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 a method that exploits a reduced-dimension RRT search, constraining a subset of limbs to an inverse kinematics table. We 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.

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

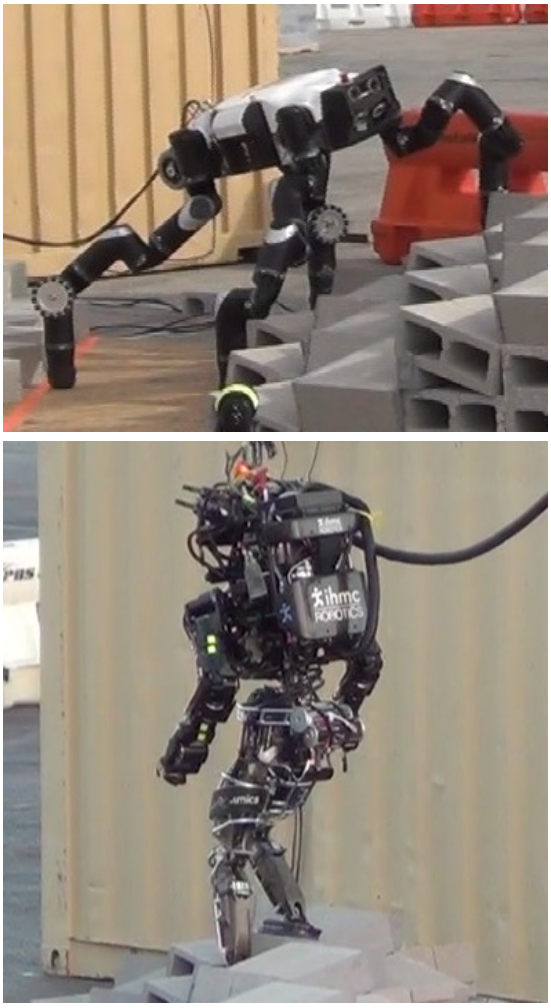

Figure 1: RoboSimian (top) and the Atlas (bottom). The quadruped can balance more easily, but its high-DOF limbs complicate planning.

*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

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 of seven joints per limb. On the other hand, the robot is also required to walk over rough terrain such as the one pictured in (Fig. 1).

Advantages of using RoboSimian for rough terrain locomotion include (1) its large, stable base of support, and (2) existence of redundant kinematic solutions, toward avoiding collisions with complex terrain obstacles. 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, and (3) having rear limbs outside the field of view requires adequate perception with respect to a world map. In our results, 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 real-world testing indicates that perception limitations provide the greatest challenge in real-world implementation. 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.

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.

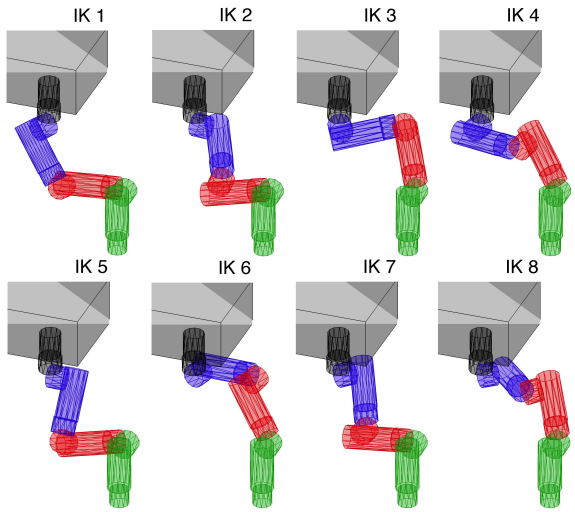

Figure 2: Redundant inverse kinematic (IK) solutions for RoboSimian.

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)), which presents its own set of (adaptive) computing resource allocation problems, to cover relevant dynamic motions with adequate fidelity. 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 highly dimensionality, with either 9 or 16 angular degrees of freedom at each node, yet 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 pre-computed 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 motion planning approach, R2T2, which blends "RRTs with Tables" (of IK solutions) for tractability, and 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

Our approach begins with a set of candidate footholds locations. We use a graph search to find a specific foothold plan, consisting of a series of steps that will be taken. Because our approach allows body motion during a step, we next search for body poses for the initial and final pose for each step, over a hori-

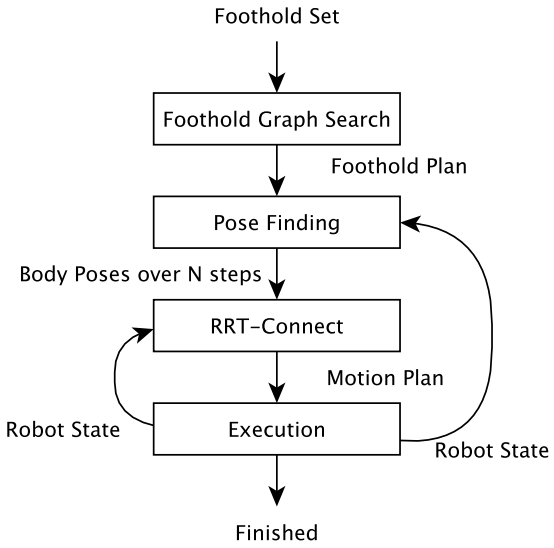

Figure 3: Planning phases in our approach

zon of the next several steps. These poses are passed to our RRT-Connect implementation, which finds a path between the steps, respecting static stability, kinematic feasibility, and collision constraints. The motion plan is then executed on the robot (or a simulation). If more steps remain to be planned, control passes back to the RRT-Connect planner. Otherwise, control passes back to the pose finder. This process repeats until an error has occurred, or the goal has been reached. These phases 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, i.e. 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 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 a focus of this paper, but some performance data will be presented in Section 3.2 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, a node identifies a particular stance among the set of possible footholds. Our cost heuristic is the linear distance between the centroids of the footholds of two nodes (or between a node and the goal location). We expect future work to modify this cost heuristic to reflect other planning preferences, such as the preferential use of certain footholds over others.

The search can enforce a gait order or allow free gaits with steps in any order. In general, free gaits increase the complexity of the search because each node has more potential child nodes. However, for the DRC terrain the free gait search is advantageous because it enables the negotiation of difficult parts of the terrain. An experiment with gait order fixed did not find a solution.

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, many such solutions result in joint reconfigurations once continuous limb motions are planned. Also, ikfast does not distinguish among "families" of kinematics when giving joint solutions; our grouping of solutions depends upon a customized but relatively slow IK solver, written in-house at UCSB. 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 3 stance legs and 1 swing leg, we set a nominal body orientation (roll, pitch, and yaw) heuristically, to match the underlying foothold locations and heights. Then, we search numerically over a 6-DOF $7 \times 7 \times 9 \times 5 \times 5 \times 5$ (x, y, z, roll, pitch, yaw) grid of potential body poses centered on the heuristic pose. 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 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 number of dimensions substantially when compared to a naive approach while simultaneously addressing kinematic closure of the stance legs and allowing the use of all degrees of freedom on the swing limb in order to allow dexterous motion.

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 Katie Byl (2014).

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

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

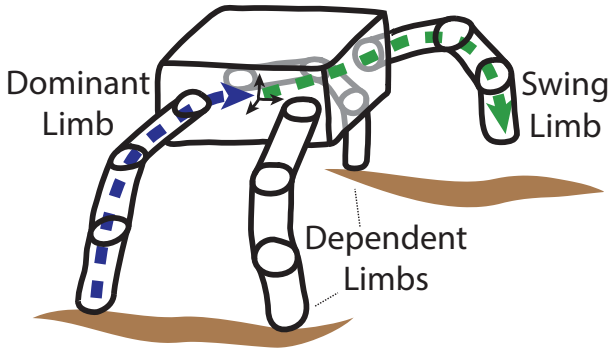

Figure 4: Cartoon sketch from Satzinger & Byl (2014) illustrating the design of our RRT-Connect configuration space parameterization for RoboSimian.

Figure 4 illustrates this approach. During a swing motion, the complete pose of the robot can be specified by the 7 joints 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.

Algorithm 1 Procedure for calculating the full pose (consisting of q and C_{body}^{world}). $C_{f_i}^{world}$ give 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 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 RRTC 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 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 experiments that demonstrate the capabilities and benefits of our approach to trajectory planning for Robosimian.

3.1 Feasible Step 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 sys-

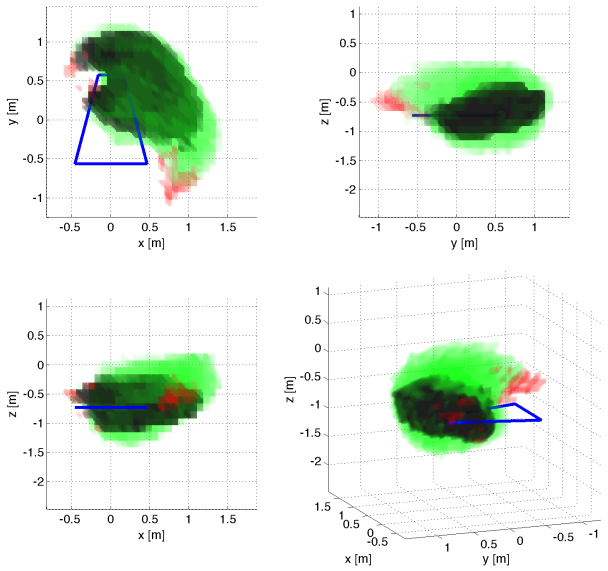

Figure 5: Volumetric rendering of the reachable volume from several perspectives. Black shows the volume S, and green shows the relative complement of S in D (showing the region reachable only with body motion durind a step). Red shows regions contained in S and D where the RRT solvers were not able 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.

tem 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 Simulated Traversal of DRC Terrain

We performed a simulation experiment 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 the highest terrain level in order to allow the A* planner to find a feasible solution. The goal location was set 9 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 6). We also did not consider the orientation of the footholds, which are on locally sloped surfaces for the second half of the terrain.

Although collision detection with the terrain is used during RRT planning, integration into body

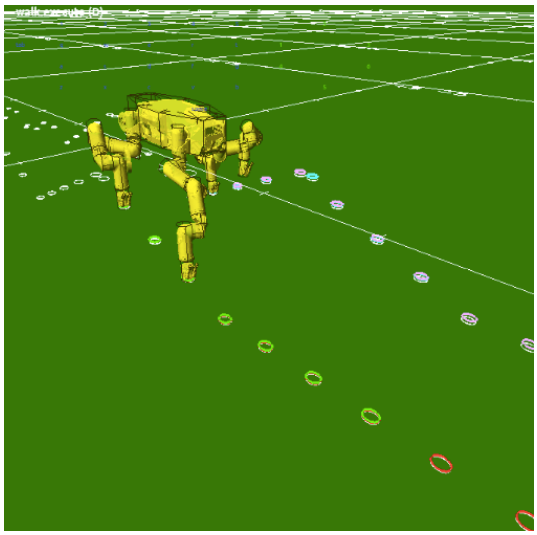

Figure 6: RoboSimian straddles the second ridge on a simulated crossing of the DRC terrain

pose finding and A* planning is not complete as of writing. If the initial and final body poses chosen for a step happen to be in collision with the terrain, the RRT solvers will be unable to find a solution. Therefore, for this simulation, a terrain collision model was not used. However, checks for self collisions between different parts of the robot are always made.

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. Pose finding was performed every four steps, and chose 8 body poses (for the beginning and end of each step). The pose finding search used 8 threads.

RRT based pathfinding algorithms were used to find feasible and stable paths between the initial and final poses given by each step or body movement. Our RRT 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 solver three (or four) times with each choice of dominant limb in turn. 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.

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 1: Time spent in various subtasks (non-overlapping) while simulating DRC terrain crossing.

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, call the RRT 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 planned sequence of footholds. Therefore, we postpone calling the RRT solvers until just before the motion is to be executed so that motions are planned with the most up to date information possible.

3.3 Obstacle Avoidance

In the prior simulation, we did not include a terrain collision model. In order to demonstrate the capability to avoid complex obstacle geometry, we generated a set of obstacles for Robosimian to avoid when taking one step forward. Boxes were arranged in the formation specified in Table 2 and shown in Figure 7.

A foothold plan was generated to move Robosimian's back left leg forward 0.50 meters. Such a situation would likely be encountered on rough terrain, especially those involving fallen debris.

Because our pose finding algorithm does not yet account for terrain collisions, the initial and final body poses were chosen manually to keep Robosimian's center of pressure within the support polygon during swing. A rotation was added to the final body pose to allow for body adjustments required to step over the obstacles. The footholds for the non-swing limbs were set in the crawl gait formation with end effectors approximately one meter apart in both the x- and y-directions. The end effector of the swing leg was set 0.10 meters above its nominal walk pose to account for contact behavior.

A swing trajectory to maneuver Robosimian's back left leg over the obstacles was successfully found using RRT-Connect. The generated trajectory included 241 waypoints with a playback time of 12.13 seconds. Such complicated maneuvers would not be possible using only IK table solutions since many of the poses required of Robosimian to swing its leg over the boxes are atypical. As demonstrated by this example, when traversing rough terrain, many situations require unique and complicated movements beyond those that are reasonable to store in an IK table. The combined RRT plus IK tables approach allows us to generate feasible solutions for unpredictable situations within an acceptable time frame. Computation time for the RRT to generate the swing trajectory depicted in Figure 7 was 17.12 seconds. The RRT required 8227 nodes from the starting foothold and 7912 nodes from the goal foothold for a total of 16139 nodes.

Another important aspect of this approach to solving for swing trajectories is the flexibility of the body pose during swing. The movement of the body is integral to the RRT's ability to find a solution for the foothold as the contortions required of the swing limb without movement of the body would result in collisions between joints and the body.

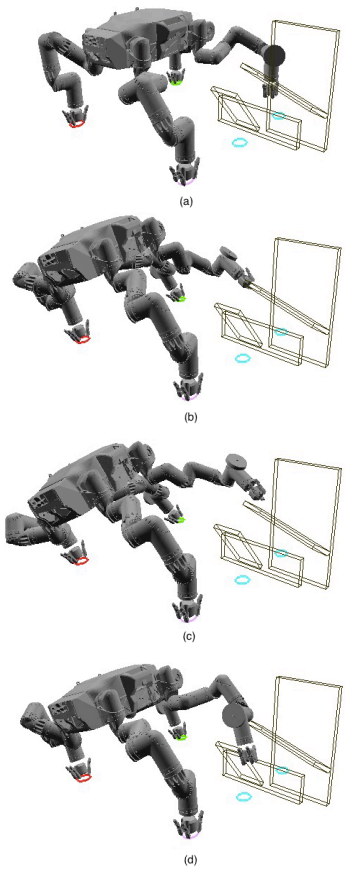

Figure 7: Demonstration of an RRT generated swing trajectory with four boxes arranged as obstacles. RoboSimian's back left limb successfully avoids the boxes as it swings to its next foothold 0.50 meters forward in the x-direction.

3.4 DARPA Robotics Challenge

Our methods have been tested for short foothold plans in lab on RoboSimian 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 DRC using RRT-Connect for a set of heuristic footholds planned blindly on terrain, using force

	Position (meters)			Rotation (°)			Size (meters)		
	x	y	z	roll	pitch	yaw	x	y	z
Body - entry	0.10	0.10	0.00	0	0	0	—	—	—
Body - exit	0.10	0.10	0.00	10	-5	10	—	—	—
Foothold - FR	0.51	0.51	0.65	—	—	—	—	—	—
Foothold - BR	-0.51	0.51	0.65	—	—	—	—	—	—
Foothold - BL (start)	-0.51	-0.51	0.55	—	—	—	—	—	—
Foothold - FL	-0.51	-0.51	0.65	—	—	—	—	—	—
Foothold - BL (end)	-0.01	-0.51	0.55	—	—	—	—	—	—
Box 1	-0.22	-0.50	0.56	0	0	0	0.04	0.61	0.19
Box 2	-0.29	-0.75	0.26	0	0	0	0.04	0.41	0.76
Box 3	-0.22	-0.64	0.31	-10	30	5	0.20	0.61	0.03
Box 4	-0.27	-0.30	0.56	0	50	0	0.49	0.20	0.03

Table 2: Body poses, foothold locations, and box poses and sizes for the RRT demonstration in Figure 7. The acronyms FR, BR, BL, and FL refer to the front right limb, back right limb, back left limb, and front left limb, respectively.

feedback to detect ground contact. Results were still good enough to place 5^{th} and qualify for the final competition, scheduled for mid 2015.

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 considered 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.

In the interest of being useful to our team in the DARPA Robotics Challenge, we are constrained to work with the robotic system we have, rather than counterfactual versions with more sophisticated low-level control and interpolation. For this reason, our definition of "the time it takes for the robot to execute the plan" (and therefore our notion of optimality) uses the greedy timestamp assignment algorithm described above.

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

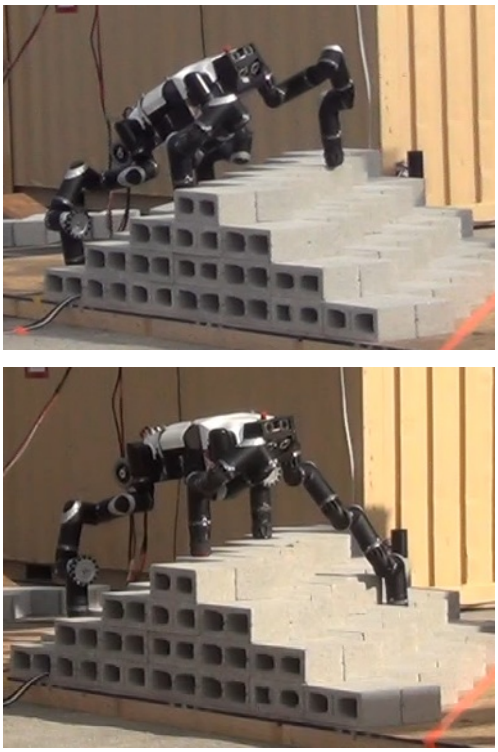

Figure 8: RoboSimian pitches its body and stretches to a near-singular configuration to traverse terrain at the DRC.

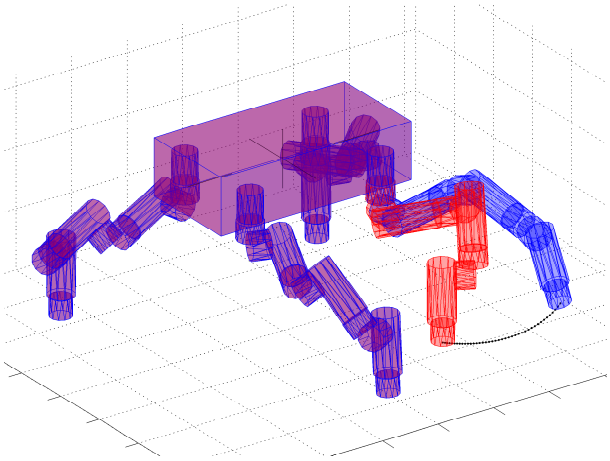

Figure 9: An example of a time-optimal swing leg trajectory, used in benchmarking RRT statistics. All joint are frozen thoughout the motion, except the swing leg, and when swing-leg joints move in a straight-line trajectory in joint space, the end effector moves in a smooth curve. Our RRT search allows 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.

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 9 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.

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 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 11 shows the results of these random trials.

Figure 11 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

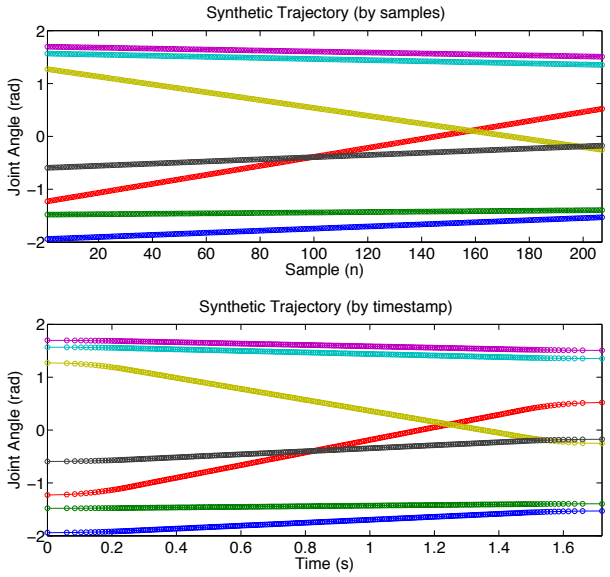

Figure 10: (top) Synthetic trajectory is generated by linear interpolation between initial and final position. (bottom) Timestamps for each waypoint are assigned to respect velocity and acceleration limits. Joint angles are only shown for the swing limb, as all other joints are held constant.

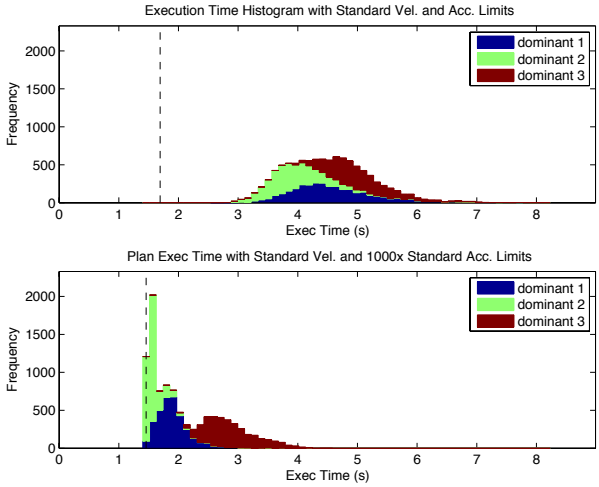

Figure 11: (top) Stacked histograms showing the distibution of execution times of 10000 random trials, separated by dominant limb. (bottom) Same as above, but reprocessed with 1000 times greater acceleration limit, showing that accelerations are the dominant reason for suboptimal solutions. The dashed vertical line shows the execution time of the synthetic reference trajectory with the appropriate velocity and acceleration limits.

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. 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.

This suggests that our planners are generating unsmooth paths with frequent changes in direction.

Figure 12 shows the relationship between execution time and the number of nodes (waypoints) in the finished path. This is the same dataset as in Figure 11, 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 computation, 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 13 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 dom-

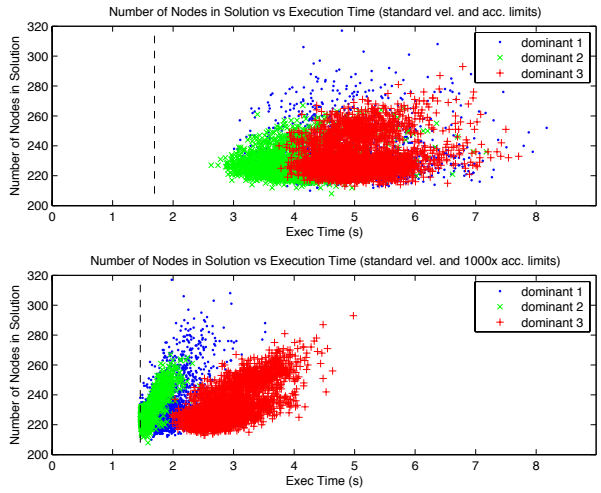

Figure 12: (top) Scatter plot of number of nodes in the solution path vs the total execution time, separated by dominant limb. (bottom) Same, but with 1000x the normal acceleration limit.

inant 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.

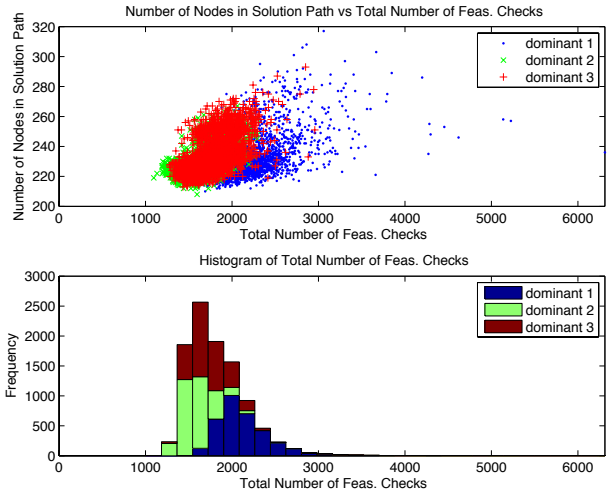

Figure 13: Total number of feasibility checks serves as a proxy for overall planning time.

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 13 and t_{exec} is drawn from the distributions shown in Figure 11, while t_{feas} is a constant that depends on the overall speed of the computer and software being used. Figure 14 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.

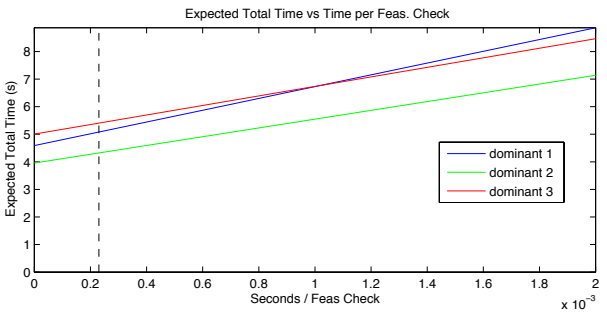

Figure 14: Expected total time depends on feasibility checking speed.

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 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. 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.

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.

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., 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., 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.

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

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.

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.