# ATHLETE's Feet: Multi-Resolution Planning for a Hexapod Robot

**Tristan B. Smith  Javier Barreiro**
**David E. Smith  Vytas SunSpiral**
Intelligent Systems Division
NASA Ames Research Center
Moffett Field, CA 94035–1000
{Tristan.B.Smith, Javier.Barreiro,
David.Smith, Vytas.SunSpiral}@nasa.gov

**Daniel Chavez-Clemente**

Department of Aeronautics and Astronautics
Stanford University
Stanford, CA 94305–4035
dchavez@stanford.edu

**Abstract**

ATHLETE is a large six-legged tele-operated robot. Each foot is a wheel; travel can be achieved by walking, rolling, or some combination of the two. Operators control ATHLETE by selecting parameterized commands from a command dictionary. While rolling can be done efficiently with a single command, any motion involving steps is cumbersome - walking a few meters through difficult terrain can take hours. Our goal is to improve operator efficiency by automatically generating sequences of motion commands. There is increasing uncertainty regarding ATHLETE's actual configuration over time and decreasing quality of terrain data farther away from the current position. This, combined with the complexity that results from 36 degrees of kinematic freedom, led to an architecture that interleaves planning and execution at multiple levels, ranging from traditional configuration space motion planning algorithms for immediate moves to higher level task and path planning algorithms for overall travel. The modularity of the architecture also simplifies the development process and allows the operator to interact with and control the system at varying levels of autonomy depending on terrain and need.

## Introduction

ATHLETE (All-Terrain Hex-Limbed Extra-Terrestrial Explorer) is a large six-legged robot developed at the Jet Propulsion Laboratory (see Figure 1). ATHLETE is a flexible platform designed to serve multiple roles during manned and unmanned missions to the moon, including transportation, construction and exploration. It is intended to be remotely operated from earth or by astronauts on the moon.

In the two years since prototype ATHLETE (1/2 scale) robots became operational, a wide array of capabilities have been demonstrated (Wilcox et al. 2007; Heverly and Matthews 2008). ATHLETE can roll on smooth terrain, 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 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,[1] making operation tedious.

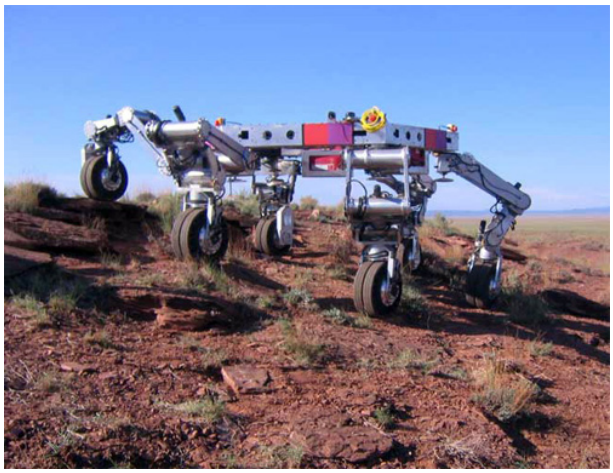

Figure 1: ATHLETE on a hillside.

The goal of this project is to make ATHLETE operation faster and more efficient by automatically generating sequences of commands. These sequences are especially important in situations that require walking, but even those situations must incorporate rolling from the outset. As an example, the most efficient way to start climbing onto a ledge is to lift up the front leg, roll forward, and put the foot down on the other side.

Often, motion planning for such robots has been done with configuration space planning. Configuration space is the set of all possible robot configurations (each determined by a unique set of joint angles), and a path through that space represents a sequence of moves the robot can make to get from one configuration to the next.

Configuration space planning for ATHLETE was implemented by Hauser, Bretl, and Latombe (2006) but ran into various difficulties due to the size and complexity of ATHLETE's configuration space. First, search was slow; moving one body length on irregular ground took 26 minutes to com-

[1] Active compliance for walking is significantly more difficult to implement. For example, how should weight be redistributed when a leg is picked up?

pute. Second, a complete and accurate model of the environment was assumed, but is not available in practice. Finally, rolling was not included but is expected to be a significant part of an appropriate plan for almost any terrain.

The realities of ATHLETE's operational environment suggest that configuration space planning is much too detailed for anything beyond immediate moves. The quality of terrain knowledge degrades quickly over distance; due to its imprecision, stereo reconstruction is restricted to the nearest 10 meters or so (see Figure 2) and even that data is less accurate as distance increases. Planning beyond that distance requires lower resolution (e.g. satellite) imagery. Uncertainty regarding future configurations also increases quickly as terrain inaccuracies combine with the inability to accurately model the way that ATHLETE's joints often sag or shift under pressure. Even worse, any command to roll with active compliance will lead to an entirely unpredictable configuration.

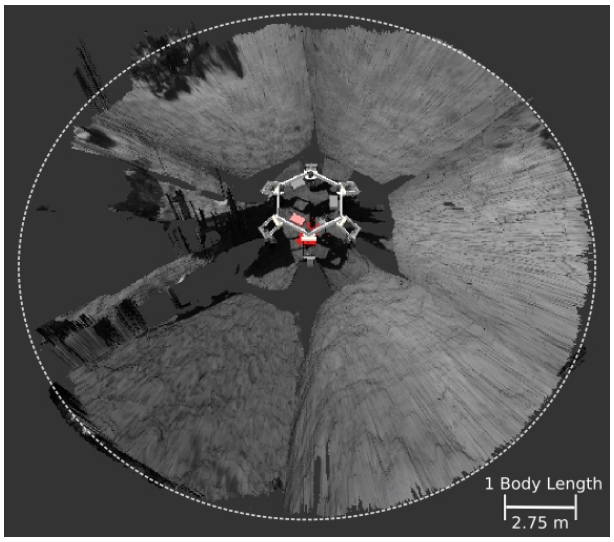


Figure 2: Stereo reconstruction of terrain using ATHLETE's onboard cameras. Note the blindspots outside of each leg.

For these reasons, we have chosen to break down the planning problem into multiple layers, each operating at a different level of granularity and over a different horizon.

Consider moving ATHLETE from a lunar landing site to base camp, hundreds of meters or even kilometers away. At the highest level, a route planner will use low-resolution data to determine a rough route to base camp (Figure 3). This planner prefers smooth terrain, avoids impassable terrain, and might incorporate other high level goals, such as minimizing distance or energy, or including desired waypoints for scientific observation or mapping.

Subsequent planning layers take over planning within the range of ATHLETE's onboard cameras (the dotted circle in Figure 3). Given a desired direction of travel and the camera data, a chassis planner determines a path for the chassis from the current position to the edge of the field of view in the desired direction (Figure 4). This path travels between, over, and around obstacles and elevation changes, so that rolling

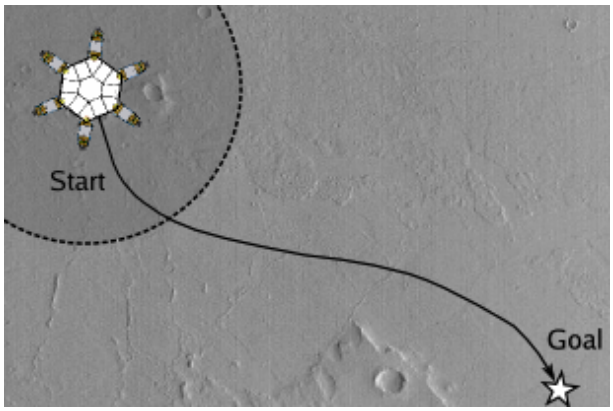


Figure 3: A route plan at the highest level will extend well beyond the range of ATHLETE's cameras.

is likely to work and walking, if necessary, is likely to be straightforward. For example, on flat terrain littered with small boulders, this planner might find a path where combinations of zig-zags and chassis rotations allow ATHLETE to get across the terrain simply by rolling and occasionally picking up feet.

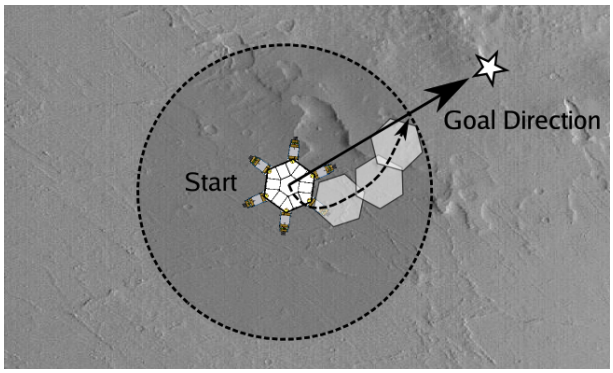


Figure 4: The chassis planner combines rotations and translations to meet the route plan at the horizon.

At the next level of granularity, a move planner will search for combinations of rolls, steps, and chassis shifts that move forward along the given chassis plan while maintaining stability. For example, if the next move should be a step, the move planner must decide which leg to use, and where to step, based on stereo analysis of the terrain.

Finally, at the lowest level, traditional configuration space planning will be used to determine sequences of joint manipulations that achieve each specific step while avoiding collisions (with itself and the terrain) and obeying torque limits.

Interestingly, the multiple levels of planning parallel the multiple granularities of execution available. High level commands (rotations, rolls and chassis shifts) are determined by the chassis and move planners while the low-level commands required for stepping are determined by the configuration space planner.

Note the similarities with how we humans travel. Imag-

ine hiking up a rocky mountainside. At a high level, you scan the horizon for a goal destination and plan some zigzags to get there. You then head off in the initial direction, searching for a path with minimal ups and downs. Along that path, you select specific stable places to put your feet. Meanwhile, your low-level body controls figure out how to get each foot to the desired locations. Occasionally, you look up and replan based on new information, or because you get stuck. This analogy suggests that our approach is reasonable and also that the resulting plans are most likely to be understandable and acceptable to human operators.

Even though the proposed planners will initially only suggest commands to an operator, they will be tightly integrated with each other and with execution. Any planner could be required to replan if either 1) the goals in its current plan cannot be achieved by a lower-level planner, 2) execution has resulted in a state that conflicts with its current plan or 3) the user modifies goals in the current plan.

Our proposed architecture combines algorithms familiar to the robotics community with those more familiar to an AI audience. Low level kinematics and physical constraints can be abstracted away from planning algorithms used for the higher levels. Design is also driven by the need for incremental progress. The low-level planner is already available and can be used by operators while higher level planners are under development.

The next section describes how ATHLETE is operated today. We then overview our architecture before describing each level of planning in more detail. Finally, we mention related work, and discuss possible future extensions.

## ATHLETE Today

ATHLETE has six legs attached to a hexagonal chassis and is omni-directional. Each of ATHLETE's six legs has hip, knee, and ankle joints (each with 2 degrees of freedom), resulting in 36 degrees of kinematic freedom.[2] At the end of each leg is a multi-purpose wheel. The wheel can be locked to serve as a foot or unlocked to roll; it can also be used to operate tools using a mechanical adaptor.

ATHLETE's design includes 15 pairs of stereo cameras; one pair on each side of the hexagon, one at the end of each foot (primarily for tool operation) and 3 pairs on the inside of the hexagon to view the terrain between and around the feet. Together, these cameras cover most of the terrain within a few body-lengths although some blind spots (see Figure 2) exist around the legs. Other sensors monitor forces and joint torques.

### Modus Operandi

Operators use laptops to send parameterized commands to ATHLETE. While the command dictionary is sizeable, human operators almost exclusively use a small handful. The most common mobility commands include stepping, rolling and shifting the chassis to rebalance. The roll command allows active compliance; joints are adjusted to keep the chassis centered and level while the individual wheels roll up and down over uneven terrain. Therefore the roll command can be used to travel large distances. However, other commands invoke much smaller moves, often changing the location of a body part by tens of centimeters or less.

Intuitively, walking with ATHLETE is a matter of choosing the order in which to step the feet. Experience commanding ATHLETE quickly makes it clear that it is much more about maintaining balance (keeping the center of gravity within the feet placements) and weight distribution. Therefore, much of the challenge with walking is shifting the chassis to appropriate locations between steps. There are also interesting trade-offs between stability and mobility - spreading ATHLETE's legs out makes it more stable but less able to move.

[2]These are not all independent in practice due to closed chains through the terrain.

## Architecture

Figure 5 shows the architecture overview of the proposed system. It includes the following features:

- A human operator specifies goals to the planners and approves or modifies any commands before they are sent to ATHLETE.
- There is one planner for each one of the layers outlined above. The operator can interact with any of the planners.
- ATHLETE is issued commands and returns state updates and new stereo images when requested.

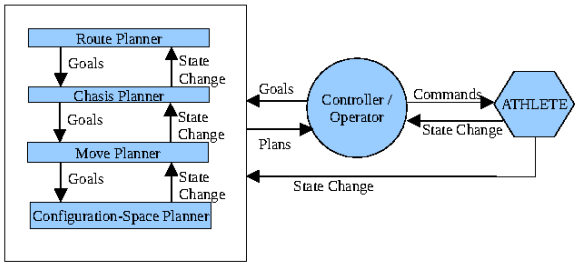


Figure 5: Architecture overview.

Although the Configuration Space planner outputs plans in the form of command sequences that could be directly executed by ATHLETE, final authority about what ATHLETE does resides with a human operator, assisted by a GUI-driven software controller. This allows the operator to increase the number of operations to be performed autonomously as the planning tools become more reliable. The operator will remain in the loop for the foreseeable future because of uncertainty at execution time and safety concerns.

A nice consequence of the decomposition into multiple planners is that the operator may interact with any of them to set goals and obtain plan information at that level. For instance, the operator could set waypoints for the route or chassis planners, or ask the configuration space planner for the configurations it is considering and choose one directly. This will allow for a more gradual adoption of the technology, compared to a black-box approach that can only take high level goals and outputs command-level plans.

The system builds upon EUROPA (Extensible Universal Remote Operations Planning Architecture),[3] a planning and scheduling library and toolset that has been applied to various NASA planning problems (Frank and Jonsson 2003) including satellite observation scheduling and solar array constraint management on the International Space Station (Reddy et al. 2008). EUROPA provides a plan database for plan representation that makes it easier to integrate the different planners as will be explained shortly. EUROPA also provides powerful modeling, constraint reasoning and debugging tools, and an extensible architecture so that new search strategies for the different planners can be plugged in when needed.

A significant challenge with this architecture is to coordinate the four planners effectively. We adopt a coordination architecture pioneered by IDEA (Finzi, Ingrand, and Muscettola 2004), and further refined in T-REX (McGann et al. 2008). From the coordination point of view, the different planners are equal peers, and communicate with each other using a common EUROPA database. A planner can post goals for another planner to refine, and can monitor state variables to determine whether it needs to re-plan, either because one of its posted goals failed to be satisfied, or because the state of the world changed in a way that is inconsistent with the current plan. This approach avoids any special-purpose coordination logic in the planning code, which would make the system more complex and potentially brittle. Each planner's model explicitly states when goals are to be posted, and also what happens when re-planning is triggered by a failed goal or a change in state variables. To avoid non-terminating cycles, we only allow a planner to post goals to, and to monitor state variables in, the lower-level planner immediately below it (see Figure 5), so the route planner can only post goals to and monitor the chassis planner, which in turn can only post goals to and monitor the move planner, and so on.

Figure 6 shows an example of the coordination.[4] The chassis planner contains a timeline to represent the position and orientation of ATHLETE's chassis over time. The *at* states on that timeline are represented in the EUROPA database as tokens (a EUROPA data structure that represents both actions and states). These *at* states generate *at* prerequisites on the frame timeline in the move planner. The move planner interprets these as goals; when they are posted it attempts to generate a plan to achieve them. The resulting *at* tokens for the frame will also impose constraints on tokens in the leg timelines; the move planner must determine appropriate steps, rolls, and frame shifts that fill in those timelines and satisfy all constraints.

This is how information is propagated from higher-level planners to lower-level ones. In the opposite direction, higher level planners monitor state changes in lower-level ones. For instance, when a goal fails to be satisfied in the move planner, that goal is removed from the frame timeline (and disallowed in future plans) which in turn causes an inconsistent state in the chassis timeline. Consequently, the chassis planner will re-plan and either post new goals for the move planner, or determine that its own goals are unachievable and remove them, thereby propagating the inconsistent state up to the path planner.

In a similar fashion, new telemetry coming in from ATHLETE is incorporated in the form of tokens on timelines owned by the configuration space planner (the lowest level planner), this new information may cause inconsistencies which will then trigger re-planning. To avoid read-write conflicts all accesses to the EUROPA database are serialized as described by McGann et al. (2008).

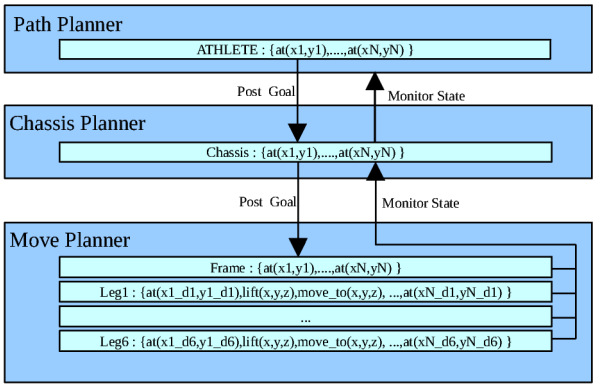


Figure 6: Example of planner coordination.

As we have seen, there are two ways to initiate a planning cycle:

- By posting goals to a planner, which may result in re-planning by the planner that owns the affected timeline and in derived sub-goals for lower level planners.
- By changing the state of a timeline, which may trigger re-planning in the planner that owns the timeline and potentially in higher level planners as well.

Re-planning is expected to happen often. Terrain data will be updated whenever the chassis is shifted and ATHLETE will quickly drift away from the exact location and configuration modeled in the plan, especially when rolling with active compliance. While ATHLETE's actual position after a move will almost always differ from the planned position, re-planning is only required when the planned position of the subsequent move can no longer be achieved, as determined by the configuration space planner.

Finally, during execution we expect ATHLETE to fail on some commands deemed acceptable by all planners. This could be a result of terrain that wasn't accurately modeled, or torque on some joint that exceeds safety limits. In these cases, the failure can be blamed on the current low-level command issued by the configuration space planner, and, in a similar fashion to the previous cases, the feedback loop coming from ATHLETE will update a state variable to cause an inconsistent state and trigger re-planning, which might propagate all the way up to the route planner.

We now consider each of the four planners in more detail.

[3] https://babelfish.arc.nasa.gov/trac/europa

[4] More detailed presentations are available elsewhere (Finzi, Ingrand, and Muscettola 2004; McGann et al. 2008).

## Route Planner

The goal of the high level route planner is to determine a reasonable 2D route through the terrain based on low resolution terrain data.

Usually, route planning simply minimizes distance traveled based on binary terrain data; points in the terrain are passable or not. For ATHLETE, however, there is a wide range of passable terrain. A route across flat terrain is much preferable to a route up and over a rocky hillside for many reasons (it is safer, faster to plan for, and much faster to execute), even though both are doable.

The route planner divides the terrain into a grid. Each square is roughly the size of ATHLETE and is assigned a cost that corresponds to the expected time to cross it, where squares likely to require difficult stepping are an order of magnitude more expensive than squares that require little or no stepping. The route planner then uses A* search where the heuristic is the Euclidean distance to the goal square.[5]

We don't expect this highest level plan to need much interaction with the lower level planners. In very complex and constrained terrain, the planner may have to settle on a route that turns out to be impassable when higher resolution data becomes available. However, because of ATHLETE's high mobility and the expected terrain it will face, we expect the best-looking route to almost always be traversable.

This planner may be extended to include waypoints, or to update the optimization criteria; for example, we might want to optimize cumulative expected reward while secondarily minimizing duration.

## Chassis planner

The chassis planner can be considered the high-level planner for the region within ATHLETE's field of view. Given a direction of travel determined by the high-level planner, the chassis planner assumes ATHLETE's nominal pose, which strikes a balance between stability and mobility, and creates a plan that combines translations and rotations of that fixed configuration to reach the given edge of the field of view, as shown in Figure 4.

The placement of the feet in ATHLETE's nominal position define six vertices of a hexagon. We want the easiest way to get those six vertices across the terrain. For example, consider a flat boulder-strewn area. Some boulders can be avoided completely. By using appropriate rotations, other boulders can pass between the feet without the need to step over them. Finally, where stepping cannot be avoided, we want to minimize the number of steps required. For example, Figure 7 shows one approach to a boulder requires four legs to step over it, while another orientation requires only the front and back legs to step over the boulder, a much easier operation.

To minimize stepping, the goal is to minimize the number of elevation changes that the six feet (in their fixed positions) must pass over during the translations and rotations in the plan. We again use A* search with Euclidean distance used for the heuristic cost estimate.

[5]D* search was considered but A* is fast enough for this domain.

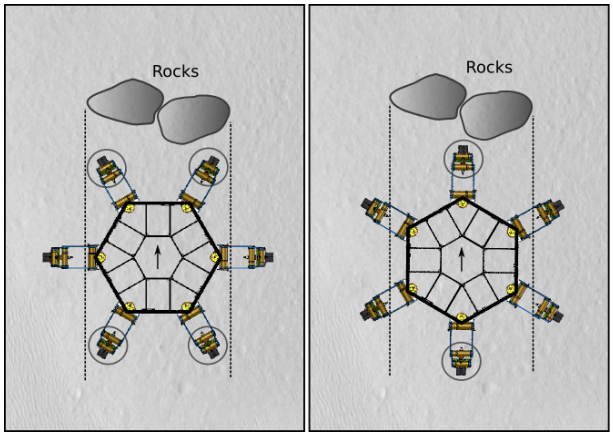


Figure 7: The chassis planner can minimize stepping by considering only the orientation and position of the chassis when approaching unavoidable boulders.

The cost for a translation or rotation is the sum, over all legs, of the distance traveled and total elevation change for the leg. Notice that distance, and possibly elevation change, will be non-zero even for rotations. We add additional penalties when:

- Adjacent feet have significant elevation changes at the same time. It is difficult to pick up two adjacent feet simultaneously while maintaining stability.
- Feet move into undesirable terrain (steep or slippery). These positions might be necessary, but are discouraged.

Finally, the algorithm must avoid large elevation changes in a couple of ways. First, the terrain between the six vertices must not be so high relative to those vertices that the chassis cannot avoid it. Second, the elevation difference between any pair of feet cannot exceed a certain amount so that all legs can simultaneously reach the ground.

Of the four planners, this is the one most likely to have horizon difficulties because we are planning right to the edge of the current field of view. However, we expect to replan each time the chassis moves; therefore replanning will occur well before the horizon is reached.

## Move Planner

The move planner is tasked with choosing a short sequence of the following moves that achieve the rotations and translations given by the chassis plan:

- **Roll:** Move the entire robot to a new position by lifting legs as necessary to avoid obstacles and rolling all feet still in contact with the ground.
- **Step:** Move a given foot to a new position on the terrain.
- **Shift chassis:** Move the chassis to a new position, leaving the feet where they are.

There is a fourth move, rotation, that will be part of the resulting sequence but is predetermined by the chassis plan.

Note that rolling could look very much like stepping. For example, a single roll might include holding up leg 1 to

avoid a rock, rolling 1 meter, putting leg 1 down, lifting leg 2 to avoid a second rock, and rolling for another meter. However, we only consider lifting legs straight up and down and can therefore easily determine the maximum rollable distance in a given direction without search as follows. As we move forward, if progress of any leg is impeded, put down any leg that won't be impeded if it is on the ground, and see if the impeded leg(s) can be lifted without losing stability.

Our search algorithm for this planner faces two decisions at each step: whether to roll, step, or shift, and the desired position. We first consider the latter.

For rolling, the desired position is simply the farthest position along the chassis plan which can be reached with rolling. Now suppose we want to take a step with a given foot and need to choose a new position. Figure 8 shows how we narrow this down to a handful of possible choices. First, given that the chassis is fixed in space, we have a region ($R$) of positions that can be physically reached by the foot. We only consider the subset ($S$) of those points that will result in a stable stance, where the robot won't tip over or exert too much force on any joint.[6] Within that subset, we focus on points that result in progress in the direction of the chassis plan ($D$). Finally, a subset of those points (the $X$s) corresponds to acceptable terrain (not too steep or slippery) and we prefer the position that gives us the most forward progress.

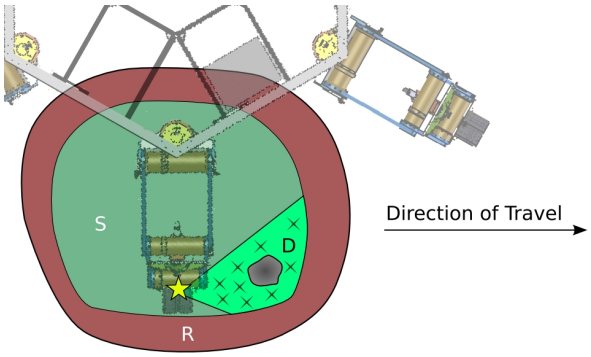


Figure 8: For a given foot, $R$ is all reachable points, $S$ is stable points, $D$ is desirable points (forward progress) and the $X$s correspond to points that are acceptable terrain.

Similar to Figure 8, the chassis has regions of reachable, stable, and desirable points. The only difference is that all desirable points are acceptable because terrain is not a factor.

Now that we have a process to select the new position for any given move, we must decide which move to make. Because rolling is fastest in practice, and is preferred by operators, we always roll if possible. When rolling is not possible, we simply select the move (step or shift) that results in the greatest progress along the chassis plan. Algorithm 1 shows how we determine a prioritized stack of moves with favored moves at the end (any roll gets first priority, followed by moves in the order of the distance they can move).

[6] A stable stance is a function of the terrain and the positions of the chassis and other legs.

**Algorithm 1**

**function** SELECT-GOOD-MOVES(*current-position*, *chassis-plan*)

```
1:  stack = {}
2:  for all move in f1, f2, f3, f4, f5, f6, chassis do
3:      X = {X's from Figure 8}
4:      for all (i = 0, i < p, i++) do
5:          select x ∈ X which is closest to the goal
6:          delta = distance from current position to x
7:          stack.push_back(move, delta)
8:          remove x from X
9:      end for
10: end for
11: sort stack with lowest delta at the front
12: d_r ← max rollable distance along chassis-plan
13: if d_r > 0 then
14:     stack.push_back(roll, d_r)
15: end if
16: return stack
```

Finally, we implement the move planner as a depth-first chronological backtracking algorithm where Algorithm 1 provides the set of possible branches for each search node. The set of possible moves returned by Algorithm 1 include at most a single roll and $p$ positions for each foot or the chassis, resulting in a branching factor of $7p + 1$. To limit the search space, we consider only a small subset of the $7p$ prioritized moves, and expect to plan for no more than 10 moves at once. Therefore, we expect the algorithm to be instantaneous in most terrain, and to complete within a handful of seconds in difficult terrain where the going will be slow regardless due to operator precautions.

We have not described how the regions of Figure 8 are actually computed. These require an understanding of both obstacles and the physical constraints and kinematics of ATHLETE. Fortunately, however, the required calculations are already performed by the low-level configuration planner described in the next section, and are fast to compute, so we can rely on calls to those low level libraries.

The move planner takes full advantage of EUROPA's modeling language, constraint engine, and search tools. We discretize the actions ATHLETE can take, but not the action parameters (where to step to, for example), as the custom algorithm can handle continuous state space.

## Configuration Space Planner

While we can directly execute rotation, chassis shift, and roll moves produced by the move planner, the configuration space planner is required to refine step moves. Those require a set of trajectories for each joint that ensures that overall motion of the robot avoids collisions, singularities, and joint limits. The kinematic chains of ATHLETE's legs create strong interdependence between the joints with respect to overall position.

The core difficulty is the need to plan in two different spaces at once: task space and configuration space. Task space refers to the normal world we are used to operating

in, and within which our goals, terrain, and obstacles are defined. Defining motion plans for the end point of a leg in task space makes it easy to avoid collisions, but may result in paths that are unreachable at some points, or which get too close to a singularity. A singularity is a location where, due to the particular alignment of the joints, near-infinite velocities may be required to move the end point in certain directions (e.g. the knee suddenly flips around by 180). Finally, while the path of the end point may be collision free, one must ensure that all the segments of the leg avoid collision.

To deal with many of these problems, one needs to also plan in configuration space, where each dimension represents the range of motion of one of the joints of the robot. Paths through this space represent feasible motion of the entire robot. Unfortunately, it is difficult to translate the terrain and other obstacles into this space efficiently – especially if the terrain data is being collected in real time by sensors (such as cameras). Thus, a common approach is to use a sample planner where a set of paths are defined in configuration space, often through a random process, and then samples along those paths are checked for collisions. Collision checking is done by taking the joint values at a point along the trajectory and applying them to a model of the robot which can than be evaluated for task space collisions with the environment or itself. Paths through configuration space are generated and tested until a route to the goal is achieved.

Configuration space planning for ATHLETE is computationally expensive yet must be tightly interleaved with execution. This is achievable because 1) we only plan for specific moves, thereby requiring only a subset of the 36 dimensional space and 2) the uncertainty (at this level of resolution) of future states means there is little reason to make plans beyond a few steps.

## Related Work

There are many applications where it has been useful to break planning into multiple levels. In fact, it could be argued that most robotics is really done this way, with low-level controllers doing something intelligent while higher-level pieces plan for the bigger picture. Our high-level route planning has simplified away all of ATHLETE's joints and instead classifies terrain traversability based on characteristics of the terrain itself. This approach was used in the Morphin software (Singh et al. 2000) to find paths for planetary rovers through natural terrain. We also currently use a derivative of that software in our stand-alone single footfall planning system. An extension to Morphin created at JPL called GESTALT (Goldberg, Maimone, and Matthies 2002) is currently being used by the MER rovers on Mars as a means to plan local traversability (Maimone, Leger, and Biesiadecki 2007).

The control and planning for legged walking robots is an extremely active field where research can be grouped into two categories: dynamically controlled robots, and statically stable robots. The majority of current research efforts are focused on dynamically controlled robots (for example, Campbell and Buehler (2003) and Kimura, Fukuoka, and Cohen (2007)). 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 (Playter, Buehler, and Raibert 2006) 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, etc. (Poulakakis, Smith, and Buehler 2005).

Unfortunately, dynamic control of a robot results in an inability to predict the results of motion commands on the robot, as it will reactively respond to the environment. This is generally unacceptable for planetary robotics where the expense of the missions, the inability to rescue the robot, and the intermittent communications require motion plans to be executed with precision and the final state of the robot to be predictable. Thus ATHLETE is designed as a statically stable walking robot (Kar 2003). Many such robots exist, the most famous of which is probably Dante II (Wettergreen 1995), which descended into the mouth of an active volcano.

Statically stable walking robots require explicit planning for leg positions. Most statically stable robots implemented in the past have been much simpler mechanisms, usually having three or four degrees of freedom per leg. ATHLETE, with its 36 degrees of freedom, is challenging to plan for, and requires the use of advanced motion planning algorithms such as Sample Based Planners (Lavalle 2006).

## Future Work

Implementation of the proposed system is not complete. The configuration space planner has been implemented and tested with ATHLETE by operators. Preliminary versions of the move and route planners have been implemented and tested on simulated data. The chassis planner has not been implemented yet, and the planners run separately and have not yet been combined into the described hierarchical system.

In additional to facilitating initial development, the proposed architecture is intended to make future extensions simple and make it easy to experiment with different algorithms and heuristics at each level. For example, route planning could easily incorporate other goals (e.g. science or mapping goals) as has been done for other rover projects. Also, we may want to optimize plans for various metrics (e.g. energy, duration, or minimal number of commands). Or, we might want to experiment with machine-learning approaches for the move planner, or add specific pre-defined gaits for crossing benign terrain (sandy terrain that is flat but requires stepping, for example).

As soon as users are prepared to allow some automation, tight integration of control and execution will be crucial, both because of the high-level of uncertainty regarding the terrain, and because ATHLETE often gets into unexpected configuration space singularities. We believe that combining the four planners will make replanning quick and easy.

In practice, ATHLETE also regularly hits configurations where the motors are unwilling to execute a seemingly reasonable command. Usually, this situation is due to force sensors drifting over time and can be solved with a command that picks up and puts down each leg in turn to recalibrate

the sensors. We expect to incorporate this recalibration as an option when replanning due to a failed command.

There are also blindspots due to occlusions from rocks and the legs themselves. In difficult terrain, it may be important to fill in some blindspot information with terrain data before proceeding. In extreme cases, a plan could involve raising a leg high above the ground to get images of unknown terrain from the foot cameras.

Finally, we may want to incorporate image taking actions into the plans. It takes time to acquire and process new images, and the planners could help optimize how often images are taken and how much of the field of view is included.

## Conclusion

We have described a way to combine four different planners, each operating at a different level of granularity, to automate the mobility of the six-legged ATHLETE robots. First, the route planner uses low-resolution terrain data to produce a rough route to a distant goal. Then, within ATHLETE's field of view, the chassis planner refines that plan in an attempt to maximize rolling and simplify stepping. This plan is further refined by a move planner into a sequence of steps, rolls, and chassis shifts. Finally, a low level configuration planner fills in the details of the steps based on the kinematics of ATHLETE.

Another way to think about the different levels is with respect to the model of ATHLETE used. The route planner models ATHLETE as a single point. The chassis and move planners model it as 7 points (6 feet and the chassis); for the chassis planner, the positions of the 7 points are fixed relative to one another while they can each move independently in the move planner. Finally, the configuration space planner models all 36 degrees of kinematic freedom but only has to solve problems involving a subset of configurations space.

In our architecture, different plan components are determined by different planners; the chassis planner chooses rotations, the move planner adds rolls and chassis shifts, and the configuration space planner figures out a sequence of commands to achieve each step. This approach allows the planners to nicely mirror the multiple granularities of execution available on the robot.

Using multiple levels, which combine various algorithms from both the AI and robotics communities, has several other advantages. First, it nicely handles the fact that 1) terrain knowledge decreases quickly with distance and 2) uncertainty regarding ATHLETE's exact location and configuration increases quickly after the first few moves in a plan. Second, using more abstract models to solve the longer horizon problems means that all planners can be expected to be fast, simplifying the ability to replan. Finally, from a practical standpoint, it allows the various planners to be developed simultaneously but separately while knowledge about kinematics and physics is not needed in the three higher level planners.

Recall that ATHLETE is currently operated manually using it's command dictionary. The goal of this project is to speed up that process by suggesting commands to the user. We expect the adoption of automation to be an incremental process. However, even when only the lowest level commands are fully automated, having the multiple levels of planning available is important both so that the next few suggested commands lead towards a reasonable solution, and so that the operator can see why those particular commands were chosen.

## References


Campbell, D., and Buehler, M. 2003. Preliminary bounding experiments in a dynamic hexapod. In Siciliano, B., and Dario, P., eds., *Experimental Robotics VIII*. Springer-Verlag. 612–621.

Finzi, A.; Ingrand, F.; and Muscettola, N. 2004. Model-based executive control through reactive planning for autonomous rovers. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*.

Frank, J., and Jonsson, A. 2003. Constraint-based attributes and interval planning. *J. Constraints* 8.

Goldberg, S.; Maimone, M.; and Matthies, L. 2002. Stereo vision and rover navigation software for planetary exploration. In *IEEE Aerospace Conference*, volume 5.

Hauser, K.; Bretl, T.; and Latombe, J.-C. 2006. Motion planning for a six-legged lunar robot. In *Workshop on the Algorithmic Foundations of Robotics*.

Heverly, M., and Matthews, J. 2008. A wheel-on-limb rover for lunar operation. In *Proc. of the Ninth Intl. Symp. on AI, Robotics, and Automation in Space (iSAIRAS-08)*.

Kar, D. 2003. Design of statically stable walking robot: A review. *Journal of Robotic Systems* 20(11):671–686.

Kimura, H.; Fukuoka, Y.; and Cohen, A. H. 2007. Adaptive dynamic walking of a quadruped robot on natural ground based on biological concepts. *Int. J. Rob. Res.* 26(5):475–490.

Lavalle, S. 2006. *Sampling-Based Motion Planning*. Cambridge University Press.

Maimone, M.; Leger, C.; and Biesiadecki, J. 2007. Overview of the mars exploration rovers' autonomous mobility and vision capabilities. In *IEEE Int. Conf. on Robotics and Automation Space Robotics Workshop*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008. A deliberative architecture for AUV control. In *Int. Conf. on Robotics and Automation (to appear)*.

Playter, R.; Buehler, M.; and Raibert, M. 2006. Bigdog. In *SPIE Defense and Security Symposium, Unmanned Systems Technology*.

Poulakakis, J.; Smith, A.; and Buehler, M. 2005. Modeling and experiments of untethered quadrupedal running with a bounding gait: The scout II robot. 24(4):239–256.

Reddy, S. Y.; Iatauro, M. J.; Kurklu, E.; Boyce, M. E.; Frank, J. D.; and K.Jnsson, A. 2008. Planning and monitoring solar array operations on the ISS. *(under review)*.

Singh, S.; Simmons, R.; Smith, T.; Stentz, A.; Verma, V.; Yahja, A.; and Schwehr, K. 2000. Recent progress in local and global traversability for planetary rovers. In *IEEE International Conference on Robotics and Automation*.

Wettergreen, D. 1995. *Robotic Walking on Natural Terrain: Gait planning and behavior-based control for statically-stable walking robots*. Ph.D. Dissertation, Carnegie Mellon University.

Wilcox, B. H.; Litwin, T.; Biesiadecki, J.; Matthews, J.; Heverly, M.; Morrison, J.; Townsend, J.; Ahmad, N.; Sirota, A.; and Cooper, B. 2007. ATHLETE: A cargo handling and manipulation robot for the moon. *Journal of Field Robotics* 24(5):421–434.