, level ε of resolution is achieved without collisions. This technique is ε-accurate, with ε currently set to 0.01rad for our application. The bisection technique is observed to work well in detecting collisions because if a given segment is in collision, the middle point of the segment has a high probability of being in collision. Thus by performing successive sub-segment bisection a path in collision can generally be detected quickly. More details on this bisection collision checking are provided in the following section.

3. *Single-Query Bi-Directional (SBL) Motion Planner*

For the most general step planning, we make use of a sampling-based technique that explores the configuration space of a single leg to find collision-free paths between the current and desired configurations. Specifically, we have chosen the Single-query Bi-directional planner with Lazy collision checking (SBL) developed by G. Sanchez and JC Latombe.[18]

As with all other sampling-based motion planning approaches, SBL works on the principle that it is considerably cheaper computationally to check if a single configuration is collision-free than to explicitly construct the C-Space with obstacles. The search for feasible paths is conducted by sampling configurations between the start and goal, and verifying if (a) they are feasible, and (b) they can be connected without collisions.

Since SBL only explores a fraction of the configuration space, one might wonder what guarantees it offers in terms of finding a solution. To answer this question, two concepts are useful: *completeness* and *denseness*. Completeness is generally defined as the ability of a planning algorithm to report whether there is a solution connecting any two configurations in a finite amount of time. Because sampling-based techniques do not explicitly construct the configuration space, they cannot provide this guarantee. Specifically, these techniques cannot conclusively say that a solution does NOT exist. Therefore, more relaxed notions of completeness are defined for sampling-based techniques, depending on the sampling approach they use. Deterministic-sampling algorithms are said to be *resolution complete* if they are shown to sample the space densely. This means that they will find a solution in finite time if one exists, but might run indefinitely if there is no solution. Similarly, randomized algorithms are said to be *probabilistically complete* if the probability of them finding a solution converges to 1 as the number of samples increases. This completeness is highly dependent on *dense sampling*, which means that, while the algorithm will not sample every point in the space, it will generate samples that get arbitrarily close to any point in the space as time goes by. SBL has been shown to be probabilistically complete, since it performs dense randomized sampling.

SBL has characteristics that make it adequate for single-stepping on ATHLETE. First, it has been experimentally observed to significantly reduce planning times in six-DOF robots by virtue of delaying collision checking until it is absolutely necessary. This lazy approach particularly benefits problems where triangulated meshes with a large number of triangles represent the robot and its environment, as is the case for ATHLETE. Second, SBL explores the configuration space bi-directionally. This has been observed to have a higher likelihood of success in finding paths through narrow-passage and bug-trap situations. Such situations arise in the ATHLETE robot when trying to take a step toward the inside of the chassis or in highly cluttered environments.

SBL receives two parameters: the maximum number of milestones to generate (s) and a distance threshold (ρ) that defines "closeness" between configurations. The algorithm proceeds by growing two C-Space trees T_1 and T_2 rooted at q_I and q_G toward each other. On every iteration, one of the trees is selected at random with probability 0.5, and a new milestone m_{new} is added to it (EXPAND-TREE step). The planner then checks if a connection can be established between the trees (CONNECT-TREES step), and if so it generates a candidate path τ from q_I to q_G. This path includes a segment called a *bridge*, connecting m_{new} to m', the nearest milestone in the opposite tree. If τ is found to be collision-free, success is returned. Otherwise iterations continue for s steps, at which point failure is returned. This means that either no path exists, or SBL was unable to find one.

The EXPAND-TREE step proceeds as follows: from the selected tree T, an existing milestone is selected at random with probability $\pi(m)$, which is inversely proportional to the density of milestones of T near m. Then, a collision-free configuration is randomly selected within an adaptively-selected distance $r \leq \rho$ of m, and is added to T as the new milestone m_{new}. This selection strategy distributes the exploration around areas reachable from the root configurations, and at the same time prevents over-sampling. It should be noted that only m_{new} is checked for collisions at this stage, not the segment connecting it to m. Segment checks are postponed until they are absolutely necessary in the CONNECT-TREES step. This "lazy" approach has the effect of reducing the total number of expensive collision checks.

The CONNECT-TREES step of SBL is executed when the L_∞ distance between m_{new} and m' is smaller than or equal to the distance threshold ρ. At this point the candidate path τ is checked for collisions in detail by the TEST-PATH routine, and τ is returned as the motion plan if it is collision-free; otherwise, iteration continues. Within TEST-PATH individual segments are assigned an integer check score (κ) of 0 if only the endpoints are known to be collision-free, or 1 if the midpoint is also collision-free. The general rule is that for any κ, a total number of points $2^\kappa+1$ have been checked in that segment. When the length of the sub-segments is smaller than a pre-defined value ε the segment is declared collision-free. Note that the iterative bisection checking strategy is followed to a resolution ε as explained in Section 2 above.

For further details on SBL the reader is referred to the original paper by Sanchez and Latombe. The complete single-step planning process for ATHLETE can now be summarized as follows: once a terrain model has been acquired and analyzed for reachability (Sections II.C and II.D), the human operator selects a leg to move, and a target location for that leg's wheel. This fixes the initial and goal configurations (q_I, q_G respectively). In theory, using SBL to connect q_I and q_G would provide the solution we need. However the ATHLETE robot is, by design, highly compliant and this means that as a leg is lifted, the redistribution of loads among the remaining legs causes the legs to comply and the chassis to sag. To account for this sagging at planning time, we generate intermediate lift and drop waypoints (q_L, q_D) depending on the situation, as shown in Fig. 9, and these waypoints are connected using SBL, with simple linear task-space motions between q_I-q_L and q_D-q_G.

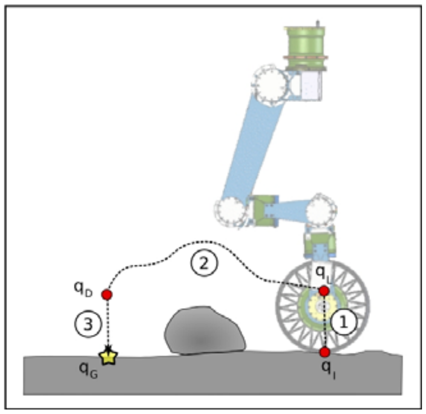


Figure 9. Execution of a step along a planned, obstacle-free path.

4. Path Smoothing

It is important to point out that SBL generates plans that are usually not smooth and acceptable for execution on ATHLETE due to the nature of randomized sampling. This usually results in large and unnecessary swings of the leg, as well as jerky back-and-forth motions as shown in Fig. 10.1. Therefore, the plans are post-processed by iteratively discarding unnecessary nodes via Dijkstra's algorithm, and bisecting the simplified plan to provide new nodes to be used for further path simplification. These steps are repeated until the improvement between successive iterations is no longer significant (measured by the reduction in total Euclidean path length in configuration space). The smoothing process is summarized in Fig. 10. It can be seen that, as the smoothing progresses, the path gradually becomes shorter. Our tests have resulted in sufficiently smooth paths when the length change between successive iterations drops below 25%. The total number of waypoints is also reduced, usually obtaining at most five steps. We have found that this smoothing process takes significantly less time than the original SBL planning step.

The result of combining this form of path smoothing with the output of the SBL planner has been very compelling. At first we had been hesitant to use the SBL planner because the plans it produced seemed particularly poor, with unnecessarily large erratic motions, even in the absence of obstacles. This behavior was the result of the random sampling used to search the configuration space. With the path smoothing in place, the generated plans are smooth and simple, with no extra erratic motion, and traverse a fairly minimal path between start and end points. One can think about this two-step process as first finding a *feasible path* (SBL planner) through the high dimensional configuration space, and then using that path as a seed for an *optimization process* (path smoothing) that produces a simple and efficient set of motions to arrive at the goal.

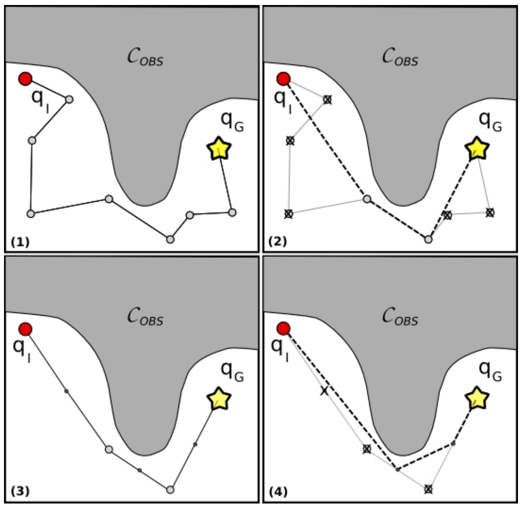


Figure 10. Path smoothing in a hypothetical 2D C-space: the initial and goal configurations (q_I and q_G) are separated by a C-obstacle. (1) Motion plan with N_1 nodes generated by SBL is usually not smooth; (2) the shortest path is found using Dijkstra's algorithm, resulting in a smoother motion plan with $N_2 \leq N_1$ nodes; (3) the simplified path is bisected, a step that adds N_2-1 nodes; (4) Dijkstra's algorithm is re-run.

5. A Task Space Planner*

Despite the success of the SBL with smoothing approach outlined above, we are interested in exploring other approaches. Our main desire is to explore the quality of plans and runtime constraints of approaches that do not depend upon randomized configuration space search. As mentioned elsewhere, we initially implemented a simple linear task space planner. This often fails due to the linear path not being entirely within the workspace of the leg. We have now extended that approach by implementing an A* task space planner. This approach discretizes the task space into 20cm grid cells and then propagates a wave front from the start configuration until the goal configuration is achieved. We use the same collision detector and path smoothing algorithms described above to ensure a safe and efficient plan. While this work is still very early in its development, our initial experiments indicate that plans can be successfully generated in a comparable amount of time as the SBL planner. The main weakness of this approach is that its success is dependant upon the resolution of the grid cells – if the resolution is too coarse it will not succeed in finding a path, but as the resolution is increased the run time quickly escalates. Over the near term we will be comparing the quality of plans produced by the two planners over a variety of terrains and stepping situations. A likely result is that we will keep both methods in FootFall, with a coarse-grid A* planner being the default approach, and the SBL planner used as a more complete back-up method should A* fail to find a path.

III. Field Test Results and Future Work

In June of 2008, NASA's Human Robotic Systems Project, part of the agency's Exploration Technology Development Program, held an Integrated Field Test at Moses Lake, Washington. During this test, robotic systems from a variety of NASA centers were brought to the sand dunes outside of Moses Lake where they tested possible mission scenarios. ATHLETE was involved in a number of these tests, including long distance traverses, docking, use of the new habitat mockup, navigating steep slopes, and testing the FootFall software. The environment in the sand dunes was harsh, and challenged many of the robots with rain, sand storms, mechanical failures, and schedule slips. As a result, there was only time to perform a few tests of the FootFall software, but in that time we were able to successfully demonstrate that the overall system worked and was capable of generating valid motion plans for

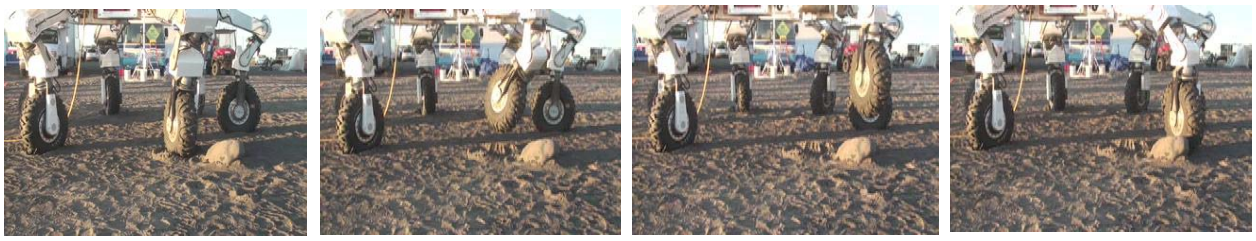


Figure 11. AHLETE taking a step over a rock -- motion planned by the Footfall Planner Software

ATHLETE to step over a rock to an operator specified location. These tests were also successful from the perspective of highlighting which parts of the solution are currently fragile and should be improved over the next year. The most challenging aspects of operations at the moment are: the compliance in the robot, visibility, and self imaging. In order to make progress towards next year's goal of walking the entire robot forward by stepping all the legs at least once, we will need to tackle all these issues.

1. Compliance

ATHLETE is somewhat compliant, which results in a discrepancy between planned motions and actual executed motions. Most of this compliance comes from the joints and from the wheels, which are deformable. As a result it is common to command the leg to lift by 20 cm while the wheel never brakes contact with the ground, but rather only manages to unload tension in that leg while the frame flexes and droops. This is a problem because a motion may be generated which avoids collision with the environment, but at run time the compliance in the robot results in the leg colliding with the rock, as it never lifted itself high enough. For the field test, we implemented a number of heuristics that compensated for this behavior and allowed us to generate valid collision free paths. The first heuristic we used was to modify the generated DEM to "lift" the terrain up by 15 cm. This simulates the effect of the robot sagging and coming closer to the ground, and thus the planner naturally finds a path that lifts the leg higher above the terrain. Another heuristic we used was to insert way points 20 cm above the chosen start and goal positions, and then plan a route between these "lifted" points. Finally, we also maintained a minimum safety buffer zone around all obstacles, which helped avoid plans which seemed feasible to the planner, but which involved passing very close to collisions. While these heuristics worked and allowed us to safely step over the rock, they came at a cost. Each modification listed above effectively reduced the workspace that the robot could plan through, making it more difficult to find a valid motion plan. As an operator, it was easy to see that some locations should be possible to step to, but the motion planner could not find a valid plan. This was compounded by the fact that the new habitats carried on top of the robot required a relocation of the generators, which were now hanging below the chassis and severely limited the range of motion of the hip joint. Operationally, we may be able to simply stand the robot up taller before attempting to walk, in order to get more room to maneuver between the ground and the chassis. We will explore this, and other, operational solutions over the next year.

While operational approaches will help inform us about solutions to walking, the main focus of our efforts over the next year will be to remove most of these heuristics with a more robust solution. Current plans are to develop an on-board sag-compensation control mode. This approach would use a model of the compliance of the robot, and feedback from on board sensors, to actively control all the support legs to maintain the position of the chassis as the "lift leg" breaks contact with the ground. This will keep the robot from drooping and thus minimize the difference between the planned and executed motions. One can think of this as a similar on-board capability as the compliant driving that is currently used on ATHLETE. In this mode, the legs actively actuate to maintain the level of the chassis as the robot drives over uneven terrain and hills. Conceptually, we will be extending this "level chassis guarantee" to the walking and tool use domains. From an architectural point of view, this split between planning for a single leg and depending on on-board sag-compensation offers a major advantage over planning for the entire 36-DOF configuration space of the robot, which is computationally expensive and even more fragile to the realities of execution.

2. Visibility

While ATHLETE has many stereo camera pairs, their placement is optimized for driving, not walking. The six NavCams in the outer ring are pointed out towards the horizon and on flat terrain only see ground that is beyond the maximum reach of the legs. Thus, the three inward facing HazCams must be used. Figure 12 shows a terrain map using all of the NavCams and HazCams, and one can see the empty gap between the two where no data is gathered. Careful inspection will reveal that while the above-mentioned gap exists, the bulk of the missing data is actually caused by the occlusion of the terrain by the legs themselves. Because we must use the HazCams, which look through the body of the robot, the very leg one wants to move blocks the view of much of the workspace of that leg. Operationally we are able to take steps that are tangential to the circle of the robot. During the field test we were able to accommodate this limitation, though it also further reduced our available workspace, limiting the range of possible rock-avoiding steps we could take.

Looking forward though, our objective for the next year is to step every leg such that the robot translates its center of mass forward in a given direction. This will require moving directly into a blind spot for at least one leg. A basic operational solution will be to lift a leg, re-image the terrain, and then plan for placing the leg down again. While this will work, it is unwieldy and will not be extensible to walking over a boulder field in real-time.

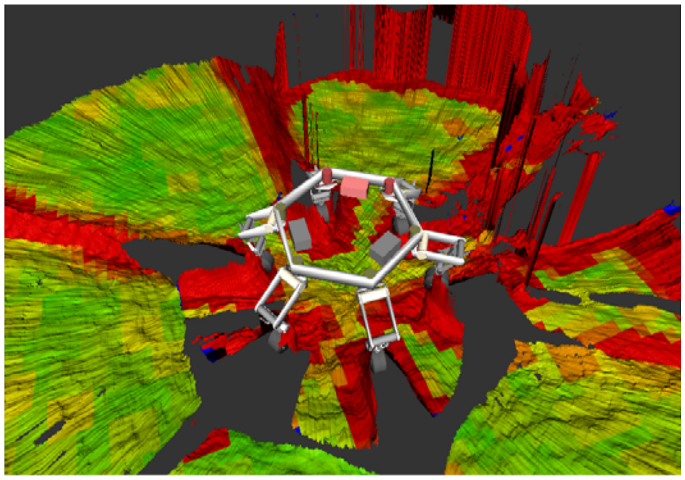


Figure 12. Limitations to Visibility

The most appealing solution for walking would be to add cameras optimized for monitoring foot placement of the robot. Unfortunately, due to complexities in the wiring and computational infrastructure, it is unlikely that more cameras will be added to the current version of ATHLETE. Thus, over the next year we will also explore approaches to incrementally build terrain maps integrated over time, which would preserve terrain knowledge even as it becomes temporarily occluded. Research and development on Bundle Adjustment[19] software within the group over this last year has shown promise for enabling the desired terrain integration. Conceptually, Bundle Adjustment enables precise alignment of overlapping images by optimizing the supplied camera parameters such that a smoothly integrated terrain model is produced. One challenge of working with ATHLETE has been camera calibrations – given the harsh conditions, slight compliance in the frame, and rigorous testing that ATHLETE experiences, the cameras are often slightly miss-calibrated to each other. The bundle adjustment code will also enable more robust stereo reconstruction because it is tolerant of, and corrects for, slight errors in the camera models.

We have also explored using the ToolCams to gain visibility into the occluded areas. These cameras, which are mounted directly above the wheels, give high-resolution imagery of the environment around the wheel, and will not be occluded by the legs. Unfortunately, it is difficult to incorporate their data into a meaningful terrain model, because of perspective and compliance. Their location close to the ground creates such a difference in perspective that it is challenging to integrate the scene with terrains maps reconstructed from the chassis mounted HazCams. Deeply compounding this is the error in the camera models caused by their location at the end of a long compliant kinematic chain. As compliance models of the robot are developed over the next year, we will revisit the ToolCams and see if the problem has become more tractable.

3. *Self Imaging*

The two issues of compliance and visibility combine together into another challenge: the ability to account for portions of the robot structure that appear in the images it acquires. Since we are using the HazCams, which look through the robot, they end up imaging the legs of the robot, and reconstructing them as part of the terrain. This has many undesirable side effects, such as causing our collision detector to think the robot is in collision with the terrain. The first effort to deal with this was to use software from JPL that uses the forward kinematic solution to mask out chunks of the image based on where the legs are supposed to be. Unfortunately, with the compliance in the robot, the results were not beneficial, as the masks often missed parts of the actual legs. Our current approach is to flatten part of the image mesh used by the collision checker around the location of the foot. This frees us to plan without being in a false-positive collision, but has two limitations. First, there are often stereo-reconstruction artifacts left in the image beyond the “flattened” area which do not exist in reality (these are “shadows” of the edge of the leg). Furthermore, there is concern about cutting data out of the system – in order to ensure that we flatten enough terrain to account for position errors from the compliance we have to use a large template. This large “flattened area” means that a rock immediately next to a wheel may be flattened away, and then ignored during the planning process. Again, we will revisit this issue as the compliance models are developed, hopefully allowing for more minimal alteration of the source data.

IV. Conclusions

A couple high level observations have come out of working on this project. First, it is interesting to realize that most walking robots have had the luxury of being physically designed to simplify the task of walking. ATHLETE on the other hand has a much more complex set of constraints, and as such its design is much more general purpose and flexible. While this makes the task of planning for walking more difficult, it is a challenge that every biological system faces, and which must be considered by all future general-purpose robots. Another realization is that most motion planning algorithms for robots with high DOF have been developed mainly in simulation and few of them have been implemented on real robots operating in natural unconstrained environments with noisy sensors. As a result, issues of limited terrain visibility and spatially/temporally degrading terrain knowledge are poorly explored. A consequence of noisy sensors and actuators is that plans are only relevant for a short distance from the robots current configuration. Beyond a step or two, joint space plans will rapidly become meaningless, especially if on-board active compliance is being used for sag-compensation. As a result, mid-level and higher route planning for walking robots needs more attention. Specifically, terrain classification and route planning for wheeled robots is fairly well understood, but how does one extend this to walking robots? Terrain traversability for a walking robot is ultimately dependant on the configuration of the robot while traversing that exact location, but planning through configuration space over large distances is of limited value. We have made an initial attempt at defining an appropriate set of hierarchical planners to handle this problem, which will be published shortly,[20] and we are eager to start implementing these ideas over the next year.

Finally, the challenges we faced dealing with self-imaging were particularly philosophically interesting. Biological systems clearly have an ability to segment themselves from the surrounding environment in every sensory modality used (vision, sound, force, etc). It is fascinating that we are now dealing with the same issues as we push the boundaries of robotic operation in natural environments. Ultimately, I suspect that this issue will be solved by ever richer multi-modal models of the environment which are integrated over time, and which enable robots to more clearly define “self” versus “not-self.” This, of course, lets the imagination run wild with ideas of intelligence and self-consciousness deriving from the basic computational requirements of orchestrating motion and manipulation in unstructured environments.

Acknowledgments

This work was performed as part of the Human Robotic Systems Project, which is funded by NASA’s Exploration Technology Development Program. The authors would like to thank Brian Wilcox and the whole ATHLETE team for allowing us the opportunity to work with such an amazing robot. We would also like to thank Kris Hauser at Stanford University for providing some of the SBL and collision checking software which was modified and incorporated into this work.

Part of this work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1]Wilcox, B. H., Litwin, T., Biesiadecki, J., Matthews, J., Heverly, M., Morrison, J., Townsend, J., Ahmad, N., Sirota, A., and Cooper, B. “ATHLETE: A cargo handling and manipulation robot for the moon.” *Journal of Field Robotics* Vol. 24, No. 5, April 2007, pp. 421–434.

[2]Heverly, M., and Matthews, J., “A wheel-on-limb rover for lunar operation.” *Proceedings of the Ninth International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*. February 25-29, 2008.

[3]Collins, C., “Stiffness Modeling and Force Distribution for the All-Terrain, Hex-Limbed Extra-Terrestrial Explorer (ATHLETE)” *Proceedings of ASME Design Engineering Technical Conferences*, Las Vegas, NV, 2007.

[4]Campbell, D., and Buehler, M. “Preliminary bounding experiments in a dynamic hexapod.” *Experimental Robotics VIII*, Editors Siciliano, B., and Dario, P., Springer-Verlag, 2003, pp. 612–621.

[5]Kimura, H., Fukuoka, Y., and Cohen, A. H. “Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts.” *International Journal of Robotics Research*, Vol. 26, No. 5, 2007, pp. 475–490.

[6]Playter, R., Buehler, M., and Raibert, M. “BigDog” *In Proceedings SPIE Defense and Security Symposium*, 2006.

[7]Poulakakis, J., Smith, A., and Buehler, M. “Modeling and experiments of untethered quadrupedal running with a bounding gait: The scout II robot.” *The International Journal of Robotics Research*, Vol. 24, No. 4, April 2005, pp. 239–256.

[8]Kar, D. “Design of statically stable walking robot: A review.” *Journal of Robotic Systems*, Vol. 20, No. 11, 2003, pp. 671–686.

[9]Wettergreen, D. “Robotic Walking on Natural Terrain: Gait planning and behavior-based control for statically-stable walking robots.” Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA, 1995.

[10]Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., and Schwehr, K. “Recent progress in local and global traversability for planetary rovers.” *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, April 2000.

[11]Goldberg, S., Maimone, M., and Matthies, L. “Stereo vision and rover navigation software for planetary exploration.” *Proceedings of IEEE Aerospace Conference*, volume 5, 2002.

[12]Maimone, M., Leger, C., and Biesiadecki, J. “Overview of the mars exploration rovers’ autonomous mobility and vision capabilities.” *Proceedings of IEEE International Conference on Robotics and Automation (ICRA) Space Robotics Workshop.* Roma, Italy, 14 April 2007.

[13]NASA VisionWorkbench. 2006. NASA Technology Number ARC-15761, http://ti.arc.nasa.gov/visionworkbench/

[14]Stoker, C. R., Zbinden, E., Blackmon, T. T., Kanefsky, B., Hagen, J., Neveu, C., Rasmussen, D., Schwehr, K., and Sims M., “Analyzing Pathfinder Data Using Virtual Reality and Super-resolved Imaging,” *Journal of Geophysical Research*, Vol 104, No E4, pp. 8889-8906, April 25, 1999.

[15]Gottschalk S., Lin M., and Manocha D. “OBB-Tree: A hierarchical structure for rapid interference detection.” *In Proceedings of ACM SIGGRAPH*, 1996. pp 171–180.

[16]PQP: The Proximity Query Package. Software Download: Hhttp://www.cs.unc.edu/~geom/SSV/H

[17]Schwarzer, F., Saha, M., and Latombe, J. “Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments”, *IEEE Transactions on Robotics*, June 2005.

[18]Sanchez, G., and Latombe, J. C. “A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking.” *Proceedings International Symposium on Robotics Research (ISRR'01)*, Lorne, Victoria, Australia, November 2001.

[19] Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A. “Bundle Adjustment – A Modern Synthesis.” In *Proceedings of the international Workshop on Vision Algorithms: theory and Practice*, September, 1999.

[20]Smith, T., Barreiro, J., Smith, D., SunSpiral, V., Chavez, D. “ATHLETE's Feet: Multi-Resolution Planning for a Hexapod Robot.” To appear in proceedings, *International Conference on Automated Planning and Scheduling (ICAPS)*, Sydney, Australia, Sept. 2008.