Search-based Planning for a Legged Robot over Rough Terrain

Paul Vernaza, Maxim Likhachev, Subhrajit Bhattacharya, Sachin Chitta, Aleksandr Kushleyev, Daniel D. Lee

Abstract— **We present a search-based planning approach for controlling a quadrupedal robot over rough terrain. Given a start and goal position, we consider the problem of generating a complete joint trajectory that will result in the legged robot successfully moving from the start to the goal. We decompose the problem into two main phases: an initial global planning phase, which results in a footstep trajectory; and an execution phase, which dynamically generates a joint trajectory to best execute the footstep trajectory. We show how R* search can be employed to generate high-quality global plans in the high-dimensional space of footstep trajectories. Results show that the global plans coupled with the joint controller result in a system robust enough to deal with a variety of terrains.**

I. INTRODUCTION

We consider the problem of planning for a quadrupedal robot over rough terrain. Traditionally, a solution to this problem may involve two parts: (a) sensing to perceive and create a model of the terrain in front of the robot and (b) planning and control to negotiate the rough terrain. In this work, we focus solely on the second part of the problem, namely planning and control for locomotion over rough terrain. We make the assumption of precise knowledge of the terrain and pose of the robot with respect to the terrain. This assumption is mostly justified by our use of a very accurate motion capture system and precise laser scans of the rough terrain. In spite of this assumption, creating a deliberate plan that will take the robot safely from one location to another remains an extremely challenging problem.

This problem can be approached using either a local or global planning approach. A local planning approach in [1] first planned an overall trajectory for the body of the robot and then planned over a finite horizon of footsteps. This approach had the benefit of planning over a small horizon and thus the planner could react faster to perturbations since it was always replanning the next step from the current position of the robot. This planner, however, has the disadvantage of getting stuck in local minima.

In contrast, a planner that pre-plans a global path from the start position all the way to the goal can avoid such local minima. Further, the actual *plan execution* is much faster since computationally intensive planning decisions do not have to be made in realtime and planned trajectories can be directly played back on the robot. A global planning method does have the disadvantage of greater initial *planning* time and the need for expensive replanning when the actual trajectory of the robot deviates from the planned trajectory. Replanning can be minimized by designing a controller that can accurately track the trajectories that the planner generates. Replanning is also minimized by choosing the cost function for the planner appropriately so that the planned trajectories lie well within the capabilities of the controller.

The challenge for a global planner lies primarily in the high dimensionality of the robot's state and action spaces. With the capabilities of current computing and search technology, it is necessary to exploit structure in the problem to make finding plans in this space a feasible task. We therefore present a decomposition approach based on a combination of global planning in the space of footstep trajectories and execution of this global plan online by solving for the remaining degrees of freedom necessary to execute the footstep trajectory. Effective planning in this high-dimensional space is made possible by our use of the R* search algorithm, which combines aspects of deterministic and randomized search to produce a feasible solution that also approximately minimizes our cost function. The effectiveness of the search enables us to specify a detailed cost function that takes into account many factors contributing to the stability and efficiency of the trajectory, thus (hopefully) maximizing the probability of the trajectory's successful execution.

A. Previous work

Statically stable walking has been widely studied in [2], [3], [4], [5]. Recent research using the LittleDog platform [6], [7], [8], [1], [9] has led to further exploration of quadruped gaits for locomotion over rough terrain. However, there has been limited recent research into the use of graph-based planners for legged locomotion. In particular, Eldershaw and Yim [10] and Hauser et. al. [11] have demonstrated graph-based planners for legged vehicles based on Probabilistic Roadmaps (PRM), a typical randomized planning algorithm. These techniques suffer from the shortcoming that the generated plans, though feasible, may be far from optimal. Post-processing may therefore be required to generate acceptable results [11]. Furthermore, in both of these cases ([10],[11]), the PRM-based planner is part of a greater planning hierarchy, which increases the complexity of the overall architecture. In our case, a single graph-based planner is sufficient to efficiently generate plans that can subsequently be followed by a reactive controller.

We also note that computational efficiency is of the utmost importance in our application. Kolter et. al. [8] address the efficiency issue by first planning a rough trajectory for the robot's center of mass without considering the trajectory of its feet. Footsteps are then planned to roughly follow this tra-

P. Vernaza, M. Likhachev, S. Bhattacharya, A. Kushleyev, D. D. Lee are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104 {vernaza,maximl,subhrabh,akushley,ddlee}@seas.upenn.edu

S. Chitta is with Willow Garage Inc., Menlo Park, CA 94025 sachinc@willowgarage.com

jectory while considering appropriate constraints. The initial center of mass trajectory provides a restricted search space to allow a simpler planner to be used to find footsteps. This restriction step may eliminate good plans from consideration before they reach the second-stage planner. Our planner does no such initial approximate pruning of the state space, and hence does not suffer from this problem. It is nonetheless efficient enough to generate plans of sufficient quality in short order.

B. Organization of this paper

This paper is structured as follow. We proceed by first discussing some important preliminary issues regarding quadrupedal walking that will simplify the development of our method (although we note that an extension to arbitrary leg configurations would be straightforward). We then describe the graph-based planner and the controller used to execute the plans. Finally, we present experimental results and conclusions.

II. PRELIMINARIES

It will help the exposition of our method to first briefly mention some assumptions we make about structure of the problem and the nature of the plans we will generate, and why these assumptions are valid.

First, we will assume a quadrupedal robot. This assumption is convenient for our purposes, but it is not critical; extending the method described in this paper to the case of $N \geq 4$ legs is straightforward. Fewer legs would necessarily violate the following assumption. Namely, we also restrict our attention to plans in which the robot is always *statically stable*. This implies that the robot's center of mass remains within the convex hull of its supporting feet at all times. The convex hull of supporting feet is otherwise known as the *support triangle* in the quadrupedal case, and is illustrated in Figure 1. In other words, this assumption implies that the robot will only raise one leg at a time, while the others remain stationary. This may result in a *crawl gait*, where the legs are raised in a specific, periodic pattern. Our plans are not forced into any particular gait, though they are slightly encouraged to follow a crawl.

Although statically stable walking tends to be much slower than dynamic locomotion, it is simple and somewhat error-tolerant, which is of the utmost importance in dealing with very rough terrain. Although dynamic legged locomotion over rough terrain is certainly possible in some cases with comparatively simple controllers [12], it is generally an extremely difficult problem due to the added complexity induced by the dynamics of legged locomotion. We therefore focus on the (still-difficult) problem of finding effective, statically stable plans.

Finally, we assume precise knowledge of the rough terrain. A motion capture system registers the robot and terrains. The terrains are laser mapped a-priori and the controller has full knowledge of the terrains. The controller also has knowledge of the state of the robot, namely all the joint angles and the pose of the robot.

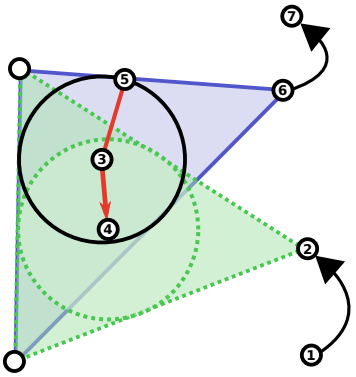


Fig. 1. Figure illustrating a possible walking sequence. The blue, solid triangle is the *support triangle*; i.e., the projection onto the ground of the triangle formed by the three supporting feet. The point labeled 3 is the *incenter* of the support triangle. The *incircle radius* is the distance from 3 to 5. The hind-right foot, initially at 1, is planted at 2, at which point the green, dotted triangle becomes the new support triangle. A highly conservative controller would initially have the center of mass at 3 and move it to 4, the incenter of the new support triangle, hence moving the center of mass backwards. The distance from 3 to 4 is the *COM travel*. After relocating the center of mass, the robot moves its front-right leg from 6 to 7.

III. THE METHOD

We begin by notionally constructing a *stance graph*, which describes possible transitions between foothold configurations of the robot. The goal of our footstep planner is to find a low-cost path through this graph that leads from the initial stance to a goal stance. This plan is incomplete in the sense that it includes only partial stance configurations, and does not include detailed joint positions or solutions for joint movements necessary to effect the plan. At runtime, a separate controller fills in these details, takes into account small plan deviations, and executes the joint controls necessary to follow the plan. We will first describe how the stance graph is constructed and how costs are assigned to paths in the graph. We will then describe details of the graph search algorithm and conclude this section by describing the controller used to follow the path.

A. Graph construction

Formally, we define the stance graph as a tuple (S, E), where each node $s \in S$ is a *stance*, and an edge $e \in E \subseteq S \times S$ in the graph represents a plausible transition from one stance to another. A stance is a tuple $(f_{fl}, f_{fr}, f_{hl}, f_{hr}, l)$ of (in our case) four feet locations $f_i \in \mathbb{R}^2$ and an index l representing the next foot to move in the nominal gait. The feet positions are defined in Cartesian coordinates with respect to some two-dimensional coordinate system level with the nominal ground. Subscripts fl, fr, hl, hr represent the front-left, front-right, hind-left, and hind-right feet, respectively.

It is neither desirable nor necessary in our method to actually store the entire stance graph in memory at once. Instead, we only require a way to generate successor stances (i.e., nodes adjacent to a given node in the stance graph) of any given feasible stance. In order to do this efficiently, we employ two fast pruning techniques; the first restricts

footholds to a sample of plausible candidate footholds, and the second restricts successor stances to those that could be reached by reaching for a candidate foothold in one step.

Candidate foothold selection is done as a preprocessing step that occurs once before planning. The preprocessing yields a set of "good" footholds spanning the entire terrain under consideration, as shown in Figure 2. These footholds are sampled with likelihood inversely proportional to the terrain cost, which is described in section III-B.8. Additionally, footholds are sampled so as to maintain a desired density per unit area, which can be varied to trade-off planning speed against precision.

Given a query stance, we can now generate a successor stance by examining candidate footholds in the workspace of the next foot to move according to the nominal (crawl) gait. A "null move" successor is always generated to allow the planner to bypass the nominal gait, with a certain associated cost. The other successors are generated by calculating an approximate workspace for the next foot to move. This calculation is necessarily approximate, as the state does not include the full pose of the robot, which affects the foot workspace, along with the specific shape of the terrain.

Therefore, we compute a nominal body pose that approximates the pose the controller will effect when executing this plan. Given this nominal body pose, we compute footholds that are approximately reachable from that pose. This is efficiently implemented by modeling the foot's workspace as a rectangle on the ground plane positioned with respect to the shoulder of the moving leg. A set of valid footholds is then generated by including all footholds on the terrain that fall within this rectangle. We note that a rectangular workspace was chosen for simplicity and efficiency, though this shape does not exactly match the actual workspace of the foot.

Since the successor-generating subroutine is called repeatedly by the planner, it is essential that it be as efficient as possible. We therefore precompute the workspace calculation by discretizing the set of possible support triangles. By exploiting symmetry and carefully choosing an efficient parametrization of the set of possible support triangles, we are able to feasibly store workspaces for all relevant configurations, using a 1cm resolution for footholds.

B. Cost function

As previously mentioned, we are not only interested in finding a feasible path of stances leading to the goal; we would also like to find one that is approximately optimal, in some sense. We define a path to be an ordered list of edges where each edge (s_i, s_j) represents a transition between two stances s_i and s_j. We define the cost of a path $c(P)$ to be the sum of edge costs $c(s_i, s_j)$ along the path P where the edge cost is the sum of the stance transition cost and the cost of being in stance s_j itself.

$$c(P) = \sum_{(s_i,s_j)\in P} c(s_i, s_j) \tag{1}$$

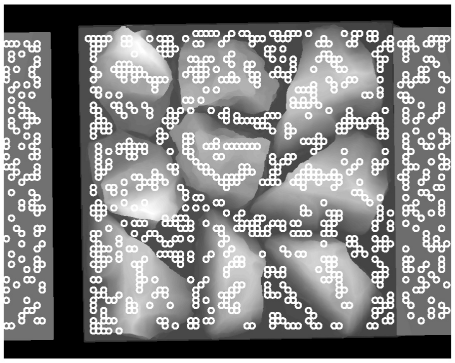


Fig. 2. Illustration of footholds selected for the rocks-with-gap scenario used in experiments

An edge (s_i, s_j) may represent a *null move* if s_i and s_j have the same foothold positions but different indices for legs to be picked up, i.e. $(f_{fl}, f_{fr}, f_{hl}, f_{hr})$ are the same for s_i and s_j but $l_i \neq l_j$. In this case, $c(s_i, s_j)$ is a constant cost chosen to discourage the plan from skipping legs in the gait cycle. Thus, the plan may choose to skip a leg in the gait cycle for this cost, or it may follow the gait cycle at no additional cost. For all other edges, for simplicity, $c(s_i, s_j)$ is a nonnegatively weighted sum of nonnegative component costs, with biases that can be used to set a nominal value for each component cost:

$$c(s_i, s_j) = \sum_{k=1}^{K} \max(w_k(c_k(s_i, s_j) - b_k), 0) \tag{2}$$

This obviously implies nonnegative costs, but it also implies monotonicity of cost, which greatly simplifies our choice of a heuristic function later on. Appropriate weights w_k were determined through a combination of intuition and experimentation. Though the simple linear form of the total cost admits more sophisticated ways of determining the weights (e.g., learning them), this issue is beyond the scope of this paper.

We now describe the component cost functions. As a whole, they are designed to encourage stable, robust plans that are also as efficient as possible.

1) Center of mass travel: the distance that the center of mass would move in moving from the *incenter* of the first support triangle to the incenter of the second support triangle (Figure 1). The incenter is the point in the support triangle farthest from all edges, and is therefore the most stable location for the center of mass with respect to perturbation or uncertainty, in one sense. Summing this distance along the path yields the total distance traveled by the center of mass. This cost is critical for the purposes of speed, since it discourages the center of mass from moving backwards (away from the goal). Unnecessarily moving the center of mass is costly from both time and risk perspectives.

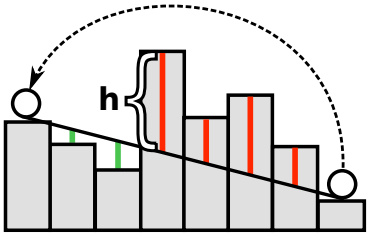


Fig. 3. Figure illustrating collision cost. Bars are proportional to terrain height, and circles indicate initial and final feet positions. Collision cost is proportional to the length denoted h in the figure.

2) Overhead: a constant cost associated with each step made. This encourages plans with few steps, since steps incur overhead associated with shifting the body and other miscellaneous effects. Note that this cost does not necessarily scale with the center of mass travel cost, as in the case of pure rotations, for instance.

3) Incircle radius: In order to make the plan as robust as possible, it is helpful to ensure that the center of mass can always move to a location such that small perturbations do not result in the center of mass falling outside the support triangle. The *incircle radius* is the radius of the largest circle inscribed in the support triangle, and is equal to the minimum distance from the incenter to any boundary. Therefore, it is helpful to penalize support triangles with small incircle radii, as these correspond to support triangles where it is not possible to move the center of mass far away from the support triangle boundaries. We assign a cost $\exp(-\alpha_r(r_i - \hat{r}_i))$ to a radius r_i, where $\hat{r}_i$ is a nominal minimum incircle radius, and α_r is a parameter.

4) Reachability: Although obviously unreachable stances are rejected by the successor-generating function, it may generate stances that may yet be unreachable due to a variety of factors. The reachability cost takes some of these factors into account and assigns a cost to the stance transition corresponding to the likelihood that the transition is unreachable.

This is done by computing nominal poses for both the initial and final stances, given the footholds, the terrain, and the expected controller behavior. A cost is computed based on the distance from the initial shoulder position to the desired foothold; a distance exceeding the leg length is penalized quadratically, thus assigning a high cost to footholds unreachable from the initial stance. Similarly, after shifting the center of mass to the new support triangle, one of the legs may get "left behind" (i.e., overextended). A cost is therefore assigned in the same way for this leg, and the two costs are summed to form the reachability cost.

5) Collision: Two types of collisions are accounted for in the cost function: collisions of the feet with the terrain, and collision of the body with the terrain. A body collision cost is computed as the maximum height of the terrain under the body, subtracting the mean height of the feet. A foot collision cost is computed as the maximum terrain height exceeding the height of a line connecting the initial and final footholds, as depicted in Figure 3

```
1  while s_goal is not expanded
2    select unexpanded state s ∈ Γ (priority is given to states not labeled AVOID)
3    if path that corresponds to the edge bp(s) → s has not been computed yet
4      try to compute a path from bp(s) to any state equivalent to s using ARA*
5      if failed then label state s as AVOID
6      else
7        create a new state s' to which a path was computed
8        add s' and edge bp(s) → s' to Γ, set bp(s') = bp(s)
9        remove s from memory
10         update g(s') based on the cost of the found path and g(bp(s'))
11         if g(s') > w h(s_start, s') label s' as AVOID
12   else //expand state s (grow Γ)
13     let SUCCS(s) be K randomly chosen states at distance Δ from s
14     if goal state within Δ or s = s_start, then add it to SUCCS(s)
15     for each state s' ∈ SUCCS(s)
16       add s' and edge s → s' to Γ, set bp(s') = s
```

Fig. 4. High-level overview of the modified R* search

6) Foot height variance: The variance of the feet height is penalized in order to encourage the robot to stay level. High variation in the feet heights is usually associated with undesirable rolling and pitching, which may cause instability.

7) Terrain slope: A high cost is assigned to hind feet placed on slopes whose normals are opposed to the direction of motion, which is a common cause of slippage.

8) Terrain cost: A high cost is also assigned to footholds that are unstable. A foothold is stable if it does not reside on the slope and small deviations from the foothold location do not result in drastic drops in heights. On the other hand, a foothold that is a local minimum in the height terrain is considered to be stable since the slippage is less likely to occur. The terrain cost is also set high for the footholds that are surrounded by footholds of high height because the robot foot can easily get stuck at such a location.

C. Graph search

We use a slightly modified version of R* [13] to search the stance graph. Pseudocode is given in Figure 4. R* operates by constructing a small graph Γ of sparsely placed states - in our case, stances - connected to each other via edges. Each edge represents a path in the original stance graph in between the corresponding states in Γ. In this respect, Γ is related to the graphs constructed by randomized motion planners [14], [15]. The difference is that R* constructs Γ in such a way as to provide explicit minimization of the solution cost and probabilistic guarantees on the suboptimality of the solution. To achieve these objectives, R* grows Γ in the same way A* grows a search tree.

At every iteration, R* selects the next state s to expand from Γ (see figure 4). While normal A* expands s by generating all the immediate successors of state s, R* expands s by generating K states residing at some distance Δ from s (lines 12-16). The distance Δ is some metric that measures how far two states are from each other. In our domain, this metric is Euclidean distance in between the center of mass positions corresponding to these states. If a goal state is within Δ from state s then it is also generated as the successor of s. A goal state is generated as any state whose center of mass is within the desired goal location. We also add a goal state to the list of successors of state s_{start}. This is the first modification from the original R* algorithm [13] that

allows us to avoid randomization in planning if a problem can be solved using a single execution of a deterministic search. R* grows Γ by adding these successors of s and edges from s to them.

A path that R* returns is a path in Γ from the start state to the goal state. This path consists of edges in Γ. Each such edge, however, is actually a path in the original stance graph. Finding each of these (local) paths may potentially be a challenging planning task. R* postpones finding these paths until necessary and tries to concentrate on finding the paths that are easy to find instead. It does this by labeling the states to which it can not find paths easily as AVOID states. Initially, when generating K successors, none of these states are labeled as AVOID - R* does not try to compute paths to *all* of the generated states. Instead, only when state s is selected for expansion does R* try to compute a path from the predecessor of s, stored in the backpointer of s $bp(s)$, to any state equivalent to s (lines 3-11). Two states s and s' are assumed to be equivalent to each other if the distance in between their center of masses is within small delta (three centimeters in our experiments).

While the original version of R* uses weighted A* (A* with heuristics multiplied by some weight $w > 1$) to compute local paths, our version of R* uses ARA* [16] to compute them (line 4). ARA* is an anytime version of A* that tries to compute a potentially highly suboptimal solution quickly and then tries to improve the solution as planning time allows. This allows us to improve the quality of transitions, if time permits us to do so.

R* stops the ARA* search, however, if it fails to find the path easily. (Within few seconds.) If it does fail, then R* labels state s as AVOID state since it assumes that it will be time-consuming to find a path to state s. If ARA* search does find a path, then the cost of the found path can be used to assign the cost of the edge $bp(s) \rightarrow s$. The cost of the edge and the cost of the best path from s_{start} to $bp(s)$, stored in $g(bp(s))$, can then be used to update $g(s)$ in the same way A* updates g-values of states.

R* provides probabilistic guarantees on the suboptimality of the solution. The uncertainty in the guarantee is purely due to the randomness of selecting K successors during each expansion. For a given graph Γ, on the other hand, R* can state that the found path is no worse than w times the cost of an optimal path that uses only the edges in Γ. To provide the suboptimality guarantees and minimize solution costs while avoiding as much as possible the states labeled AVOID, R* selects states for expansion in the order of smaller $f(s) = g(s) + w\,h(s)$), same as in weighted A*. However, it selects these states from the pool of states not labeled AVOID first. Only when there are no more such states left, R* starts selecting AVOID states (in the same order of f-values). In our domain, the heuristics $h(s)$ is set to Euclidean distance in between the center of mass of state s and the desired goal location.

D. Plan execution

The output from the foothold planner is an ordered list of stance transitions (s_i, s_j) for the robot to follow. Each stance transition requires moving only one foot of the robot. The planner does not specify a body trajectory or joint trajectories to achieve the desired footholds. Thus, the controller must compute and specify time-parameterized trajectories for the body and legs. The planner takes into account several factors like static stability, reachability and speed. However, the controller still must design the body trajectory carefully to account for small deviations from the plan. In addition to admitting static stability, the body trajectories must avoid collision with the terrain.

Let P be a path through the stance graph generated by the planner. Let $p_i \in P, i = 1, \ldots, n$ denote the i^{th} stance transition in the path where n denotes the total number of stance transitions in the path. The three feet that stay in contact with the ground in the i^{th} stance transition form the triangle of support for the current stance.

We will now present the strategy followed by the controller. We break the strategy into three parts (a) the motion of the projected (x, y) positions of the center of gravity of the robot, (b) control of the roll, pitch, yaw and height of the center of gravity of the robot and (c) control of the flight leg. Note that the body of the robot has a significantly higher mass than the legs of the robot and thus we consider the center of gravity of the robot to be the center of gravity of the body alone. Further, the origin of the local coordinate system of the robot is placed at the center of gravity of the body and thus we can decouple the strategies for the different degrees of freedom in this manner.

1) Motion of projected center of gravity: Intuitively, the controller tries to minimize the time spent by the robot in quad-support, i.e. the time spent with four feet on the ground. Each transition between stances may consist of two parts (a) a flight phase where the body of robot is moving with three legs in contact while the flight leg moves to the next foothold and (b) a quad phase where the body of the robot is moving with four feet on the ground. The quad phase facilitates the motion of the body to a new pose where it is possible to initiate the flight phase of the next leg to be picked up. However, we will show that the quad phase can be eliminated from alternate stance transitions by an appropriate choice of intermediate body poses.

Consider the robot to be initially in a stance s_k with the projection of the center of gravity of the robot onto the XY plane given by the point P_k^s as shown in Figure 5. The center of gravity of the robot lies within its current triangle of support (denoted by T_k in Figure 5). The controller must now move the robot to the next stance s_{k+1} specified by the planner. The first step in the stance transition is to initiate the flight phase of the leg with index l_k. During the flight phase, the robot can move its body to any point within the current triangle of support and still satisfy static stability. The flight phase may need to be followed by a quad-support phase. The choice of target body poses for the flight and quad-

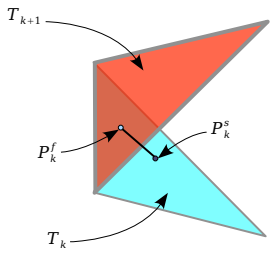


Fig. 5. Trajectory of projected position of center of gravity for transition between two stances with overlapping triangles of support.

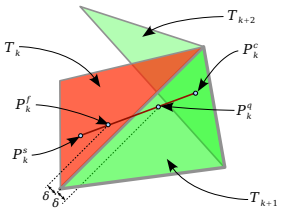


Fig. 6. Trajectory of projected position of center of gravity for transition between two stances whose triangles of support do not overlap.

support phases depends on the current stances and the next two stances chosen by the planner. There are two different cases depending on these stances that the controller needs to deal with.

Case 1

There is an overlap between the current triangle of support for the robot and the triangle of support corresponding to the next stance (Figure 5). This occurs when the foot l_{k+1} specified for pickup in the next stance is not diagonally opposite to the current foot l_k being picked up. In this case, the robot chooses a final body pose for the flight phase where the center of gravity of the robot is at the centroid (P_k^f in Figure 5) of these two triangles of support corresponding to p_k and p_{k+1}. Further, the quad-support phase of the motion for the transition between stances is eliminated. The projected motion of the center of gravity of the robot in the flight phase is along the line from P_k^s to P_k^f.

Case 2

There is no overlap between the current triangle of support and the next triangle of support. This occurs when foot l_{k+1} is on the opposite side of the robot with respect to foot l_k. This transition between stances now requires a quad-support phase after the flight phase to get the projected center of gravity over the common edge between the two triangles of support. During the flight phase of leg l_k, the controller first moves the projected center of gravity to a point P_k^f (see Figure 6) close to the common edge of support while keeping a distance δ away from the edge.

This reduces the distance traveled in the quad-support phase where the controller moves the projected center of gravity to a point P_k^q within the next triangle of support, again at a distance δ away from the common edge between the two support triangles to provide a margin of stability for the next flight phase (corresponding to the next stance transition). The controller chooses not to move completely to the incenter of the next triangle of support, electing instead to travel a smaller distance in quad-support while still maintaining a reasonable margin of stability.

The choice of P_k^f and P_k^q depends on the nature of the next two stance transitions. If the triangles of support for the next two stances overlap (as in Figure 6), P_k^f and P_k^q are chosen on the line joining the current position P_k^s of the projected center of gravity and the incenter P_k^c of the overlap region. If the triangles of support of the next two stances do not overlap, P_k^f and P_k^q are chosen on the line joining P_k^s and the incenter of the next triangle of support.

It is possible that at the beginning of a plan step, the controller needs to move the projected center of gravity into the triangle of support before initiating the flight phase of the leg to be picked up. This happens in the first step of the plan where the robot starts off in a nominal position with all four feet on the ground and must first move its projected center of gravity into the appropriate triangle of support before initiating the flight phase.

2) Control of the roll, pitch, yaw and height of the body: We will now specify the control strategy for the other four degrees of freedom of the body, i.e. the roll, pitch and yaw of the body and the height of the body above the ground. The controller chooses target values for each of these degrees of freedom and designs body pose trajectories to achieve the target values at the end of the flight phase of a stance transition. The controller uses these degrees of freedom mainly to avoid collisions of the body with the terrain, to allow generation of collision free trajectories for the legs and increase the reachable workspace for the legs so that the feet can reach the desired final footholds.

The roll degree of freedom is used to allow easier pickup and collision avoidance for the flight leg by increasing the workspace available to it. However, excessive rolling motions of the robot can easily destabilize it and so the controller only specifies very small rolling motions.

The yaw of the robot is controlled to keep the body of the robot centered between the two sets of feet on the left and right sides. Given a stance $p = (f_{fl}, f_{fr}, f_{hl}, f_{hr}, l)$, the desired yaw γ_d is calculated using the sum of the vectors joining the left and right sets of feet of the robot, i.e.

$$\gamma_d = \text{atan2}(f_y, f_x),$$

where,

$$f = (f_x, f_y) = (f_{fr} - f_{hr}) + (f_{fl} - f_{hl}).$$

The target pitch of the robot at the end of a stance transition is based on the difference between the average positions of the front and rear feet of the robot in the next stance. The pitch of the robot thus follows the contours of the terrain. The controller pitches the front of the robot up when the front feet are higher than the rear feet (e.g. when the robot is climbing up), and it pitches the robot down when the front feet are lower than the rear feet (i.e., climbing down). Note that the target pose of the robot is based on the position of the feet in the *next* stance. Thus, if the stance transition

involves picking up a front leg and putting it higher on a step, the controller simultaneously pitches the body up. This body motion is essential to make the new foothold of the front leg *reachable*.

The height of the robot is the only additional parameter available to the controller since all the other degrees of freedom for the pose of the body at the end of the stance transition have been chosen. The controller searches through a pre-defined range of body heights and checks explicitly for reachability of the next foothold with the given target pose. It chooses the highest height in this range for which the desired foothold is reachable.

3) Foot flight trajectories: The control strategy detailed above specifies a complete body trajectory for the stance transition. In addition, the controller must also specify trajectories for the flight phase of the leg that will be moved to the new foothold as part of the stance transition. The leg trajectory is designed so that the foot of the robot *follows* the contour of the terrain. Other alternatives include implementing trapezoidal or rectangular trajectories where the foot is raised a constant height above the ground. However, such trajectories often lead to workspace violations or require large motions of the joints and thus cannot be executed in short times. Minimizing the height through which the leg is lifted reduces the demands on the actuators and results in faster trajectories to achieve the desired foothold. Figure 7 shows foot trajectories designed by this controller to move the front left leg forward to the desired foothold over a variety of terrains.

4) Trajectory time parameterization: The speed of execution of the trajectories is a key parameter that determines the speed of completion of the gait and the robustness of the gait to disturbances. Executing faster gaits places greater demands on the actuators of the robot and so we modify the trajectories online to prevent saturation of the actuators. This is achieved by slowing down the gaits at points where the controller sees the torques coming close to saturation. However, since speed of completion is a primary objective, trajectories are also sped up in cases where the actuators are much below the saturation torques.

IV. EXPERIMENTAL RESULTS

We implemented and tested our approach on LittleDog, a small quadrupedal robot built by Boston Dynamics. Our experiments consisted of attempting to make LittleDog cross a variety of rough terrains using our method. As mentioned earlier, we assume accurate knowledge of the terrain and the pose of LittleDog with respect to the terrain. To obtain this information, we use a Vicon optical motion capture system to track the poses of both LittleDog and the terrain. The terrain consists of multiple rigid "terrain boards" that have been accurately scanned with a laser scanner to produce height maps. Fusing all this information yields a single, globally-referenced height map and LittleDog's pose with respect to the same reference frame. Figure 7 shows visualizations of the results of registering the terrain boards and LittleDog using the motion capture data.

We validated our method by applying it to the four scenarios depicted in Figure 7. In each scenario, LittleDog was assigned a goal location on the terrain about 1.8m away from its initial position that required it to traverse irregular terrain. Hazards on these terrains included various combinations of gaps, slopes, high steps, rocks, and crevices, as well as other challenges. We allowed the R* planner to plan for 85 seconds in each case, using a single-threaded C implementation running on a dual-core, 3 GHz Intel Xeon processor. We then executed the plans using our controller. This process was repeated ten times for each of the four terrain configurations, yielding the results shown in Table I. A trial was considered successful if the robot managed to reach the goal without falling over or becoming stuck. Video stills from an accompanying video of a successful trial are shown in Figure 8.

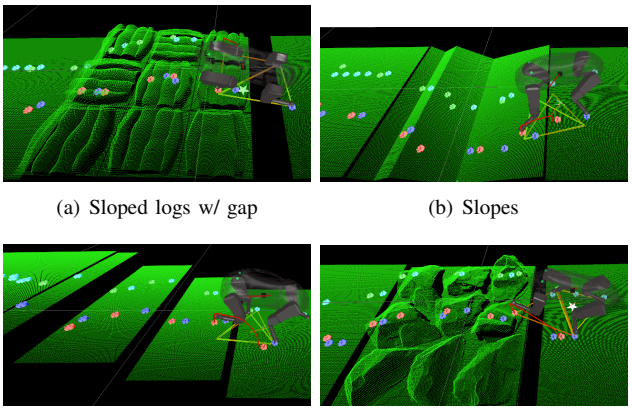
To provide a baseline for comparison, we also modified our R* implementation slightly to transform it into a weighted A* implementation. We then used the weighted A* planner to perform trials on the rocks-with-gap scenario, while using the same controller as in the other experiments. The ϵ parameter was set to the same ϵ as the starting ϵ used in the R* trials. Table II summarizes the outcome of this experiment. In each trial, the weighted A* planner did successfully generate a feasible plan; however, the costs of these plans were high compared to the plans generated by R*, indicating a much higher risk of failure. Attempting to execute these plans therefore predictably resulted in zero successes in ten trials, whereas executing the R* plans resulted in seven successes in ten trials.

This experiment illustrates both the ability of the R* planner to adequately minimize the cost function and the need to adequately minimize the cost function in the first place. Simply finding feasible plans, as a more traditional randomized planning algorithm might produce, is not sufficient to ensure success in this domain.

An analysis of the failures of the R* experiments yielded a few common causes. In multiple cases, failure resulted from a knee or other leg part colliding with the terrain. Our present cost function only explicitly penalizes feet collisions. Since the absence of a foot collision along a trajectory does not imply absence of collision with some other part of the leg, this is to be expected. Slippage was also a major cause of failure, particularly on the rocks board. Although some of the terrain costs try to mitigate the risk of slippage, uncertainty in foot placement tends to make it very difficult to completely eliminate the risk.

V. CONCLUSIONS

We have presented a search-based planning and control system for a legged robot over rough terrain. Central to our system is the planning of a trajectory of footholds using R*, a search-based method that combines strengths of both deterministic and randomized planning methods. This planner approximately minimizes a pathwise cost function that encourages various stability, robustness, and efficiency metrics. We have also presented a controller that is able to



(a) Sloped logs w/ gap (b) Slopes

(c) Steps (d) Rocks w/ gap

Fig. 7. Renderings of different terrain scenarios along with planned footsteps as LittleDog executes plans from the experimental trials.

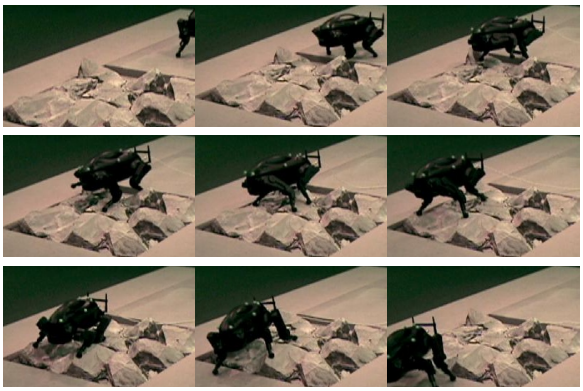


Fig. 8. Video stills from accompanying video of a successful trial on the rocks-with-gap scenario.

Scenario	Success rate	Mean speed (cm/s)
Sloped logs w/ gap	90%	2.95
Slopes	70%	2.35
Steps	50%	1.98
Rocks w/ gap	70%	3.13

TABLE I

RESULTS OF EXPERIMENTS ON LITTLEDOG ROBOT WITH R* PLANNING. 10 TRIALS PER SCENARIO, SPEED AVERAGED OVER SUCCESSFUL RUNS

Planner	Success rate	Mean cost	Cost st. dev.
R*	70%	1032	51.5
weighted A*	0%	3224	244.9

TABLE II

COMPARISON OF R* AND WEIGHTED A* PLANNERS ON ROCKS-WITH-GAP TERRAIN

effectively follow the trajectories generated by the foothold planner, thereby showing that this particular decomposition approach is feasible.

Our experiments have demonstrated the feasibility of using our system to control an actual quadruped over rough terrain. However, we have also identified some common failure modes that will need to be addressed in the future to further improve reliability and robustness. In particular, it is anticipated that better reactive recovery methods will greatly help improve the reliability of our system. Although other details such as improved collision modeling may also help, they come at the expense of requiring greater computational effort, which may degrade overall performance.

VI. ACKNOWLEDGMENTS

Thanks to Jon Bohren for his input and for helping with experiments. This work was supported by the DARPA Learning Locomotion program, grant FA8650-05-C-7260.

REFERENCES

[1] S. Chitta, P. Vernaza, R. Geykhman, and D. D. Lee, "Proprioceptive localization for a quadrupedal robot on known terrain," in *The IEEE International Conference on Robotics and Automation*, 2007.

[2] S. Hirose, H. Kikuchi, and Y. Umetani, "The standard circular gait of a quadruped walking vehicle," *Advanced Robotics*, vol. 1(2), pp. 143–164, 1986.

[3] S.-M. Song and K. J. Waldron, *Machines that Walk: the Adaptive Suspension Vehicle*. Cambridge, MA: MIT Press, 1989.

[4] S. Hirose and K. Yoneda, "Dynamic and static fusion gait of quadruped walking vehicle on winding path," *Advanced Robotics*, vol. 9(2), pp. 125–136, 1995.

[5] S. Chitta and J. P. Ostrowski, "New insights into quasi-static and dynamic omnidirectional quadruped walking," in *Proc. IEEE Intl. Conf. on Intelligent Robots and System*, Maui, October 2001.

[6] J. R. Rebula, P. D. Neuhaus, B. V. Bonnlander, M. J. Johnson, and J. E. Pratt, "A controller for the LittleDog quadruped walking on rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[7] D. Pongas, M. Mistry, and S. Schaal, "A robust quadruped walking gait for traversing rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[8] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008.

[9] B. K, A. Shkolnik, S. Prentice, N. Roy, and R. Tedrake, "Reliable dynamic motions for a stiff quadruped." in *To appear in Proc. of the 11th Int. Symposium of Experimental Robotics*, 2008.

[10] C. Eldershaw and M. Yim, "Motion planning of legged vehicles in an unstructured environment," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2001.

[11] K. Hauser, T. Bretl, J.-C. Latombe, and B. Wilcox, "Motion planning for a six-legged lunar robot," in *The Seventh International Workshop on the Algorithmic Foundations of Robotics*, 2006.

[12] J. D. Weingarten, R. E. Groff, and D. E. Koditschek, "A framework for the coordination of legged robot gaits," in *The IEEE Conference on Robotics, Automation, and Mechatronics*, 2004.

[13] M. Likhachev and A. Stentz, "R* search," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2008.

[14] L. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[15] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 995–1001.

[16] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.