Tractable Locomotion Planning for RoboSimian

Brian W. Satzinger, Chelsea Lau, Marten Byl, Katie Byl *

Abstract

This paper investigates practical solutions for low-bandwidth, teleoperated mobility for RoboSimian in complex environments. Locomotion planning for this robot is challenging due to kinematic redundancy. We present an end-to-end planning method that exploits a reduced-dimension RRT search, constraining a subset of limbs to an inverse kinematics table. Then, we evaluate the performance of this approach through simulations in randomized environments and in DRC-style terrain both in simulation and with hardware.

We also illustrate the importance of allowing for significant body motion during swing leg motions on extreme terrain and quantify the trade-offs between computation time and execution time, subject to velocity and acceleration limits of the joints. These results lead us to hypothesize that appropriate statistical "investment" of parallel computing resources between competing formulations or flavors of random planning algorithms can improve motion planning performance significantly. Motivated by the need to improve the speed of limbed mobility for the DARPA Robotics Challenge, we introduce one formulation of this resource allocation problem as a toy example and discuss advantages and implications of such trajectory planning for tractable locomotion on complex terrain.

*B. Satzinger, C. Lau, M. Byl and K. Byl are with the Robotics Laboratory, University of California at Santa Barbara, Santa Barbara, CA 93106, USA, {bsatzinger, cslau12, marten.byl, katiebyl}@gmail.com

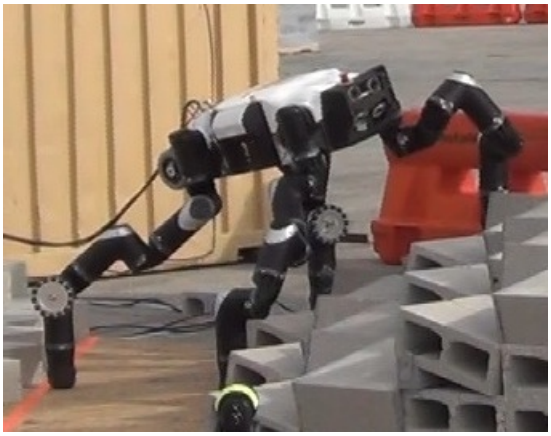

Figure 1: RoboSimian can balance more easily than a biped, but its high-DOF limbs complicate planning.

1 Introduction and Problem Statement

RoboSimian is a human-scale quadruped robot built by JPL to compete in the DARPA Robotics Challenges (DRC). It is inspired by an ape-like morphology, with four symmetric limbs that provide a large dexterous workspace and high torque output capabilities. The DRC is motivated by the possibility of using robots to aid in responding to disasters such as the nuclear disaster in Fukushima. The tasks in the DRC are representative of situations a robot might encounter in a disaster response scenario. Many of the tasks require manipulation, leading to the Robosimian's high degree of freedom design with seven joints per limb. On the other hand, the robot is also required to walk over rough terrain, as pictured in Figure 1.

RoboSimian is designed with distinct advantages that facilitate rough terrain walking. For example, its long-limbed, quadrupedal form provides a large, stable base of support, and the existence of redundant kinematic solutions increases the likelihood that joint configurations can be found to reach particular, desired footholds while avoiding collisions between the limbs and terrain. However, these same advantages provide significant challenges in experimental implementation of walking gaits. Specifically:

1. A wide support base results in high variability of required body pose and foothold heights, in particular when compared with planning for humanoid robots.

2. The long limbs on RoboSimian have a strong proclivity for self-collision and terrain collision, requiring particular care in trajectory planning.

3. Having the rear limbs outside the field of view requires creation of an adequate world map.

We present a tractable means of planning statically stable and collision-free gaits, which combines practical heuristics for kinematics with traditional randomized search algorithms, such as rapidly-exploring random trees (RRTs). In planning experiments, our method outperforms other tested methodologies, while field testing indicates that perception limitations provide the greatest challenge in real-world implementation. Our simulation tests show that the planner is capable of traversing DRC-style terrain as well as a variety of randomized terrains. We also examine the performance of our method relative to a known time-optimal synthetic reference trajectory, revealing that we are generating considerably sub-optimal solutions with respect to execution time under RoboSimian's joint velocity and acceleration limits. We further determine that the sub-optimality can largely be attributed to a lack of smoothness in the solutions combined with the acceleration limits of the robot, which we suggest may be the most important area of future research toward improving our speed of locomotion on rough terrain.

The problem of generating desired joint reference trajectories for this high-dimensional quadruped to walk on rough terrain is an example of kinodynamic planning (Donald et al. (1993), Donald & Xavier (1995)), simultaneously considering kinematic constraints as well as dynamics. For RoboSimian, the primary kinematic challenges involve selecting among redundant solutions and avoiding collisions of the robot with terrain obstacles and with itself, while the main dynamic constraints are joint velocity limits, acceleration limits, and static balance requirements. For quasi-static walking, consideration of joint accelerations and allowable center of pressure (aka ZMP) location are not usually key considerations. However, we have recently discovered that acceleration limits for RoboSimian slow trajectory execution time dramatically for typical RRT solutions, and we are currently implementing appropriate smoothing.

Comparing with past work in planning quadruped locomotion on rough terrain for LittleDog (Byl et al. (2009), Byl (2008), Kolter & Ng (2011), Zucker et al. (2011)), two particular challenges for RoboSimian are that it has seven degrees of freedom (DOFs) per limb, rather than three, and that perception relies solely on on-board sensing, rather than the use of motion capture (Vicon) along with saved (point-cloud) terrain maps.

Each of RoboSimian's identical four limbs consists of a kinematic chain of six rotational DOFs to define the (6 DOF) position and orientation of a lower leg segment, shown in green in Figure 2, relative to the body frame. A final (7^{th}) rotational joint simply allows the most distal end, or foot, of the lower leg to twist relative to the leg, so that the L-shaped lower leg segment itself can yaw while the foot remains fixed with respect to the ground. Even with only six actuators to set the 6-DOF pose of the lower leg, there are frequently redundant solutions. Qualitatively, each solution involves making one of two geometric choices (akin to "which way to bend an elbow") at each of three points along the chain: 2^3 results in a total of 8 IK families, as depicted in Fig. 2. The workspace and proclivity for self-collision of each family is different, and solutions for continuous trajectories in task space within a single family sometimes require discrete jumps in joint angles, so that kinematic planning is quite complex. In our problem formulation, we seek tractable methods to design trajectories for

all 28 actuated joints, for slow walking with high-torque joints, given a set of candidate footholds on complex terrain.

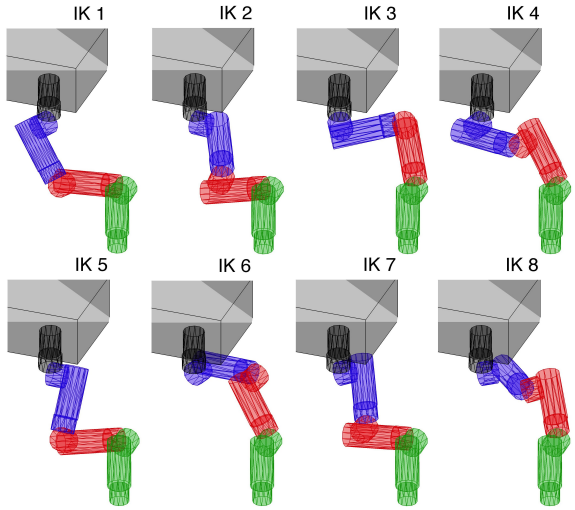

Figure 2: Redundant inverse kinematic (IK) solutions for RoboSimian.

As computational resources increase dramatically over time (via Moore's Law), it is clear our notions of tractable motion planning should evolve. One approach that is likely to thrive in this setting is the use of trajectory libraries (Stolle & Atkeson (2006), Schaal & Atkeson (2010), Jetchev & Toussaint (2009), Hauser (2013)), where precomputed trajectories are refined according to the specific situation. This presents its own set of (adaptive) computing resource allocation problems, to cover relevant dynamic motions with adequate fidelity. We are in the process of developing trajectory libraries for flat terrain locomotion. Another approach is to allocate computation cycles toward on-the-fly trajectory planning (e.g., via RRTs), and this is the path we investigate in exploiting cloud computing resources that will be provided to teams competing in the DARPA Robotics Challenge in June 2015. We note that practical use of RRTs in a time-constrained contest setting has already been investigated by Kuwata et al. (2008) on MIT's team for the DARPA Urban Challenge. In that work, random trees are built using a very low-dimensional search: i.e., x,y coordinates, plus a heading angle associated with each node. Our trees have significantly higher dimensionality, with either 9 or 16 angular degrees of freedom at each node. However, with our formulation, this search is still practical for on-the-fly locomotion planning. An open question we pose in this work is how best to allocate parallel computing resources to improve (quantitatively and/or qualitatively) the plans we generate for RoboSimian.

The rest of this paper is organized as follows. Section 2 presents our technical approach, which divides motion planning for the 28 actuated joints and 6-DOF pose of the robot into a lower-dimensional RRT search that pins either two or three limbs to a precomputed inverse kinematics (IK) table. Section 3 provides results for the approach, aimed at demonstrating practicality for real-world planning, while Section 4 introduces and analyzes a planning framework that uses parallel randomized searches as a means of achieving near-optimal and qualitatively diverse plans from which a remote operator can choose. Finally, Sections 5 and 6 provide conclusions and discuss potential extensions for future work. This paper builds upon the experimental results first presented in Satzinger et al. (2014). Compared with this prior work, Section 2 now provides a more complete overview of our A* foothold graph search as well as our motion planning approach, R2T2, which blends "RRTs with Tables" (of IK solutions) for tractability. Section 3 now includes results from two simulations of our planner over a simulated DRC terrain in addition to a real-world demonstration with RoboSimian in an outdoor environment. To demonstrate the robustness of the planner, we also show statistical results from a set of 900 RRT runs for swing trajectories on a terrain with several randomly-placed obstacles. Section 4 newly introduces a resource allocation framework for randomized planning. This parallel planning approach is motivated by the increasing availability of cloud computing resources for robotics planning – notably, the DARPA Robotics Challenge will provide such resources in the DRC finals in 2015 – and by the difficulty of accurately quantifying cost functions with a fully autonomous planner: i.e., availability of multiple solutions improves the statistical odds of

finding any one near-optimal solution and also allows for downselection by context-aware robot operators among qualitatively different solutions.

2 Technical Approach

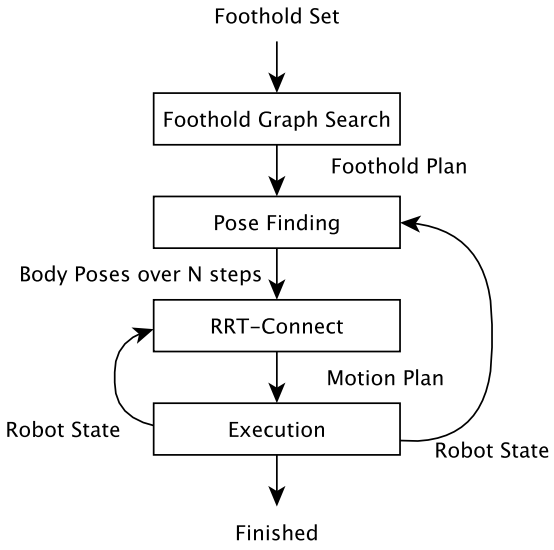

Figure 3: Planning phases in our approach

Our approach begins with a set of candidate foothold locations. We use a graph search to find a specific foothold plan, consisting of a series of steps that will be taken. At each node in the graph search, we perform a secondary search for a robot pose which is free of both self and world collisions while maintaning static stablility. If this search produces a valid body pose, then this proves by construction that it is possible to stand on the footholds described by this node. If this search fails, we assume it is impossible to stand on this node's footholds and reject the node. While the specifics of the body pose search differ, our search is analogous to the footstep planning search algorithm described in Section 4 of Zucker et al. (2011). If the graph search finds a feasible path through the stance graph, the required sequence of steps is stored in a foothold plan. If, however, the graph search fails, then a different set of candidate footholds must be provided (e.g., by a human operator), and the search restarted. The stance graph and the associated body poses which proved the feasibility of the nodes within the graph are discarded at this time.

In the next phase of planning, we again find body poses that are kinematically feasible, stable, and collision free. However, unlike during the A* search, when we were simply concerned with proving (by construction) that it is possible to stand on a single set of footholds, here we are generating the body poses which will actually be used during walking. This choice will significantly affect the overall performance. In particular, it is highly undesirable to require moving the body between every step. Accordingly, we discuss in Section 2.3 our approach for finding body pose solutions that considers multiple sequential stances together. After the Pose Finder determines feasible body poses for the beginning and end of each step, we are able to specify the complete sequence of motions (steps and body shifts). At this point, we invoke the RRT-Connect planner which plans the detailed motions between the starting and end positions, still respecting static stability, kinematic feasibility, and collision constraints. The motion plan is then executed on the robot (or a simulation).

The overall process contains multiple planning horizons of different lengths. The RRT-Connect planners always plan one motion (step or body shift) ahead, immediately before execution. The Pose Finding algorithm plans over a horizon of several (typically three or four) steps. For the experiments and simulations discussed in this paper, the Foothold Graph Search is performed over the entire terrain model to some goal location a significantly further distance (e.g., 10 meters) away. Due to inherent limitations in terrain perception at those distances, we expect that in practice the horizon used for the Foothold Graph Search will be significantly shorter (e.g., 2-3 meters).

It is possible for failures to occur at every level of the planning process, and there are a variety of automatic and manual interventions possible. The Foothold Graph Search may fail to find a solution

on the provided candidate footholds. In this case the operator is prompted to provide a different set of candidate footholds and to try again. The Pose Finding algorithm will, it turns out, never fail, as long as its input foothold plan was generated by the Foothold Graph Search. This is because, by design, the Pose Finding algorithm will always rediscover the proof-of-feasibility body poses generated during the Graph Search, if a more efficient solution does not exist.

The RRT-Connect planner may fail to find a solution[1]. In this case, the operator is given the option to have it try again (with possible success) or replan at either the Pose Finding or the Foothold Graph Search stage. Due to our dominant-limb formulation (to be discussed), we actually run multiple RRT planning queries and choose the plan with the fastest execution time of the plans that are successful. See Section 3.2 for a statistical analysis of our planners with different dominant limbs choices.

There may also be failures during the Execution phase. For example, the expected foothold position may simply be incorrect due to perception/localization errors, misapprehension of the physical properties of the walking surface (e.g., tissue paper covering a deep hole), or changes in the environment that have taken place after the world model was generated. After the robot's foot touches down and detects the ground (using force sensors), the foothold's 'true' location is then known. When running with the physical robot, there will always be discrepancies (typically small) between the expected and 'true' foothold location. We account for these discrepancies by updating the position of the foothold in the foothold plan (output of the Foothold Graph Search) and the queue of upcoming motions (output of the Pose Finding algorithm). We then check that the modified queue of motions is still valid (i.e., if we move the foothold to its new location, are the chosen body poses still feasible?). If it is, we continue execution immediately with the modified motion queue. If not, we discard the queue and re-generate it from the modified foothold plan. If this succeeds, we continue normally. If it fails[2], then the operator is prompted to choose between re-running the Foothold Graph Search on the modified foothold set, providing an entirely new foothold set, and aborting walking.

Because RoboSimian self-checks for anomalous conditions while moving, sometimes a motion is aborted before it is completed because a limit (e.g., joint tracking error) was exceeded. Sometimes an anomaly will be transient, but other times it reflects a real failure in the robot that requires operator intervention. Accordingly, if a motion is aborted during execution, the operator is given an opportunity to evaluate the robot's state and then decide to either abort walking or attempt to re-plan (using RRT-Connect) from the robot's current position to the goal location and execute the new plan.

The individual phases of our planning process are described in more detail below.

2.1 Foothold Graph Search

The first phase in our planning approach searches for a feasible sequence of steps to bring the robot to a specified goal location. First, a set of feasible footholds are selected from the terrain based on, e.g., perception and classification of the terrain, a priori knowledge of terrain shape as in DRC simulation, or manual selection. We then use a variation of the A* algorithm (Hart et al. (1968)) in order to construct a path from the selected set of footholds.

The A* algorithm makes use of a heuristic that estimates the cost of traversing between nodes. The only limitation on the heuristic is that it must be admissible (it does not overestimate the true cost). The cost function for the algorithm is the sum of the heuristic cost of traversing from the expanding node to the goal node and the actual cost of traversing from

[1]Although RRT-Connect is probabilistically complete, infinity is a long time. Additionally, it may simply be the case that the goal pose is not in fact reachable from the starting pose. Accordingly, the RRT-Connect planner will eventually time out and 'fail', for either reason.

[2]This is not inconsistent with the claim above that "The Pose Finding algorithm will, it turns out, never fail, as long as its input foothold plan was generated by the Foothold Graph Search." By modifying the position of a foothold, we no longer have a foothold plan that was generated (entirely) by the Foothold Graph Search. In practice, if foothold errors are small, this rarely happens.

the start node to the expanding node. Like the best-first search algorithm, the node with the smallest cost in the queue is expanded first. The tree is expanded until the goal node is reached.

Although A* search guarantees an optimal solution, it is suggested by Felner et al. (2003) that sub-optimal solutions may be found by related best-first search algorithms in much less time. In particular, they propose a K-best-first search algorithm that expands the K best nodes at once (A* corresponding to the special case of $K = 1$). We have implemented a modified approach where K threads expand nodes asynchronously. The empirical performance of this algorithm relative to alternative graph search algorithms is not in the scope of this paper, but some performance data will be presented in Section 3.3 of our implementation in the context of the entire system. We expect other foothold planning methods to be applicable as well (Byl (2008), Kolter & Ng (2011), Zucker et al. (2011), Vernaza et al. (2009)).

In our formulation, there is a fixed set of foothold locations provided as an input to the process. A selection of four (RoboSimian being a quadruped) unique footholds gives a particular stance, which may or may not turn out to be kinematically feasible (while also respecting collision and stability constraints). We search over an implicit graph, where nodes each correspond to stances which have been determined to be achievable, and edges connect two nodes whose stance footholds differ in only one position. Stances are determined to be achievable by searching for a body pose from which all footholds are kinematically reachable, while respecting collision and stability constraints. Our cost heuristic is the linear distance between the centroids of the footholds of two nodes (or between a node and the goal location). The complexity of the search depends on the candidate footholds provided. We provide some data about a typical example in Section 3.3. In general, we can successfully plan to walk on "nearly flat" ground with an enforced gait order, but on the DRC terrain, the free gait search or a gait search with weak order enforcement (i.e. you cannot take a step with the same limb twice in a row) is advantageous because it enables the negotiation of difficult parts of the terrain. We expect future work to modify the cost heuristic to reflect other planning preferences, such as the preferential use of certain footholds over others.

2.2 IK Tables

Once a foothold plan is determined, we must choose body poses at the beginning and end of each step. This process requires an inverse kinematics solution that addresses the difficulties inherent in high-DOF robot limbs. We used an IK table rather than an IK solver because, although an IK solver such as ikfast (Diankov (2010)) can easily provide an arbitrary number of solutions to achieve a given 6-DOF pose with RoboSimian's 7-DOF limb, ikfast does not distinguish among "families" of kinematic solutions, thus beginning and ending poses that come from different "families" may require that limb to be reconfigured during the transition. For example, one or more joints may require 180^o rotations to move from an "elbow-in" configuration to an "elbow-out" configuration. Our in-house IK solver is relatively slow but groups the solutions to minimize limb reconfigurations across the reachable workspace. To allow for a real-time implementation, we precompute an IK table in terms of only the relative 3-DOF position of a limb with respect to the body coordinate system. This exploits the fact that the lower leg need not be exactly normal to the ground during stance and greatly simplifies planning for body pitch and roll.

2.3 Body Pose Search

Another advantage of pre-computing an IK table for the (x,y,z) coordinates of a limb is that we can also test potential body poses for feasibility very rapidly. Given a set of either three stance legs and one swing leg, we set a nominal body orientation (roll, pitch, and yaw) heuristically to match the underlying foothold locations and heights. We search over a 6-DOF grid of potential body poses centered on the heuristic pose. The grid consists of a set of 50 points in the x-y plane, 7 discrete values for z, and 5 each for roll, pitch, and yaw. Thus, we examine up to $50 \times 9 \times 5 \times 5 \times 5 = 56250$ body poses. We search in an order that tests poses closer to the nominal pose

first, and we terminate as soon as a single feasible pose is found. A feasible pose consists of one that is kinematically feasible for all four limbs, with static stability on a support triangle (with margin) given by the 3 stance footholds.

In order to handle uncertainty in the terrain while planning, we also test that the swing foot will be able to reach above and below the planned foothold location while remaining kinematically feasible. This ground penetration distance can be set as a parameter. Choosing a larger value will allow greater uncertainty in the terrain, but also forces the planner to choose only conservative motions.

RoboSimian's four limbs together account for roughly 60 % of its total mass. Because limb motions affect center of mass location significantly, testing a step for feasibility requires performing two body pose searches, one with the swing foot at the initial pose, and one with the swing foot at the final pose. This also allows us to plan steps with different body poses at the beginning and end. In the results section, we will show the results of an analysis on the effect of allowing body motion during a step on the volume of reachable footholds. We will demonstrate that this capability does not have much advantage when planning a regular forward crawl gait on flat ground, but does significantly increase the reach on complex terrain with irregular steps or height changes.

We expect future work to address several shortcomings with this method of finding body poses. The approach is somewhat computationally expensive. We will quantify the time spent searching for body poses in comparison to other planning processes. The search also does not guarantee that the solution is better than other possible solutions or that it is far away from infeasibility. The authors expect to implement either a more sophisticated search or a subsequent pose optimization step to address one or both of these issues.

2.4 Motion Planning

Our general trajectory planning framework is described in more detail in Satzinger & Byl (2014), but we will reproduce figures and some explanation in this section for the sake of clarity. Once the footholds and body poses for each step are selected, a lower-level algorithm is needed to construct trajectories for the transitions between poses. Several works plan locomotion by first searching over a graph and then filling in allowable motions (Bouyarmane & A. (2012), Hauser et al. (2005), Bretl (2005)). In particular, Bretl (2005) developed a non-gaited motion planner for the LEMUR quadruped, which has 3 DOF per limb. Hauser et al. (2005) solved for non-gaited motions on a 36-DOF humanoid by focusing on clever (contact-before-motion) sampling, but a single step still required several minutes, and a plan for climbing a ladder took a few hours, computationally. We use RRT-Connect (Kuffner & LaValle (2000*b*)) to solve for a feasible paths between steps. Kuffner et al. (2002) has demonstrated this method to plan locomotion for a humanoid with 6-DOF limbs, but, in practice, this required a search over an apparently much smaller configuration space (e.g., C^3) than in our case (C^{16}).

We now provide a brief overview of the RRT-Connect algorithm, which is an extension of the original RRT algorithm (Kuffner & LaValle (2000*a*)). In the RRT-Connect algorithm, two trees are grown simultaneously – one from the start node and one from the goal node. In each iteration, one tree is chosen to be expanded randomly while the other tree is chosen to extend towards the last node in the first tree. The roles of the trees are then swapped in the next iteration. This process is continued until the connect tree is able to extend within delta of the last node of the extend tree.

To provide context for our experimental results, we will briefly summarize our implementation of RRT-Connect to find paths between initial and goal locations (from Satzinger & Byl (2014)). We parameterize the configuration space in a way that allows us to reduce the dimensionality substantially compared to a naive approach. Simultaneously, this approach also addresses kinematic closure of the stance limbs, allows full motion on the swing limb, and guarantees that the resulting joint trajectories are continuous.

We will denote a coordinate transformation from frame a to frame b by C_b^a. Coordinate transforms can be multiplied ($C_c^b C_b^a = C_c^a$) and inverted ($(C_b^a)^{-1} = C_a^b$). We assume a fixed *world* frame, a *body* frame

attached to the robot body, and foothold frames. We will refer to a foothold (frame) by f, which can be indexed by limb. Therefore, $C_{f_i}^{world}$ gives the location of the i_{th} foothold in the $world$ frame. We will use i as a generic limb index, and sometimes d and s to denote the index of a dominant or swing limb (to be defined) as appropriate. A joint trajectory over time is written $q(n)$, but where n is defined, $q(n)$ is understood to refer to a specific sample. The joint angles of limb i are denoted by q_i, while the joint angles of the entire robot (consisting of appending all of the q_i's) are designated by q. We will also designate optional function parameters (related to the swing leg) in $[brackets]$. Some functions will return a $status$ variable, intended to indicate the $Success$ or $Failure$ of the function call.

Then, we can define a forward kinematics function $FK(i, q_i)$ and an inverse kinematics function $IK\_TABLE(i, C_{f_i}^{body})$ implemented as a lookup table specifically designed with certain properties (solutions are unique, and trajectories through the workspace are smooth). An algorithmic approach for designing the IK tables is described in detail in Byl et al. (2014).

$$C_{f_i}^{body} = FK(i, q_i). \tag{1}$$

$$(q_i, status) = IK\_TABLE(i, C_{f_i}^{body}) \tag{2}$$

Figure 4 illustrates this approach. During a swing motion, the complete pose of the robot can be specified by the 7 joint angles of the dominant limb and the 7 joint angles of the swing limb. We can also allow rotation at the contact between the dominant limb and the ground by introducing 2 additional degrees of freedom to give roll and pitch at this contact. This gives a total of 16 degrees of freedom. In the results section, we discuss the effect the choice of dominant limb has on the solutions provided by the RRT-Connect algorithm. In future work, the authors plan to take advantage of this property and the availability of multiple cores to simultaneously run multiple parallel instances of the RRT-Connect algorithm with a certain percentage dedicated to each possible dominant limb according the to their probability of obtaining a "fast" solution. A post-processing procedure will then select the best solution.

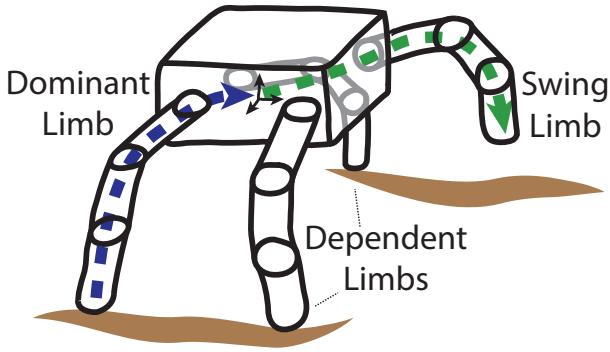

Figure 4: Cartoon sketch from Satzinger & Byl (2014) illustrating the design of our RRT-Connect configuration space parameterization for RoboSimian.

The initial and goal configurations for the dominant and swing limbs are determined using the IK table lookup procedure. However, the joint trajectories are not limited to IK table solutions in the RRT-Connect search. This allows for complicated maneuvers during swing motions while still maintaining continuity in IK solutions between steps.

The remaining two dependent limbs are not directly represented in the parameterization. We determine their positions using IK table lookups after obtaining the trajectory solutions for the dominant and swing limbs. Using forward kinematics, the joint configurations of the dominant limb are used to define the body positions along the trajectory. In turn, the body positions provide the location of the footholds of the dependents limbs relative to RoboSimian's body. It is possible to generate poses where one or more dependent limbs do not have a kinematically feasible IK table solution. These poses are considered to be infeasible and are not used.

A similar approach can be used to allow a body shift with all four feet in contact with the ground along with different initial and final body poses. This still requires one dominant limb to determine the body pose and the remaining three limbs to be dependent limbs. This gives 9 degrees of freedom, including

Algorithm 1 Procedure for calculating the full pose (consisting of q and C_{body}^{world}). $C_{f_i}^{world}$ gives the foothold locations relative to the world, d and q_d give the index and joint angles of the dominant limb, and optional parameters s and q_s give the index and joint angles of the swing limb (if applicable).

procedure $FULL\_POSE(C_{f_i}^{world}, [d, q_d], [s, q_s])$
  $C_{f_d}^{body} \leftarrow FK(d, q_d)$
  $C_{body}^{world} \leftarrow (C_{f_d}^{body})^{-1} C_{f_d}^{world}$
  for $i = 0$ to $(N_{limbs} - 1), i \neq d, i \neq s$ **do**
    $C_{f_i}^{body} \leftarrow C_{f_i}^{world} (C_{body}^{world})^{-1}$
    $(q_i, status) \leftarrow IK\_TABLE(i, C_{f_i}^{body})$
    if $status \neq Success$ **then**
      return $(C_{body}^{world}, q, Failure)$
    end if
  end for
  return $(C_{body}^{world}, q, Success)$
end procedure

the roll/pitch contact with the dominant foothold.

One issue with trajectories generated using RRTs is that they are not smooth. Such trajectories slow down the overall motion during steps because the sharp changes in direction require the joints to slow down in order to obey the velocity and acceleration limits of the motors. The authors expect to address this issue using methods such as the post-processing smoothing procedure in Hauser & Ng-Thow-Hing (2010) that also takes velocity and acceleration limits into account.

All dimensions are given in radians, and angles are not wrapped at multiples of 2π. Although RoboSimian's actuators do not have hard joint limits and can continuously rotate, accumulating several rotations could damage cabling passing through the actuators. To avoid the accumulation of rotations during planning, we treat all joint angles (e.g, 0 and 2π) as distinct.

3 Results

We present results from various simulation and hardware experiments that demonstrate the capabilities and benefits of our approach to trajectory planning for Robosimian.

3.1 Feasible Step Volume

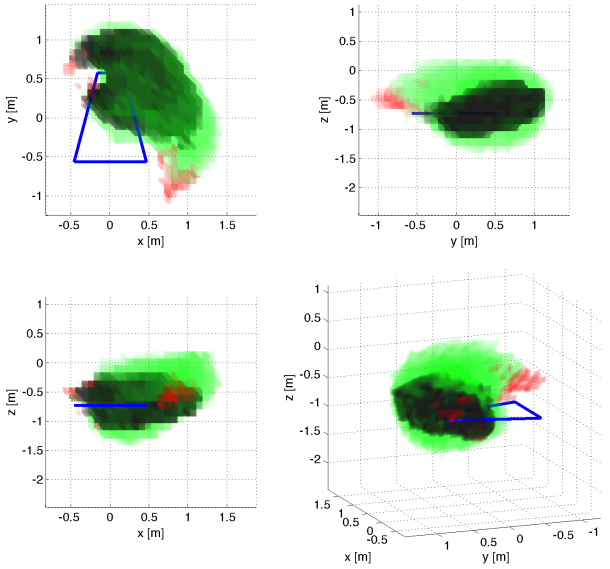

Figure 5: Volumetric rendering of the reachable volume from several perspectives. Black shows the volume S, and green shows the relative complement of S in D (showing the region reachable only with body motion during a step). Red shows regions contained in S and D where the RRT solvers were not able to find a path, despite the existence of feasible initial and final poses. Blue shows the outline of the initial support region, with the initial swing foothold embedded in the reachable volume.

In contrast with the planning approach used on RoboSimian during the 2013 DRC trials, our RRT configuration space design supports movement of the body during a step, from an initial pose to a final pose. We performed an analysis to consider the effects of this design decision on the ability for our system to plan steps on difficult terrain.

For the analysis, the robot's initial footholds are placed in a crawl-gait position corresponding to a steady state step length of 0.50 meters. Then, we

consider planning a step with one of the limbs from its initial position to a set of goal positions on a 3D grid. The search grid had 32 points in x and y, and 18 points in z. We compare the volumes reachable when the initial and final poses are forced to be the same, and when they are allowed (but not required) to be different. Therefore, the "same pose" volume S will always be a subset of the "different pose" volume D, and we can quantify the benefits of this approach by the size and shape of the relative complement of S in D.

The analysis found that 2541 of the destination foothold locations were likely reachable based on finding a feasible body pose for the end of the motion. Our RRT-Connect implementation was able to find paths to 2499 of them (98.3%). The remaining 42 points where the RRT-Connect search failed to find a path are shown in red in Figure 5. The 2499 successful paths represent 1.92 km of total motion, with an average step length (linear distance from start to finish) of 76.8 cm, and a maximum step length of 169.7 cm.

In contrast, when the body position was required to be the same at the beginning and end of the step, only 1065 steps were possible, with an average step length of only 55.7cm, and a maximum step length of only 130.6cm.

As Figure 5 shows, the reachable volume is substantially increased by allowing the body to move during a step. Notably, the maximum distance that it is possible to move the foot directly forward (positive x direction) does not vary much between S and D, so there is little benefit for a regular crawl gait on approximately flat terrain. However, S is much larger in other directions, suggesting benefits for irregular gaits, especially on terrain with large changes in height, or when the robot is turning.

3.2 Obstacle Avoidance

To characterize typical performance of our R2T2 algorithm for complex and unpredictable obstacle geometry, we generated thirty random sets of footholds and random obstacles for Robosimian to avoid when taking one step forward. For each set of footholds and obstacles, initial and final body poses were found using the body pose search described in Section 2.3, and thirty searches were run for each of the three (non-swing leg) available choices of dominant limb, for a total of 900 searches per dominant limb.

Recall that one motivation in our work is to develop a strategy that can exploit parallel computing. As a result, we are particularly interested in the statistics of the entire group of thirty searches performed for any one of the thirty random terrain cases.

Starting footholds were chosen from a uniform random distribution centered around nominal foothold positions (without rotation), with bounds of ± 0.2 meters in x (forward/backward) and y (right/left) and ± 0.1 meters in z (down/up)[3]. The goal foothold was randomly selected from a uniform distribution centered 0.4 meters ahead (+x) of the starting swing foothold, with bounds of ± 0.1 meters in x, y, and z. We placed three boxes near the swing footholds, and distributed five boxes randomly throughout the workspace. The sizes of the boxes were chosen from uniform random distributions on the intervals $[0.04, 0.2]$ meters in x, $[0.1, 0.5]$ meters in y, and $[0.2, 1.0]$ meters in z. The poses of the boxes were chosen from a uniform random distribution centered around the geometric center of the start footholds with bounds of ± 1.0 meters in x (forward/backward), ± 1.0 meters in y (forward/backward), and $[-0.1, 0]$ in z (down/up). The roll, pitch, and yaw of the boxes were also uniformly distributed on the interval $[-30, 30]$ degrees. (See Figure 6.) All simulations for this experiment were run on a computer with a 3.40 GHz Intel i7-2600 CPU. Each search terminated either when a solution was found or 100,000 nodes were added to either tree, whichever occurred first.

Of the 900 total runs, 897 (99.67%) were successful in finding full body trajectories for the step for at least one choice of dominant limb before "timing out". After running thirty trials on each terrain, we were always successful in finding multiple successful plans for each of the thirty terrain cases. The average number of nodes (both trees combined) for the 897 "best" trajectories was 34,755 with a standard deviation of 53,418. The average computation time for running the planner three times (for each available

[3]The sign convention for z is that -z is 'up' and +z is 'down'.

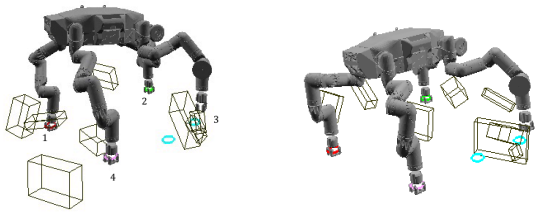

Figure 6: Two examples of randomly-generated obstacles and footholds. The robots shown are facing to the left. The front, right foot, shown at the far left in each image, is Limb 1, with limb numbers increasing as one goes clockwise, as viewed from above the robot. Thus, red circles mark footholds for Limb 1, green circles for Limb 2, blue circles for Limb 3 (i.e., the swing limb), and the pink circles for Limb 4.

choice of dominant limb) was 28.00 seconds with a standard deviation of 42.91 seconds. We summarize the success rates and computation times with respect to each choice of dominant limb in Table 1.

Within the table, notice that on some terrains, a particular choice of dominant limb sometimes results in a 0 % success rate, although solutions were found for one or both of the other limbs; we discuss likely explanations and consequences for this later in this section. Also note that the computation time of the unsuccessful RRT-Connect searches is affected by the number of nodes we allow to be added before terminating the search and can theoretically approach infinity if allowed to run to completion. In our set of searches, unsuccessful searches ran for 58.58 seconds on average before timing out. By comparison, successful searches found solutions in 3.11 seconds, nearly one twentieth of the average computation time of unsuccessful searches. However, we intend to make use of parallel searches, selecting the solution that is returned first, which would then shift emphasis to the minimum computation time with respect to the available choices of dominant limb from the aggregate of all choices. In the 900 runs, the average minimum computation time was 0.65 seconds with a standard deviation of 0.87 seconds. This highlights the influence parallel searches would have on the computation time associated with planning swing trajectories.

We find it interesting that, of the thirty sets of data, there were multiple cases where only one choice of dominant limb resulted in a solution. This provides evidence for our hypothesis that (in these cases) one stance limb must almost certainly deviate from the IK table solutions to permit a collision-free solution to be found. It also illustrates why alternate approaches, such as an RRT search over the 6 degrees of freedom of a floating base, with no dominant limb (but using IK tables to guarantee continuous solutions), would be very unlikely to find a solution for the same planning queries.

For our choice of swing limb, Limb 2 seems to provide the best chance of resulting in a solution when used as the dominant limb. However, there is one terrain example in which Limb 2 failed as the dominant limb in all thirty iterations of the RRT. These results indicate sensitivity both to the obstacle locations and sizes as well as the choice of swing limb. Our conclusion is that it is simply best to search over all possible dominant limbs since we cannot predict all scenarios in which Robosimian may find itself. Searching over multiple dominant limbs also increases the variability among successful plans found, improving the odds of finding a lucky motion plan, near the fast end of the tail in required execution time.

Computation Time vs Execution Time

The idea of terminating parallel runs of the RRT based solely on the minimum computation time brings up the question of the efficiency of the plans returned first. Table 1 summarizes the variability in *computation* time required to find either the "best of 30" parallel searches performed or the "average computation time" overall. Additional statistics of interest include the average *execution* time required for the robot to perform the resulting motion plan, as well as the potential benefit in execution time if one waits for additional parallel computations to be completed, rather than taking the first feasible solution found.

The trade-off between computation time and execution time for our collision avoidance data set is summarized in Figure 7. At left, as one waits

for longer for parallel RRT searches to complete, the best-yet execution time decreases monotonically. However, there are diminishing returns. For example, if one always waits until 2.6 times the time required for the first feasible solution to be found, the average execution time drops from 5.67 seconds to 4.18 seconds (highlighted by the + symbol at left), representing an average execution time that is 0.74 times the first-found solution (highlighted on the normalized plot at right).

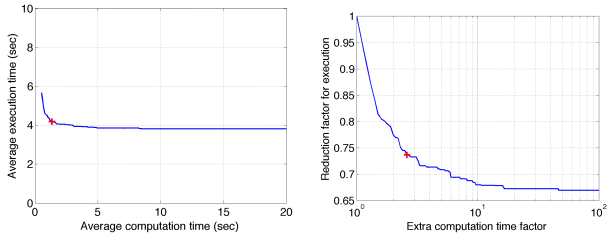

Figure 7: Computation time versus execution time. At left: best-yet execution time reduces with increasing computation time. However, there are diminishing returns over time. At right: the same data are presented with computation time and execution time normalized by the average times for the first-found solution.

In practice, we wish to minimize the *combination* of computation time plus execution time. Taking the first solution found, the average computation time is 0.51 seconds and the average execution time for the motion plan is 5.67 seconds, resulting in an average of 6.18 seconds per step on cluttered terrain. We explored policies which allowed the parallel planner to run for the *minimum* of either some constant, k_{comp}, times the first-found computation time of some fraction, f_{exec}, of the first-found execution time. Based on our sample data, in which we ran 30 searches for each of 30 randomly generated terrains, the optimal parameters for this simple policy are $k_{comp} = 2.6$ and $f_{exec} = 0.14$, resulting in the total time changing from 6.18 seconds to 5.18, i.e., to about 84% of the "greedy" first-found case. The expected reduction is execution time alone after waiting $k_{comp} = 2.6$ times the first-found solution is highlighted on both subplots of Figure 7 with a + symbol.

Trial	Limb 1 Dominant			Limb 2 Dominant			Limb 4 Dominant		
	N	T_{avg}	T_{min}	N	T_{avg}	T_{min}	N	T_{avg}	T_{min}
1	30	0.22	0.13	30	0.50	0.28	30	0.42	0.23
2	0	N/A	N/A	30	8.57	2.15	30	0.78	0.44
3	30	5.58	0.84	30	2.43	0.76	30	2.91	0.63
4	30	0.15	0.09	30	0.18	0.11	30	1.16	0.45
5	30	0.24	0.12	30	1.34	0.43	30	0.16	0.09
6	29	5.27	0.67	30	0.49	0.26	30	4.88	0.55
7	30	5.72	0.32	30	1.34	0.24	29	10.86	0.42
8	30	1.60	0.19	30	0.51	0.25	30	1.82	0.95
9	30	1.33	0.39	30	0.47	0.29	30	0.19	0.14
10	30	1.64	0.11	30	0.24	0.09	30	0.27	0.12
11	29	9.56	1.88	30	1.70	0.29	30	0.93	0.20
12	24	13.12	0.79	30	1.32	0.44	30	0.47	0.21
13	30	0.34	0.18	30	0.23	0.11	30	0.13	0.09
14	29	13.50	3.11	30	2.51	0.77	30	5.34	1.39
15	29	6.24	0.93	30	0.57	0.26	29	8.00	0.82
16	30	0.76	0.19	30	0.33	0.17	30	0.30	0.17
17	30	0.06	0.05	30	0.13	0.07	30	0.53	0.13
18	27	5.58	0.79	30	0.88	0.25	28	11.35	1.99
19	0	N/A	N/A	30	1.93	0.49	30	10.83	1.15
20	0	N/A	N/A	0	N/A	N/A	27	22.03	7.07
21	30	1.53	0.32	30	0.46	0.25	30	1.90	0.33
22	0	N/A	N/A	30	0.42	0.25	30	2.60	0.70
23	30	0.39	0.24	30	0.51	0.26	30	0.89	0.18
24	30	4.33	0.47	30	5.39	0.82	30	1.18	0.33
25	0	N/A	N/A	30	2.27	0.70	0	N/A	N/A
26	30	1.49	0.41	30	5.39	0.81	30	0.96	0.24
27	18	27.49	9.05	30	2.65	0.34	30	15.79	1.21
28	30	1.06	0.20	30	1.41	0.55	30	0.49	0.25
29	30	1.36	0.17	30	0.28	0.16	29	4.14	0.76
30	0	N/A	N/A	30	3.08	0.88	0	N/A	N/A
Summary	695 (77%)	4.02*	0.90**	870 (97%)	1.63*	0.43**	832 (92%)	3.87*	0.75**

Table 1: Results from the thirty sets of simulations with random sets of obstacles and start and end foothold locations. N is the number of successful runs in a given trial, T_{avg} is the average computation time of the N runs, and T_{min} is the minimum computation time of the N runs. We are interested in the minimum computation time as a termination criterion for parallel searches. *The overall average computation time is over the successful runs of all 900 runs per choice of dominant limb. **In the summary row, we provide the average minimum time over the thirty trials.

3.3 Traversal of DRC Terrain

We performed both simulation and hardware experiments to quantify the performance of our approach. A simulated DRC terrain was created by placing footholds on a $16 \times 16 \times 6$ inch grid, to match the spacing created by the cinderblocks used in the trial. An additional foothold was placed on each terrain level above ground level in order to provide more choices to the Foothold Graph Search (which did not find a foothold plan when a single foothold was centered on each grid tile). The goal location was set 10 meters from the starting location. In order to reduce the A* search space, only two rows of the terrain grid were populated with footholds, as this was sufficient to allow a solution (Figure 8). We also did not consider the orientation of the footholds, which in reality are on locally sloped surfaces for the second half of the terrain. Therefore, the first and second 'hills' in the course are identical in this simulation. It is not entirely redundant to cross two identical 'hills', because the robot must still negotiate the transition from one 'hill' to another.

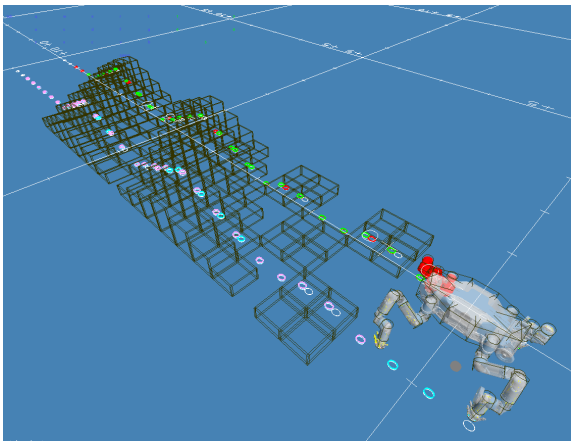

Figure 8: RoboSimian at the beginning of a simulated DRC terrain crossing, showing terrain collision geometry and foothold plan. Foothold plan is highlighted with colors by limb (front-right: red, rear-right: green, rear-left: cyan, front-left: pink). (Blue background indicates 'idle' software state.)

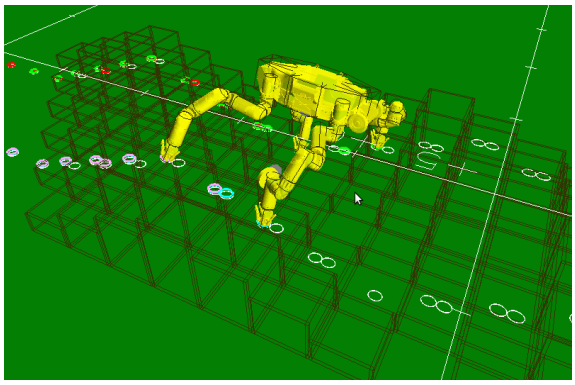

Figure 9: RoboSimian during the middle of simulation, straddling both 'hills'. (Green background indicates that a motion is currently being executed, and the yellow robot indicates the current 'live' position of the robot as reported by the low level control software. In this case, the low-level software is operating in a special offline simulation mode. When operating with hardware, the 'live' pose is the actual position of the robot (see Figure 11).

We provide the planner with a terrain model, consisting of collision geometry encompassing the cinderblock stacks. The collision model is used for checking feasibility during all phases of planning. For the described simulation experiment, the collision geometry differs from reality in the following ways. First, the cinderblocks are grouped into stacks rather than being represented individually. Second, although it is not discernable from the 'wireframe' graphics, the boxes are actually open at the top. This is a simplification made in order to work around a software bug related to selectively excluding certain end effectors from the collision geometry (which is necessary to discern if a robot pose is truly in collision with the terrain or merely resting on it with the end effector). Finally, as menioned previously, the second half of the terrain lacks the slopes present in the real terrain.

The simulation was performed using RoboSimian's control software in a special offline mode, which provides the software interfaces, interpolation algorithms, and error checking that are used with the

robot hardware. Therefore, the planning / execution cycle occured in real time with representative communications overhead. All software involved in the simulation was run on a single laptop with an Intel i7-4900MQ 2.8 GHz CPU and 16 GB of memory. All planning was done autonomously, without operator input.

The A* search was executed once at the beginning for the entire terrain. This process was multithreaded using 7 threads to expand the search tree in parallel. There were 102 footholds associated with the terrain model, and an additional 4 footholds for the robot's starting position. When the search terminated there were 4988 nodes in the closed set, and 553 nodes in the open set. Most of time spent during the Foothold Graph Search is spent determining if a stance is feasible before adding it to the graph. On average, expanding a node generated 2.01 feasible child nodes (the maximum was 14 child nodes).

We performed two experiments under identical conditions, except for the Pose Finding horizon, which was set to 3 and 4 steps ahead. Some implications of the size of the Pose Finding horizon will be discussed later in this section. The Pose Finding search used 8 threads.

Our previously described algorithms were used to find feasible and stable paths between the initial and final poses given by each step or body movement. Our formulation gives a special role to one of the three (during a step) or four (during a body shift) stance limbs, which we call the 'dominant' limb. We call the RRT-Connect solver three (or four) times iterating over which limb is 'dominant'. The solution with the shortest time (respecting the robot's actuator velocity and acceleration limits) is used. Because these calls are independent, in principle, they can be parallelized. However, for this experiment they were done serially.

For this offline experiment, we did not include uncertainty in the terrain height relative to the given foothold locations. This would have allowed us to use pipelining and, for example, plan for the next step while the current step was still executing. However, when running on hardware we discover the true height of the terrain at the end of a step when contact is made with the ground and incorporate that knowledge into the planned sequence of motions. Therefore, we postpone calling the RRT-Connect solvers until just before the motion is to be executed so that motions are planned with the most up to date information possible.

Subtask	Time (s)	Time (%)
Foothold Graph Search	169.9	13.1
Pose Finding	192.4	14.9
RRT-Connect	212.8	16.5
Execution	647.2	50.0
Other	71.1	5.5
Total	1293.5	100.0

Table 2: Reprinted from Satzinger et al. (2014): Time spent in various subtasks (non-overlapping) while simulating DRC terrain crossing, without collision detection. See the text for a description of the significant differences between this simulation, and the simulations described in Tables 3 and 4.

Subtask	Time (s)	Time (%)
Foothold Graph Search	221.60	11.52
Pose Finding	377.50	19.63
RRT-Connect	571.73	29.72
Execution	972.69	50.57
Other	1.59	0.08
Total	1923.51	100.0

Table 3: Time spent in various subtasks (non-overlapping) while simulating DRC terrain crossing, with Pose Finding horizon of 4 steps.

Tables 3 and 4 show data for two runs with identical terrain geometry, candidate foothold locations, starting position, and goal position. We varied the Pose Finding planning horizon (four steps, and three steps, respectively) for two reasons: first, in order to facilitate comparisons to previously published simulation results with a horizon of four steps in Satzinger et al. (2014) (reprinted in Table 2) and to real-world demonstration to be discussed shortly which used a horizon of three steps; and second, in order to provide an illustration of the relationship between errors in the expected positions of footholds, time spent find-

Subtask	Time (s)	Time (%)
Foothold Graph Search	221.90	12.16
Pose Finding	156.03	8.55
RRT-Connect	609.76	33.42
Execution	1057.97	57.98
Other	0.81	0.04
Total	1824.58	100.0

Table 4: Time spent in various subtasks (non-overlapping) while simulating DRC terrain crossing, with Pose Finding horizon of 3 steps.

ing body poses over a longer horizon (thus requiring fewer body motions), and time spent planning and executing (more or fewer body shift motions, depending on the output of the Pose Finding algorithm).

We will now compare these new simulation results to the results previously presented in Satzinger et al. (2014). The previous results were generated under conditions which differ from the new simulations in the following ways. The Pose Finding horizon was four steps. No collision geometry was provided by the terrain model. The 'Other' timing category included some time spent for the user to set up the simulation environment before a simulation, which has been excluded in the new simulations. The RRT timeout has been increased (because the planning problem is made more complex by collision geometry). The distance to the goal has been increased to 10 meters (from 9 meters) in order to ensure the robot completely clears the terrain. Additional candidate footholds have been provided on all levels above ground level, in order to provide more possible routes. Finally, the software used for the new trials is a newer version and contains many small adjustments and improvements that do not change the overall approach, but may affect execution time.

The data in Table 3 provides the most direct comparison (with the caveats described above) to the previous simulation results. The overall time has increased from 1293.5 seconds to 1923.51 seconds. This is primarily due to increased time spent performing RRT-Connect searches and executing the resulting motions. We attribute this to the greater complexity of planning around the collision geometry. The resulting plans are also longer, and therefore require more time to execute. The time spent performing the Foothold Graph Search has increased from 169.9 seconds to 221.60 seconds. We should note that the foothold density has roughly doubled, since there are now two candidate footholds on top of each cinderblock stack (previously there were two only on the highest level). It is also simply more computationally expensive to check for terrain collisions than not to do so, and this will necessarily slow down planning by some amount. The time spent in the Pose Finding algorithm has also increased from 192.4 seconds to 377.50 seconds. Because the terrain collision geometry rules out many body poses that would otherwise be feasible, the Pose Finder must resort to less efficient solutions which require more frequent body shifts between steps. By design, the planner tries to find body solutions with the assumption that efficient solutions (i.e., where a single body pose is feasible for several successive steps) are possible, and only after failing to find them does it attempt to find increasingly inefficient solutions (i.e., where body poses differ for several successive steps, requiring the insertion of body-shift motions in between steps). It may be possible to improve this performance, for example by modifying the Pose Finding algorithm to have more pessimistic assumptions from the beginning. The length of the Pose Finding planning horizon can also be adjusted, which we will discuss next.

Table 4 shows data generated with a shorted Pose Finding horizon (3 steps). This reduced horizon substantially reduces the time spent finding body poses (from 377.50 seconds to 156.03 seconds). In exchange, less efficient plans are generated requiring an increased number of body shifts in between steps. This increases the RRT-Connect planning time from 571.73 seconds to 609.76 seconds, and the execution time from 972.69 seconds to 1057.97 seconds. But, on balance it is 98 seconds faster to use a horizon of 3 steps. This is, of course, relative to the speed of our computer. All other things being equal, if our computer was roughly twice as fast, then it would be better to have a horizon of 4 steps.

Although simulated results suggest longer planning horizons result in more efficient plans, practical ex-

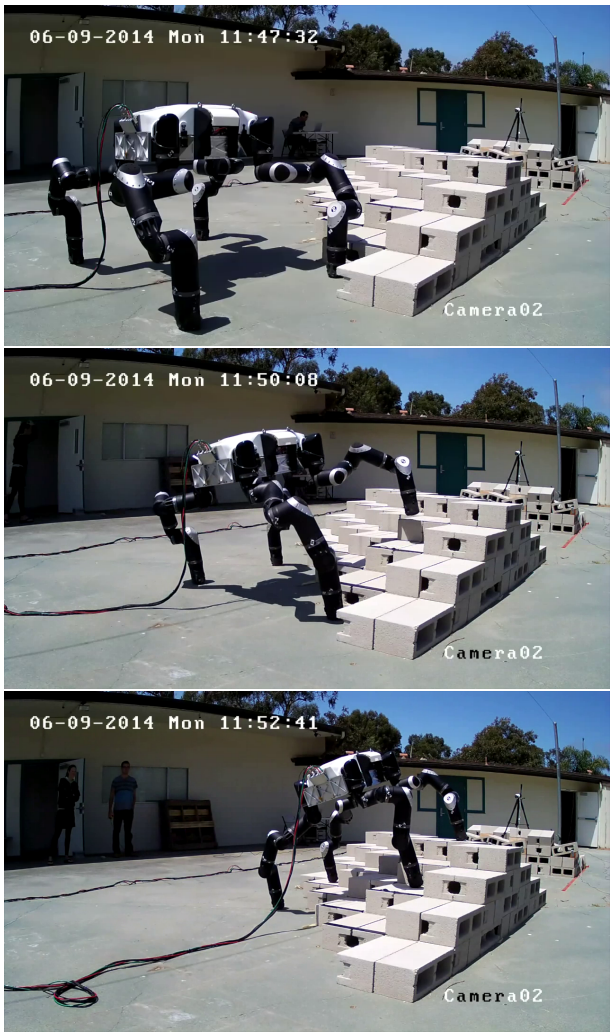

Figure 10: We used our approach with real robot hardware in an outdoor environment. Results were similar to simulations, but we were substantially limited by issues related to perception and localization. The system was able to proceed as far as shown before the robot's pose estimate error became too large for it to plan successfully. Before this occurred, it planned and executed 19 steps fully autonomously.

perience instead suggests that the planning horizon must be matched with the robot capabilities. Having a larger planning window becomes inefficient if sufficiently large errors in expected foothold locations force the robot to replan often. Figure 10 shows frames from a video of one of our experiments with robot hardware in an outdoor environment. We were substantially limited by the perception and localization available on the prototype RoboSimian hardware and software available at the time of the experiments, and, as a result, we could rarely complete 4 step motion plans without replanning. Thus, in practice, a 3 step planning horizon was more efficient. Specifically, the terrain map generated onboard the robot was not of high enough fidelity to support meaningful collision checking. To overcome this, the operator manually aligned a high fidelity collision geometry model with the low-fidelity map data. The collision geometry model is similar to the one used in the previously discussed simulations, but with only the first 'hill' present (see Figure 11) and with additional collision geometry around the cinderblocks in order to partially compensate for localization errors (by forcing a larger margin around the terrain, particularly the corners). Despite this workaround, the robot's pose estimate accumulated relatively large errors over time that eventually prevented planning.

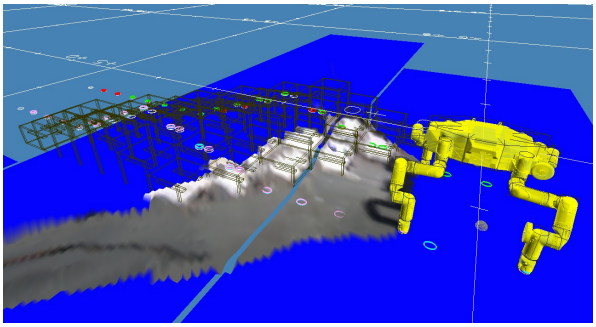

Figure 11: The world map available at the beginning of the experiment presented in Figure 10, and the collision geometry model that the operator manually aligned with the world map.

.

As a result, we did not successfully traverse the entire terrain. However, we were able to reach the top of the first 'hill' fully autonomously (taking 19 steps) before pose estimate errors accumulated sufficiently

to cause the robot to believe it was in collision with the terrain. At this point, although it was possible to manually re-align the world model and continue planning from the A* phase, we terminated the experiment. These problems would be ameliorated by better mapping and localization. Future hardware and software upgrades are expected to be made by our colleagues at JPL in order to address these issues [4].

It is a potential criticism of this work that we are planning on a model of the environment and therefore require some level of modeling and localization accuracy in order for it to be useful. However, as previously discussed, our approach has several mechanisms for dealing with discrepancies between the model and reality. In practice, we were able to walk autonomously for several minutes and take 19 steps in an outdoor environment despite significant mapping and localization errors. If a sufficiently accurate mapping and localization system is available, then our approach is able to take advantage of it in order to plan in complex environments. Section 3.2 shows simulation results in some such complex environments, where a simpler, more reactive walking system (e.g., the system used on RoboSimian in the 2013 DRC Trials) might collide with the terrain.

3.4 DARPA Robotics Challenge

Our methods have been tested in simulation and in an outdoor testing environment in preparation for the DARPA Robotics Challenge (DRC). However, the most significant (and disappointing) result for us was that drift in perception of our world map made careful foothold planning a significant challenge. In practice, RoboSimian performed the locomotion task during the December 2013 DRC Trials using RRT-Connect for a set of heuristic footholds planned blindly on terrain, using force feedback to detect ground contact. Results were still good enough to place 5^{th} and qualify for the final competition, scheduled for mid 2015.

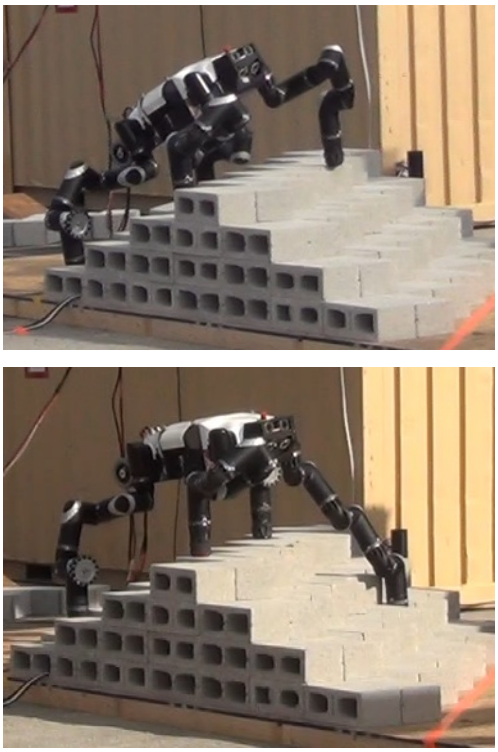

Figure 12: RoboSimian pitches its body and stretches to a near-singular configuration to traverse terrain at the DRC.

4 Comparison to Synthetic Time-Optimal Reference

RRT-Connect is probabilistically complete, but does not generally produce optimal solutions. Although sub-optimal solutions are acceptable in many situations (especially when computing truly optimal solutions is intractable), we are interested in determining how sub-optimal our solutions are in practice as an evaluative technique. Of course, a solution being con-

[4] We are extremely grateful to our colleagues at JPL for generously allowing us to have access to RoboSimian. We are emphasizing the difficulties we encountered with perception and localization in order to highlight how real world effects impact our approach. For example, the robot cannot map the far side of the hill until it has approached the summit thus it does not make sense to have a planning horizon which extends over terrain with a high degree of uncertainty. Therefore, it is likely that the robot should dynamically vary the planning horizon based upon the maps currently available.

sidered optimal is always contingent on the choice of cost function. For our purposes, we desire to minimize the time it takes for the robot to execute the plan.

RoboSimian's low-level control algorithms use trapezoidal interpolation with velocity and acceleration limits v_{max} and a_{max} to track a position reference. These limits are identical for all joints. RRT-Connect generates a sequence of joint angle waypoints without timestamps. Then, a post-processing step assigns timestamps to each waypoint greedily in a single pass. This algorithm assumes maximum accelerations (subject to velocity saturation) from waypoint to waypoint, in order to give a reference trajectory that is physically achievable under v_{max} and a_{max}. This does not preclude the existence of a reference trajectory with the same waypoints but different timestamps that is also achievable, but takes overall less time because the accelerations are less aggressive.

Due to time and resource limitations, we elected to use the greedy timestamp assignment algorithm for this analysis and our experimental testing. We are currently evaluating post-processing algorithms to "smooth" the trajectories found by the RRT in both space and time with the goal of both moving faster and avoiding agressive accelerations which are strongly correlated with tracking faults.

4.1 Synthetic Reference

Despite the limitations described above, we can generate certain trajectories that are provably time-optimal out of any achievable choice of timestamps, respecting v_{max} and a_{max}. One controlling joint trajectory $q_c(t)$ will accelerate from rest at a_{max} until the velocity saturates at v_{max} and then decelerate at $-a_{max}$ to rest, taking t_c seconds to perform the motion. The remaining non-controlling joints can follow trajectories $q_i(t)$ as long as those trajectories can be completed in less than t_c seconds so they do not become the controlling joint. For simplicity, we limit motion to a single swing limb (keeping other joint angles constant), although with care, time-optimal whole-body trajectories could be generated through a similar process. For additional simplicity, we generate our waypoints by linear interpolation between the initial and final poses. Figure 13 shows the start (red) and end (blue) poses for our synthetic trajectory, along with the resulting, curved path of the end effector. Timestamps are chosen in order to respect the velocity and acceleration limits of the robot, which are nominally 1.2 rad/s and 4.7 rad/s^2, respectively.

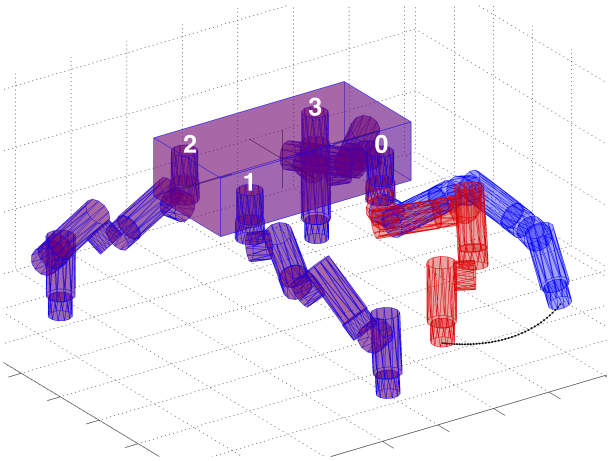

Figure 13: An example of a time-optimal swing leg trajectory, used in benchmarking RRT statistics. All joint are frozen thoughout the motion, except the swing leg, and when swing-leg joints move in a straight-line trajectory in joint space, the end effector moves in a smooth curve. Our RRT search allows for additional joint motions of the other limb. While an infinite number of equally-optimal solutions exist, it is mathematically impossible to out-perform the synthetic trajectory.

We can use the techniques previously discussed in this paper to generate plans between the initial and final poses from the time-optimal synthetic reference trajectory ($q(0)$ and $q(t_c)$). Then, we can compare these trajectories to the synthetic reference trajectory and draw conclusions about the performance of the algorithm.

Because we are considering a plan with a single swing limb, we have three choices of dominant limb. This gives three distinct formulations of the same problem. In addition to comparing the solutions to the synthetic reference as an absolute lower bound

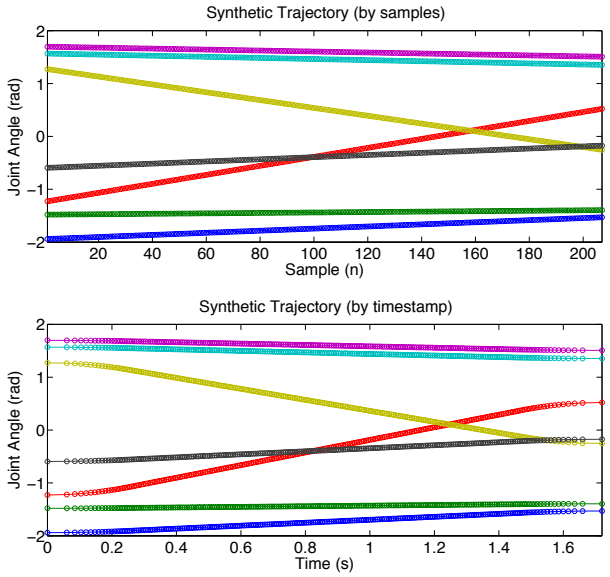

Figure 14: (top) Synthetic trajectory is generated by linear interpolation between initial and final position. (bottom) Timestamps for each waypoint are assigned to respect velocity and acceleration limits. Joint angles are only shown for the swing limb, as all other joints are held constant.

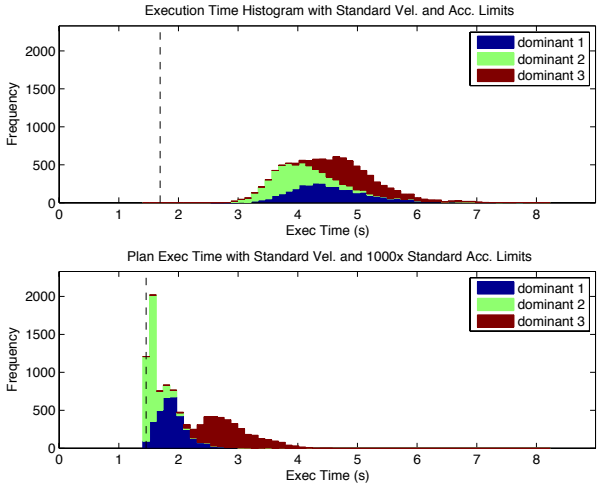

Figure 15: (top) Stacked histograms showing the distibution of execution times of 10000 random trials, separated by dominant limb. (bottom) Same as above, but reprocessed with 1000 times greater acceleration limit, showing that accelerations are the dominant reason for suboptimal solutions. The dashed vertical line shows the execution time of the synthetic reference trajectory with the appropriate velocity and acceleration limits.

on time, we can compare these three formulations in relation to each other. We performed a total of 10000 random trials with different random seeds, alternating between the three choices of dominant limb. All trials successfully resulted in a solution. Figure 15 shows the results of these random trials.

Figure 15 shows the distribution of the amount of time it will take RoboSimian to execute the plans, both with normal acceleration limits, and with greatly exaggerated acceleration limits. In both cases, the choice of dominant limb greatly affects the expected value of the execution time. In the top plot, with normal acceleration limits, there is a significant discrepancy compared to the execution time of the synthetic reference trajectory (dashed vertical line). However, in the bottom plot, when the acceleration limit is increased by a factor of 1000, many of the solutions are nearly optimal, and the overall distribution is much closer to the optimal solution. This suggests that our planners are generating unsmooth paths with frequent changes in direction. However, the choice of dominant limb still makes a large difference in the execution times, with dominant limb 2 giving nearly optimal solutions most of the time, and dominant limb 3 giving solutions taking about twice as long.

Figure 16 shows the relationship between execution time and the number of nodes (waypoints) in the finished path. This is the same dataset as in Figure 15, to give an idea of the horizontal distribution of points. Dominant limb 1 (blue) shows a tendency to generate paths with more nodes.

One question of interest is the amount of CPU time required to generate a solution. We use the total number of feasibility checks performed during a search as a proxy for CPU time, with the assumptions that feasibility checks all take the same amount of time, feasibility checking dominates the computa-

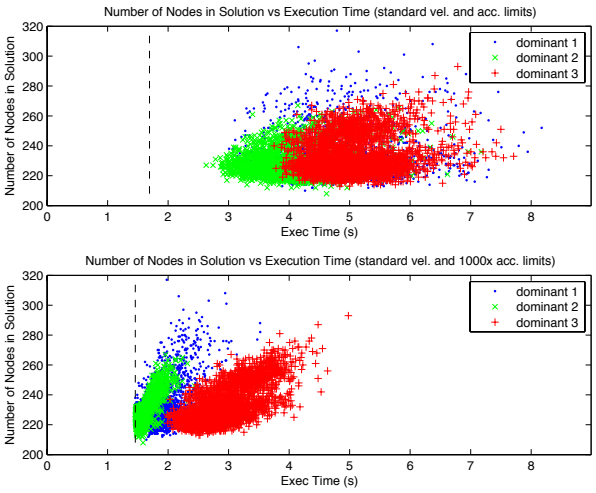

Figure 16: (top) Scatter plot of number of nodes in the solution path vs the total execution time, separated by dominant limb. (bottom) Same, but with 1000x the normal acceleration limit.

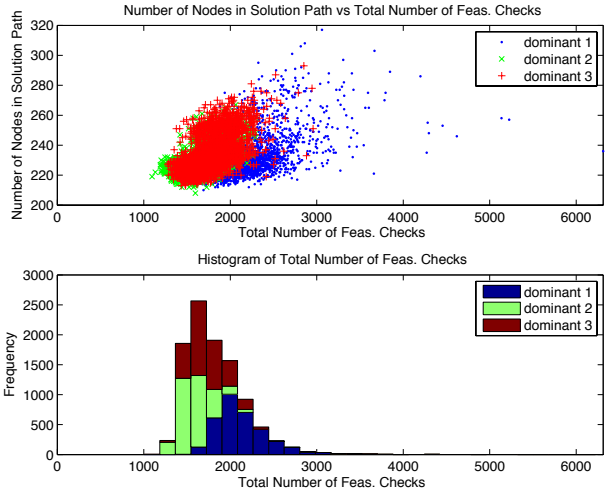

Figure 17: Total number of feasibility checks serves as a proxy for overall planning time.

tion, and that CPU time measurements are subject to errors due to thread scheduling and transient background loads on the computer as well as the overall speed of the computer. Figure 17 shows this data as a scatterplot and a histogram. Although dominant limb 2 (green) continues to be the best performing choice, dominant limb 1 (blue) and dominant limb 3 (red) have switched places. The distribution of CPU times required for dominant limb 3 (red) is nearly comparable with dominant limb 2 (green), with dominant limb 1 (blue) clearly much worse. Indeed, dominant limb 1 has the greatest mean number of feasibility checks, and a long tail of very expensive searches.

4.2 Which planner should we prefer?

Should we prefer a planner that produces faster solutions more slowly, or slower solutions more quickly? If we are to plan and then execute serially, we are concerned with the sum of both the planning time t_{plan} and the execution time t_{exec}.

$$t_{total} = t_{plan} + t_{exec} \tag{3}$$

Assuming a constant time per feasibility check t_{feas} (seconds) and a number of feasibility checks n_{feas}, we can make the approximation

$$t_{plan} = t_{feas} \cdot n_{feas}, \tag{4}$$

and therefore express the total time as

$$t_{total} = t_{feas} \cdot n_{feas} + t_{exec}. \tag{5}$$

We assume that n_{feas} is drawn from the distributions shown in Figure 17 and t_{exec} is drawn from the distributions shown in Figure 15, while t_{feas} is a constant that depends on the overall speed of the computer and software being used. Figure 18 shows the expected value of t_{total} as a function of t_{feas}, using the mean values of t_{exec} and n_{feas} that we observed in our data for each choice of dominant limb.

No matter the value of t_{feas} (which cannot be negative), it turns out that, in this experiment, dominant limb 2 (green) will always have the shorter expected total time. Ignoring limb 2 momentarily, all eyes are on the race for second place. Whether dominant limb 1 (blue) or limb 3 (red) is preferable now depends on the speed of the computer being used. If t_{feas} is less than approximately 10^{-3} seconds then dominant limb 1 is better, otherwise dominant limb 3 is better. (The value of t_{feas} that the author observes on their

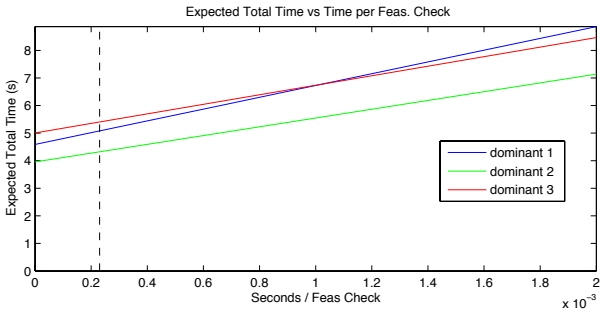

Figure 18: Expected total time depends on feasibility checking speed.

computer is approximately $2 \cdot 10^{-4}$ seconds.)

5 Conclusions and Discussion

Our approach blending **RRT**-Connect and IK **T**ables (**R2T2**) provides a computationally practical kinodynamic planning method with a high rate of success. In an example of trajectory planning across extreme terrain used in the 2013 DRC trials, planner and execution time were approximately equal. This suggests that improvements in planning speed (through more efficient allocation of computational resources among an ensemble of randomized planning algorithms) or in the execution time (through better trajectory smoothing) are both important avenues for future improvements.

We note that it would likely be desirable to allocate at least some computational cycles toward RRT*, to seek "optimal" solutions directly. However, we also note that as environments become increasing complex and stochastic, it becomes increasingly difficult for a programmer to accurately define what cost function should actually be optimized, to achieve fast and reliable locomotion. In practice, we anticipate that access to a family of feasible solutions will allow an operator to preview the best-ranked trajectory found within time limits, as a sanity check. Humans have contextual knowledge (e.g., "I can brush by the bush – but I don't want to risk close clearance with the brick wall...") that is not easily encapsulated into autonomy. A secondary advantage of producing a family of randomized plans is to allow an operator to select among closely-ranked but qualitatively dissimilar solutions. Finally, a third and perhaps most practical application is in debugging and tuning definitions of cost functions to evaluate a plan. Features can be extracted from each plan, and cost functions can be adapted based on human ranking. In practice, we are curious to determine which of these three is most useful in planning for the DRC tasks.

We conclude by noting two key potential limitations specific to the current configuration of RoboSimian. First, perception is a strong requirement for real-world implementation on RoboSimian. At the time of writing, RoboSimian currently relies on three pairs of forward-facing stereo cameras to view front limbs; however, rear limbs are often well over a meter behind the front limbs, requiring an accurate world map of previously viewed terrain. Work is currently underway to upgrade the robot with LIDAR sensing and additional camera pairs.

It is also apparent that our trajectories take longer than necessary due to a combination of low joint acceleration limits in the hardware and unnecessary high-frequency content in the joint trajectories from the randomized planner, suggesting that trajectory smoothing techniques such as Hauser & Ng-Thow-Hing (2010) could provide a significant speedup.

6 Future Work

There exists a large and growing family of randomized planning algorithms. As computational power increases with Moore's Law, it will become increasingly practical to allow such algorithms to compete in an open market during both off-line and real-time planning.

Optimal investment and betting strategies quantify risk and bet according to a certainty equivalent of expected value, which discounts expected reward as a function of uncertainty in a mathematically elegant way, now well-known as the Kelly criterion (Kelly (1956)). When a family of betting options exist, covariance must also be considered, and optimal solutions hedge by including investments that "cover the bases" for a variety of outcomes (Browne & Whitt

(1996), Cecchetti et al. (1988)). Similarly, the best way to divide computing resources should seek to reduce risk by biasing different parallel searches in different ways. We are currently investigating the development of effective, adaptive methods to do so as future work.

We are currently integrating a trajectory smoothing technique based on the methods presented in Hauser & Ng-Thow-Hing (2010) to improve the performance of our RRT planners in terms of trajectory execution time. This technique extends the RRT along time-optimal curves rather than straight-line trajectories as in the original algorithm (Lau & Byl (2015)). We have tested this implementation on toy examples with positive results, and we hope to have the same measure of success on Robosimian's platform.

Acknowledgment

This work is supported by JPL NASA Contract #1471138, which is a subcontract award for the DARPA Robotics Challenge (DRC). The authors would also like to thank the entire RoboSimian team for their efforts in designing (and debugging) the robotic system hardware and software.

References

Bouyarmane, K. & A., K. (2012), 'Humanoid robot locomotion and manipulation step planning', *Advanced Robotics (Int. J. of the Robotics Society of Japan), Special Issue on the Cutting Edge of Robotics in Japan 2012* **26**(10), 1099–1126.

Bretl, T. W. (2005), Multi-step motion planning: Application to free-climbing robots, PhD thesis, Citeseer.

Browne, S. & Whitt, W. (1996), 'Portfolio choice and the bayesian kelly criterion', *Advances in Applied Probability* pp. 1145–1176.

Byl, K. (2008), Metastable legged-robot locomotion, PhD thesis, MIT.

Byl, K., Byl, M. & Satzinger, B. (2014), Algorithmic optimization of inverse kinematics tables for high degree-of-freedom limbs, *in* 'Proc. ASME Dynamic Systems and Control Conference (DSCC), accepted'.

Byl, K., Shkolnik, A., Prentice, S., Roy, N. & Tedrake, R. (2009), Reliable dynamic motions for a stiff quadruped, *in* 'Proc. International Symposium of Robotics Research (ISER) 2008', Vol. 54, pp. 319–328.

Cecchetti, S. G., Cumby, R. E. & Figlewski, S. (1988), 'Estimation of the optimal futures hedge', *The Review of Economics and Statistics* pp. 623–630.

Diankov, R. (2010), Automated Construction of Robotic Manipulation Programs, PhD thesis, Carnegie Mellon University, Robotics Institute.

Donald, B. R. & Xavier, P. (1995), 'Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open-chain manipulators', *Algorithmica* **14**(6), 480—530.

Donald, B., Xavier, P., Canny, J. & Reif, J. (1993), 'Kinodynamic motion planning', *Journal of the ACM* **40**(5), 1048—1066.

Felner, A., Kraus, S. & Korf, R. (2003), 'Kbfs: K-best-first search', *Annals of Mathematics and Artificial Intelligence* **39**(1-2), 19–39.

Hart, P. E., Nilsson, N. J. & Raphael, B. (1968), 'A formal basis for the heuristic determination of minimum cost paths', *Systems Science and Cybernetics, IEEE Transactions on* **4**(2), 100–107.

Hauser, K. (2013), Large motion libraries: Towards a "google" for robot motions, *in* 'Proc. Workshop on Robotics Challenges and Vision', pp. 5–8.

Hauser, K., Bretl, T. & Latombe, J.-C. (2005), Non-gaited humanoid locomotion planning, *in* 'Proc. Int. Conf. on Humanoid Robots', IEEE, pp. 7–12.

Hauser, K. & Ng-Thow-Hing, V. (2010), Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts, *in* 'Robotics

and Automation (ICRA), 2010 IEEE International Conference on', pp. 2493–2498.

Jetchev, N. & Toussaint, M. (2009), Trajectory prediction: Learning to map situations to robot trajectories, *in* 'Proc. Int. Conf. on Machine Learning', pp. 449–456.

Kelly, J. L. (1956), 'A new interpretation of information rate', *Information Theory, IRE Transactions on* **2**(3), 185–189.

Kolter, J. Z. & Ng, A. Y. (2011), 'The Stanford LittleDog: A learning and rapid replanning approach to quadruped locomotion', *The International Journal of Robotics Research (IJRR)* **30**(2), 150–174.

Kuffner, J. J. & LaValle, S. M. (2000*a*), RRT-connect: An efficient approach to single-query path planning, *in* 'Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on', Vol. 2, IEEE, pp. 995–1001.

Kuffner, J., Kagami, S., Nishiwaki, K., Inaba, M. & Inoue, H. (2002), 'Dynamically-stable motion planning for humanoid robots', *Autonomous Robots* **12**(1), 105–118.

Kuffner, J. & LaValle, S. (2000*b*), RRT-connect: An efficient approach to single-query path planning, *in* 'Proc. IEEE International Conference on Robotics and Automation (ICRA)', Vol. 2, pp. 995–1001 vol.2.

Kuwata, Y., Fiore, G. A., Teo, J., Frazzoli, E. & How, J. P. (2008), Motion planning for urban driving using RRT, *in* 'Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on', IEEE, pp. 1681–1686.

Lau, C. & Byl, K. (2015), 'Smooth RRT-connect: An extension of RRT-connect for practical use in robots', *Accepted for IEEE/Technologies for Practical Robotic Applications (TePRA) 2015* .

Satzinger, B. & Byl, K. (2014), 'More solutions means more problems: Resolving kinematic redundancy in robot locomotion on complex terrain', *Submitted to IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* .

Satzinger, B. W., Lau, C., Byl, M. & Byl, K. (2014), Experimental results for dexterous quadruped locomotion planning with robosimian, *in* 'Proc. International Symposium of Robotics Research (ISER)'.

Schaal, S. & Atkeson, C. G. (2010), 'Learning control in robotics', *Robotics & Automation Magazine, IEEE* **17**(2), 20–29.

Stolle, M. & Atkeson, C. G. (2006), Policies based on trajectory libraries, *in* 'Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on', IEEE, pp. 3344–3349.

Vernaza, P., Likhachev, M., Bhattacharya, S., Chitta, S., Kushleyev, A. & Lee, D. D. (2009), Search-based planning for a legged robot over rough terrain, *in* 'Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)', IEEE, pp. 2380–2387.

Zucker, M., Ratliff, N., Stolle, M., Chestnutt, J., Bagnell, J. A., Atkeson, C. G. & Kuffner, J. (2011), 'Optimization and learning for rough terrain legged locomotion', *The International Journal of Robotics Research* **30**(2), 175–191.