Motion planning of legged vehicles in an unstructured environment

Craig Eldershaw
Oxford University Computing Lab
Wolfson Building, Parks Rd
Oxford, OX1 3QD, United Kingdom.

Mark Yim
XEROX Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto, CA, 94304, United States.

Abstract

A planner for statically-stable motion of a legged robotic vehicle over an uneven terrain is presented that can plan the footplacement of individual legs for highly cluttered terrain. A method for determining the traversability over a generic discretised height map terrain is presented. Planning is broken into two levels of refinement to reduce the overall complexity and incorporates a number of heuristics. The planner has successfully planned the motion of 6 and 8 legged configurations of the XEROX PARC PolyBot modular reconfigurable robot as well as the CMU Ambler in simulation over arbitrarily complex terrain. A distributed implementation of the planner has also been shown on PolyBot's distributed computational platform.

1 Introduction

The algorithm discussed in this paper deals with a legged vehicle moving over unstructured terrain. The types of terrain supported range from: relatively uncluttered environments such as predominantly smooth terrain with large obstacles, through to environments where specific foot placements are more rare among a sea of obstacles—like lily pads in a pond. The inputs to the algorithm are a surface map (giving height data at all points on the surface), a description of the robot, and the required starting and finishing locations of the vehicle. The output is a step-by-step route consisting of the vehicle's foot locations.

With a legged robot, each leg will usually involve several joints. When all the joints of all the legs in the vehicle combined with the pose of the vehicle itself are considered, then the total dimensionality of the problem can grow to be large (e.g > 10). As complete planning algorithms are generally at least exponential in the number of degrees of freedom [1], then it is clearly undesirable to consider all these degrees of freedom at once. The method proposed is to initially reduce the representation of the robot to just three dimensions: the location (cg_x, cg_y) and orientation (cg_θ) of the robot's centre of gravity (CG) through a processed map. The second stage extends this three-dimensional route to the detailed leg motions.

1.1 Unstructured environments

Most motion planners have dealt with structured environments with easily identified obstacles, prohibited regions of the environment into which no portion of the vehicle must enter. The environment that this paper is concerned with has no regions which are prohibited *per se* and no distinct obstacles. Rather, the entire environment forms an obstacle in varying degrees. The environment is specified by a two-dimensional continuous surface defined over a square region such as is shown in Figure 1.

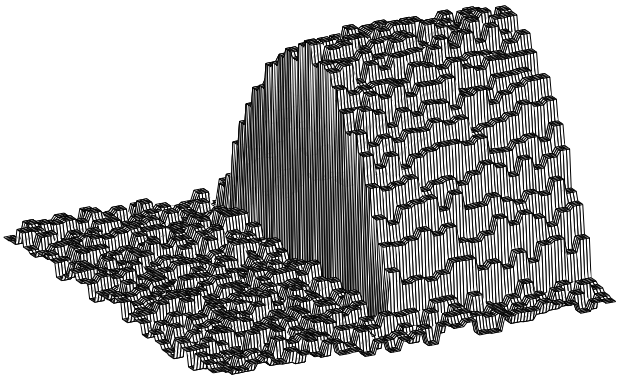

Figure 1: *A sample unstructured environment.*

The height-map shown in Figure 1 is an interesting one. The broken nature of the ground is enough to immediately defeat any wheeled locomotion unless the radius of the wheel/track is large relative to the size of the environment's bumps. A legged vehicle may be able to traverse some regions if using an open loop gait—simply thrashing its legs enough may cause forward (albeit inefficient) motion. However this would be defeated by the "cliff" which cannot be solved by anything short of a path planner. A legged vehicle has had its route successfully planned across this environment by the algorithm presented in this paper.

1.2 Target architecture

While the overall algorithm presented here can be run on many platforms and can solve a variety of problems, it was specifically intended for use on the PolyBot platform [2]. This robot is a modular reconfigurable robot, and so can form many shapes. For example different configurations

will have different numbers and lengths of legs, or may vary in the legs' points of attachment to the body. In order to plan for the locomotion of this robot, a generalised planner is needed that can support arbitrary legged robot configurations. The details of PolyBot will be discussed in Section 4, but one relevant characteristic is that the platform provides a large number of medium-speed processors with limited distributed memory. The algorithm must be capable of running on this architecture.

For a given configuration of PolyBot or an arbitrary vehicle, shape and capabilities must be given to the system. The relevant capabilities are the range and reach of each of the legs. The shape is an inverse height map of the robot's under-carriage (other aspects of the robot's body shape are irrelevant as only the under-carriage and legs interact with the environment).

2 Previous legged motion work

Some work connected with the issues discussed above has previously been done, but none really address all the issues in a fully integrated manner. There are three categories into which most existing methods fall: Gaits for smooth ground, broken ground foot planners, and higher level legged vehicle planners.

Gaits for smooth-ground A gait is a pattern of leg movements whose final state is identical to the initial one. Some of the early contributions include [3] where *wave gaits* were first formalised. [4] provides another theoretical treatment of gait construction for symmetric vehicles for locomotion at some angle (the *crab-angle*). Unusually for gait planners, this one also takes into account the overall slope of the terrain.

Foot-planners for broken ground Probably the most generic and widest adopted of the foot-planners capable of negotiating rough terrain is the *free-gait*. Initially proposed in [5], this has since been elaborated upon by other authors, in particular [6]. Given that the vehicle is moving along a set path, the ordering of when to move a foot is determined by a stability measure. An early walking robot is the Ambler [7]: a six-legged, self-contained robot that used a modified free-gait[8, 9]. Operators would give the Ambler its overall path as a series of obstacle-free arcs which it would trace out. The vehicle's own control system would adjust its height so as to keep the body level and the under-carriage's clearance from the ground within a given window.

Higher-level planners The planners in this category are higher-level ones, but unlike general motion planners, these ones have been designed for unstructured terrains with the legged locomotion aspect in mind.

As with some of the methods described previously, [10] assumes that a height map of the terrain to be crossed has been acquired somehow (a scanning laser is one obvious method). Each cell is classified as safe or unsafe. A somewhat crude planner then navigates its way through the new map ensuring that the passage between unsafe cells is greater than the vehicle width.

In [11] a comprehensive planner (supporting both high- and low-level planning) is presented. This system plans the complete motion of a spider robot (four or more legs) and a point body across a flat surface marked with valid/invalid footholds. It finds a path that the robot's body may follow that has valid footholds. It then constructs a sequence of foot-steps.

2.1 Shortcomings

It can be seen from the above review of existing planners that several deficiencies exist. Perhaps the most significant of these (as it effects *all* the planners) is a failure to support the reach of different legs. Other problems which effect some of the surveyed planners include: assuming the ground to be smooth; supporting only a fixed number of legs or a specific configuration; having overly constrained or completely unconstrained leg ranges; and lacking integration between high- and low-level planners.

A distinction is made from here-on between *reach* and *range* of a leg. Range is the region of the xy plane into which the foot can feasibly be placed (ie. constraints on how far forward/backwards/left/right the foot can go). Reach is how far down the foot can go (this need not be uniform across the entire range).

Concerning reach of legs: those papers describing the above planners that do consider it at all, blithely state that the height of the foot placement can be simply determined by the input height map. This is indeed true: having determined the (x_i, y_i) of the ith foot, then the height of its placement simply becomes $z_i = f(x_i, y_i)$ where f is the two dimensional height map. However this ignores the important fact that the height of the environment combined with the reach of the leg should in fact form a part of the feasibility test.

An example of this was shown in Figure 1. Even if the ground was smoothed out (ie. high-frequency noise removed), then all of the planners discussed in this section would fail by hitting the cliff. However a closer looks shows that nothing really distinguishes the cliff from the smaller bumps—except in scale. Further consideration shows that the reason the cliff will withstand a direct frontal assault whereas smaller terrain features succumb is because the legs of the robot have sufficient reach to step up onto (or even step over) smaller features, but not span the full height of the cliff.

That is, there is a feasible stable stance at the base of the cliff (with all feet placed at the base) and another at the top of the cliff (with all feet on the high-ground). Due to the limitation of reach of the legs, there is no stable stance for the robot with some legs at the base and some on the high-ground. If the entire raised region is treated as an obstacle, then that high ground could never be reached, even though a valid route exists via the sides.

The INRIA planner[11] which is the most interesting in many regards has the very significant limitation that it assumes all the legs have the same leg-range: that of a circle inscribed about the robot's body. Unfortunately this assumption is fundamental to the way in which the algorithm works,

In this paper, we assume that a perception and control system similar to that of the Ambler exists.

3 Splitting the planning work

To reduce the complexity of the planning, the movement plan for the robot is decomposed into two stages. The first of these simply finds the overall path the robot will take—the path the robot's CG will follow. This path will avoid crossing cliffs or ground which is broken beyond traversal, while guaranteeing that a stable set of foot placements exists. The second stage searches to find a set of valid foot movements which can allow the robot to execute this path. It should be noted that this decoupling does not rely upon the assumption (made by many other authors) that the second stage of planning is trivial due to the terrain being smooth.

3.1 High level path: PRM/CGspace

By initially ignoring the details of the legs, the problem's dimensionality is reduced to three: (cg_x, cg_y, cg_θ). The roll and pitch of the robot is assumed to be zero, and decisions about the final height of the robot are relegated to the lower-level planner. The (cg_x, cg_y, cg_θ) are discretised to form *CGspace*, a three-dimensional bitmap. The discretisation is such that the resolution in the cg_x and cg_y axes matches the size of a the smallest foot print from an individual leg. This allows a coarse resolution to speed computation and minimise memory requirements while not allowing a foot to fall in an unrepresented hole (one that is smaller than the resolution of the cell). The cg_θ axis may be discretised to any required resolution, but typically either one (ie. no rotation allowed) or at least eight (non-trivial lower values would require the vehicle to turn through an infeasibly large angle in a single movement).

This stage of the algorithm uses a version of the probabilistic road-map planner (PRM). The PRM planner (see [12]) is a graph-based one. It fundamentally works by scattering points uniformly throughout the CGspace and applying a cheap local planner to connect those points which are near neighbours. An undirected graph is built up: one node for each of the problem's start and finish points and one for each randomly added point; if the local planner successfully connects two of these points, then an edge is added in the graph connecting them. Local paths between pairs of points are not generally retained in detail due to storage restrictions, but they can be recreated cheaply later if required. When a graph search from the start node through to the finish node succeeds, then the solution may be found by re-calculating the path for each relevant edge and concatenating them to form the complete path.

The local planner employed in this case is an $A^\star$ search of the CGspace bitmap. Each *CGcell* (a cell within this map) is marked as valid if it satisfies a set of conditions, or invalid if it fails any of them. These conditions ensure that some feasible, collision-free leg configuration exists which will allow the vehicle to stand in a stable manner with its CG in that position and orientation.

Some foot positions are infeasible in their own right. This is decided by the slope of the terrain at that point. Part of the robot's description is the maximum inclination at which its feet can maintain traction. In theory, walking on slopes greater than this angle may be possible if an opposing slope is nearby which the vehicle can brace itself against (in climbers parlance "chimneying"). However this possibility would require the legs to possess significant lateral strength and would give rise to some rather complicated force-based calculations; so this possibility is ignored in this paper. The quality of a foot placement cell could incorporate properties other than slope [9] for example stability (whether sand or loose rock is detected by sensors), but these are not considered here.

In the target application, statically stable motion is desired. To ensure this, one of the necessary CGcell conditions is that a feasible foot position be found in each of the four quadrants centred about the vehicle's CG. See Figure 2 below for an example. Each of these foot positions must be chosen to be within some leg's region of motion (within reach). The reader will recognise that this is an overly strong constraint: stability requires only three points of contact. However as well as being faster to calculate, using the "quadrants method" allows a more general statement about stability (any foot in that quadrant is sufficient rather than a specific combination of three feet).

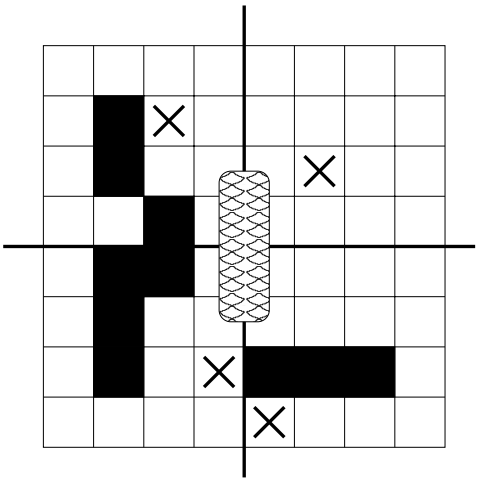

Figure 2: *The robot's range of motion is broken into quadrants. At least one valid position (shown as a cross) must be found in each. Filled squares are infeasible foot locations.*

The final condition tested is based upon each leg's vertical reach. The purpose of this is two-fold: one aim is to ensure that the legs are not placed too far apart ver-

tically (eg. two feet on top of a cliff, two at the cliff's base); the other is to ensure that the legs are long enough to hold the body off the ground (eg. feet straddling a high narrow ridge). The minimum height the robot must be at is $MinHeight = \max_{i,j}[land(i,j) + robot(i,j)]$ where i and j are coordinates in the vehicle's frame of reference; $land(i,j)$ is the height of the environment at (i,j); and $robot(i,j)$ gives *depth* of the vehicle's under-carriage below the vehicle (relative to an arbitrary horizontal zero plane running through the vehicle, from which all leg and under-carriage distances are measured) for the current CG-cell. Therefore it is required that for each of the four foot positions chosen, the height at that point plus the reach of the relevant leg (below the vehicle's zero plane) is greater than $MinHeight$. If not, then the robot's under-carriage would be in contact with the ground.

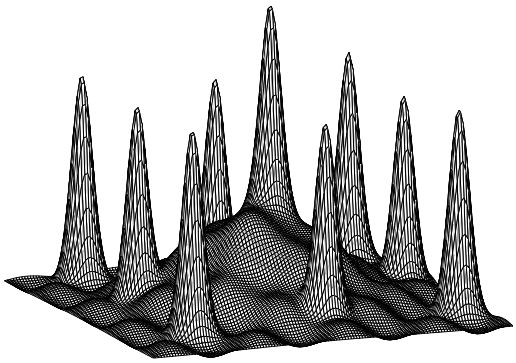

Figure 3: *A computer generated test environment.*

With these test conditions now defined, fragments of the CGspace can be materialised upon demand. Figure 3 shows a sample computer-generated terrain. The nine sharp peaks are clearly too steep for useful traction, so too are parts of the slope up to the central plateau. Those cells where feet are unable to obtain traction are shown as small circles in Figure 4. Black cells correspond to invalid CG-cells for a particular eight-legged spider. It should be noted that CGmaps are dependent upon not just the environment, but also the particular vehicle. This can be seen by contrasting Figure 4 with Figure 6, both of which are the same environment, but have been calculated for different vehicles. It can be seen that it is quite possible to have invalid foot positions which are not invalid CGcells as the vehicle can "straddle" these places. Figure 6 shows this clearly: the invalid foot region in the upper right has no corresponding invalid CGcells at all. The contrary is also true: the "black border" which can be seen around large regions of invalid foot positions occurs because the vehicle can find no support on one side.

The $A^\star$ search is performed on a block of CGspace slightly larger in the cg_x and cg_y directions than the minimum block defined by the points to be connected. The block searched always contains the full cg_θ dimension. The search is based upon 6-adjacency rather than 26-adjacency (ie. just single unit changes in at most one of the search axes at each movement). The search is permitted to wrap at the 2π boundary in the cg_θ dimension.

To reduce the number of calls to the $A^\star$ search, the PRM method was modified to generate only a series of trees (as distinct from more general graphs). Rather than attempt to join each newly selected point to *all* neighbours which are sufficiently close, such attempts are only performed if the new point is not already connected (possibly indirectly) to the existing point in question. This greatly reduces the number of planner calls ($O(n)$ rather than $O(n^2)$) and does not effect the graph's overall connectivity. The negative aspect of this is that the routes generated by the final graph search may take a considerably less direct route than would have occurred if the algorithm had attempted to connect all near-neighbours points.

To avoid all graph searches and reduce the number of distance calculations (which are used to determine if an attempt should be made to connect two points), each distinct tree is stored as a separate linked list. Once a path is found connecting a new point to one tree, it is added to that list and never compared against others in that list. If a new point is successfully connected to two separate trees, then those lists are concatenated. When the list containing the starting point and the list containing the finish point are eventually merged, then a solution is found.

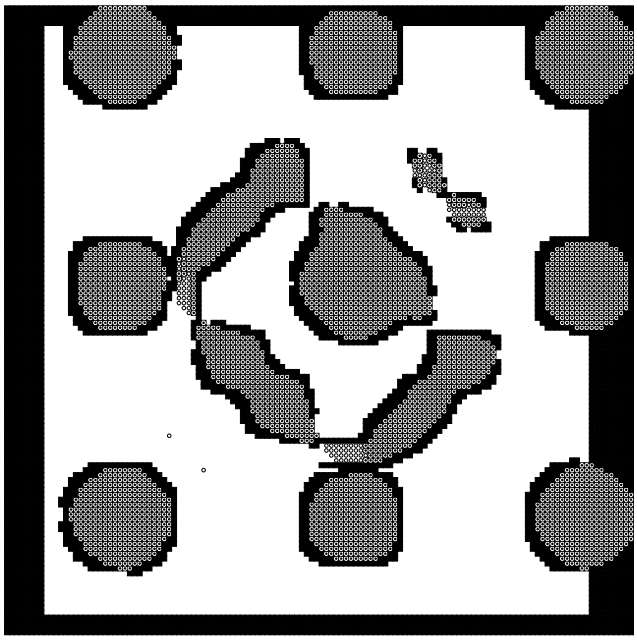

Figure 4: *The footmap and CGmap (at $CG_\theta = 0$) of the environment shown in Figure 3 for a eight-legged spider. Invalid foot positions are circles, invalid CGcells are black squares.*

3.2 Foot-level planner

The result of the high-level planner is a contiguous path of feasible CGcells. These describe the desired motion of the entire robot from the given start point to the given finish point. The second part of the algorithm finds a set of leg motions which will cause the vehicle to move along this given path from one CG position to the next.

Since (as stated previously) the desired motion is a statically stable one, all the conditions defined in the previous section for CGcell validity must still be enforced. The foot planning is fundamentally a recursive depth first search of a decision tree. At each node of the tree a decision is made to either:

- Move the vehicle's CG from the current CGcell to the next. That is, keep all the feet on the ground where they are, but adjust the leg joints so that the vehicle "leans forward" in the direction of motion. Since the CG is three-dimensional, then the "forward" may in fact be a rotation of the vehicle's body.
- Keep the CG where it is, but move one of the feet. This can take two forms: it can just pick up the foot (and keep it up) or it can move the foot to a new location. There are three reasons why this would be done:
  - because the next CG movement will bring the foot out of range
  - to avoid instability in the next CG move (a foot is about to leave a quadrant)
  - to free-up another foot which needs to move, but is currently unable to move on stability grounds.

It should be noted that (in general) at any stage there are many ways of choosing to move a leg. Firstly there is the choice of which leg to move, and then for any given leg there will generally be many different places to put it. Many of these can be ruled out due to: one foot stepping on another; the vehicle becoming unstable; a position being out of reach for that leg; or the slope being too great for traction. However that still often leaves a large number of possibilities. So the fan-out for this tree (the number of children of each node in the tree) can typically be of the order 10–100. The depth of the tree is at least as long as the number of CGcells in the CG path found earlier. That can only be achieved if no actual foot movements were necessary—which is not true in practice—so the tree can easily have a depth of hundreds. Given these (admittedly vague) figures, the number of nodes in the search tree can easily reach 10^{10}. In more complex problems the number is correspondingly higher (of the order 10^{100}–10^{1000}).

Given the size of the tree, a brute-force search is quite infeasible. The algorithm uses a series of heuristic techniques to guide the search. This works by prioritising each move at each point in the tree. A "free foot" is one which is not currently (and not immediately about to become) critical for stability. In descending order of precedence, these moves are:

1. Move the CG forward while keeping feet in place.
2. If moving the CG forward is feasible for each of the legs individually (ie. all remain in range), but would cause instability, then attempt to move a free foot into the deficient quadrant.
3. Move any free foot that is at the limit of its range and about to become infeasible to a new location.
4. Raise up (and keep up) any free foot that is at the limit of its range and is about to become infeasible, but has nowhere feasible to go.
5. If a foot that isn't free is about to become infeasible, move a free foot into the relevant quadrant to "free" the ailing one.

In addition to the above ordering of moves, when moving a foot (for any of 2, 3 or 5 above) then the possible new placements are also ordered. The ordering is such that if motion of the vehicle's CG were to continue in its current direction, then the best new foot placement would be one that remains within the feasible region of that foot for as many CG movements as possible.

One nice aspect of using a heuristic ordering to the searching of the tree is that even if the heuristically inspired "suggestions" are completely wrong, then eventually the less favoured (but correct) moves will be tried. So the overall completeness of the foot-level planner is not affected by the heuristics, but the average search time is reduced by many orders of magnitude.

3.3 From low back to high

Unfortunately the CG path returned by the high-level planner may not actually be statically traversable. Each valid cell in the path guarantees that there is a stable configuration allowing the vehicle to stand there. However it does not guarantee that there is a means of moving from any given stance in one CG position to a stable stance in the next. It is possible that only a dynamic move (one where stability is temporarily sacrificed) can achieve. It is also possible for two neighbouring CGcells to have a stable and collision-free stance but each cell may have heights too far from each other for the legs to reach and make the transition between cells.

These situations cannot be detected in the high-level planner. To do so would require all the knowledge (and work) that was deliberately separated out into the low-level planner. So it is the responsibility of the low-level planner to *heuristically* detect these and pass this information back to the high-level planner. Heuristics are required since for the low-level planner to do a *complete* test would be infeasible, as it would require exhaustively planning all possible steps from the start of the path to this point. The high-level planner can mark the offending CGcell as invalid and resume the planning. In many cases the low-level planning done so far can be usefully re-used as the newly corrected CGpath may only contain minor deviations from the previous one. This heuristic time-out and passing back of the information has not yet been implemented, however in the authors' experience, these situations have not occurred frequently.

4 Implementation

The overall algorithm presented here is quite general and so can be run on many computational platforms for use with vehicles of many different shapes and capabilities. However it was built for a specific problem. The plan generated by the algorithm is intended to be used by PolyBot. PolyBot [2] is a modular reconfigurable robot designed and built at XEROX PARC. It is capable of reconfiguring into many different topological structures—including legged vehicles (which is when this algorithm is to be employed). Also influencing the design was the eventual desire that the algorithm be *run* on the PolyBot's own CPU's.

4.1 Architecture

There are several significant aspects of the PolyBot's hardware system which should be considered: the memory size and the distributed nature of the system. PolyBot consists of a series of modules (currently twenty-four, but anticipated soon to be several hundred); each is identical and contains: a single Motorola 555 CPU (see [13]), 300kb of programmable flash and 500kb of RAM. The packet-based communications between modules is over a shared serial bus with a maximum total throughput of about 500kbps.

4.2 Distributed processing—the CG planner

While a reasonable amount of *total* computation (52K Dhrystones (v2.1) [14] times the number of modules) and memory (500kb times the number of modules) exists, it is fragmented into fairly small portions connected by a relatively slow communications link. This is one of the significant reasons for maintaining the PRM layer of the algorithm. For use on a large compute server, the $A^\star$ searching of the CGspace is quite cheap. On such a platform, the PRM step could reasonably be replaced by a single call to the $A^\star$ between the starting and finishing CGcells.

While the PRM layer is not really assisting in performance on simpler problems, on the target architecture the use of it has quite a different purpose. Each of the PRM's calls to the local planner are fairly independent. This means that the $A^\star$ searches that are to be performed between different pairs of nearby cells can be distributed and run simultaneously on the PolyBot's many processors. This allows an easy way of parallelising part of the computation.

More pressing than computational power though, is memory. The complete materialised CGspace for larger problems involving rotation requires a lot of memory—in general far more than any one module could store. However the authors' implementation never uses the entire CGspace at once—only a small region containing the two near-by cells which are to be joined. This means that at any given time, each processor need only store a small fragment of the whole, thus solving the memory problems.

Repeatedly materialising the same fragment of CGspace is expensive and due to the communications system available, passing large sections of calculated CGspace between different modules would be prohibitive. The solution is to ensure that each processor is assigned to process $A^\star$ search queries in only one region of the CGspace. The regions being sized so as to completely fit in the memory of a single module.

Using this technique, the planner was implemented on 5 PolyBot modules with a simulated environment passed to the modules from a host PC. Five modules roughly decreased the planning time by half versus one module for the case where the environment was small enough to fit in one module.

5 Examples

As mentioned in Section 2, one well known legged robot which did do step by step planning was the Ambler [7]. This is a six legged robot with two sets of stacked orthogonal legs, making it capable of a unique circular gait. The leg's range of motion is shown in Figure 5.

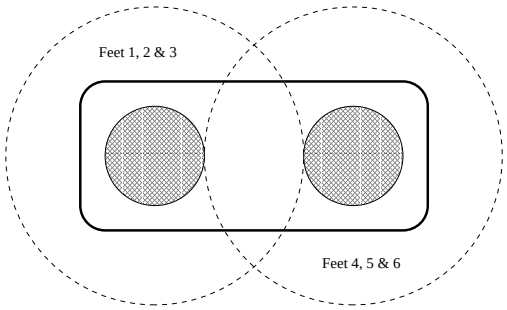

Figure 5: *The ranges of motion of the Ambler's six legs.*

The Ambler's configuration is well suited for this algorithm. Figure 6 shows the outcome of running the algorithm. Once again, invalid CGcells are marked with black blocks, and small circles indicate invalid foot regions. The considerably longer reach of the Ambler versus that of the spider robot (used in Figure 4) can be clearly seen. For example the invalid foot region in the top right is completely traversable by the Ambler (no black CGcells).

The continuous black line shows the path found by the high-level planner for the Ambler's CG to take. Snapshots of the robot are shown at various stages along the route. The crosses are the "footprints" left by the robot as it executed the full path.

6 Conclusion and Future work

This paper has presented a system which is unique in that it performs both high-level planning and step by step planning in an unstructured environment taking into account leg reachable workspaces, terrain heights and the exact shape of the robot. It is also noteworthy for being a generalised planner supporting any legged vehicle configuration. Literature concerning unstructured environments is rare due to the difficulty of precisely defining the obstacles—this aspect has been addressed by the authors. Most legged vehicles use simple gaits for locomotion and rely upon mechan-

ical flexibility to overcome minor obstacles, this algorithm uses high level planning to avoid unsurmountable obstacles, and careful foot-placement to navigate through other regions.

While the C-space of a legged vehicle is usually very large (and so expensive to create and search), this paper explains how to decouple the problem into trajectory planning and foot planning, thus reducing the dimensionality. This does introduce problems in some environments as discussed in Section 3.3, however the authors have proposed one solution, and intend to investigate this further. Even with this decomposition, the search space is still infeasibly large to search completely, and so a heuristic strategy is proposed.

It is hoped to have the actual robot with 100+ modules planning and executing its own route in 2001.

Acknowledgements: This work is supported in part from DARPA contract MDA972-98-C-0009. Also, the work of Kimon Roufas and David Duff in building the PolyBot modules (version G2) and Ying Zhang in developing the communications software for these modules is deeply appreciated.

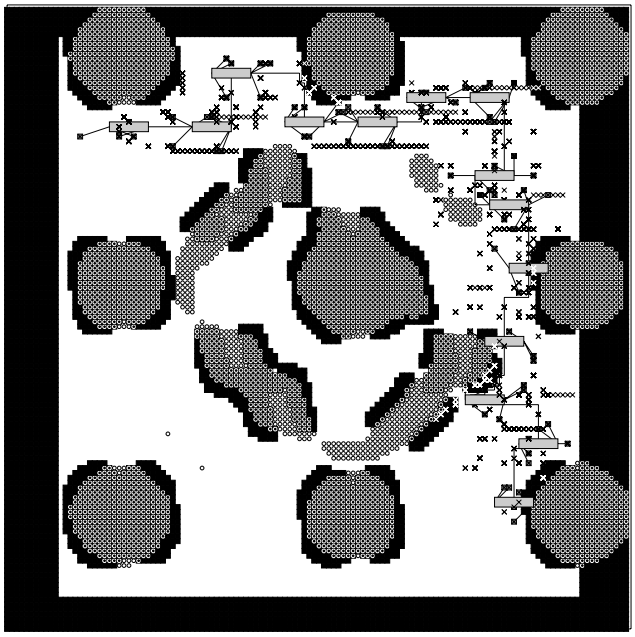

Figure 6: *The final path taken by the Ambler robot.*

References

[1] J. F. Canny, *The complexity of robot motion planning*. ACM Doctoral Dissertation Award: 1987, London: The MIT Press, 1988.

[2] M. Yim, D. G. Duff, and K. D. Roufas, "PolyBot: a modular reconfigurable robot," in *International Conference on Robotics and Automation*, (San Francisco, California, USA), IEEE, Apr. 2000. In press.

[3] R. B. McGhee and S.-S. Sun, "On the problem of selecting a gait for a legged vehicle," in *Transactions of the VI IFAC Symposium*, (Erevan), pp. 53–62, 1974.

[4] V. Kumar and K. J. Waldron, "Gait anakysis for walking machines for omnidirectional locomotion on uneven terrain," in *Seventh Symposium on Theory and Practice of Robots and Manipulators* (A. Morecki, G. Bianchi, and K. Kedzior, eds.), pp. 46–67, 1988.

[5] E. I. Kugushev and V. S. Jaroshevskij, "Problems of selecting a gait for an integrated locomotion robot," in *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, vol. 2, (Tbilisi, Georgia, USSR), pp. 789–793, Sept. 1975.

[6] R. B. McGhee and G. I. Iswandhi, "Adaptive locomotion of a multilegged robot over rough terrain," *Transaction on systems, man and cybernetics*, vol. 9, Apr. 1979.

[7] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, and W. R. L. Whittaker, "Ambler: An autonomous rover for planetary exploration," *IEEE Computer*, vol. 22, pp. 18–26, June 1989.

[8] D. Wettergreen, H. Thomas, and C. Thorpe, "Planning strategies for the ambler walking robot," in *International Conference on Systems Engineering*, IEEE, 1990.

[9] E. Krotkov and R. Simmons, "Perception, planning, and control for autonomous walking with the ambler planetary rover," *International Journal of Robotics Research*, vol. 15, pp. 155–180, April 1996.

[10] J. L. Oliver and F. Ozguner, "A navigation algorithm for an intelligent vehicle with a laser rangefinder," in *International Conference on Robotics and Automation*, (San Francisco, California, USA), pp. 1145–1150, IEEE, Apr. 1986.

[11] J.-D. Boissonnat, O. Devillers, L. Donati, and F. P. Preparata, "Motion planning for spider robots," in *International Conference on Robotics and Automation*, (Nice, France), pp. 2321–2326, IEEE, May 1992.

[12] L. Kavraki and J.-C. Latombe, "Randomized preprocessing of configuration space for fast path planning," in *International Conference on Robotics and Automation*, vol. 3, (San Diego, California, USA), pp. 2138–2145, IEEE, May 1994.

[13] Motorola Semiconductor, *Technical data sheet for MPC555*, 1988.

[14] R. P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules," *SIGPLAN Notices*, vol. 23, pp. 49–62, Aug. 1988.